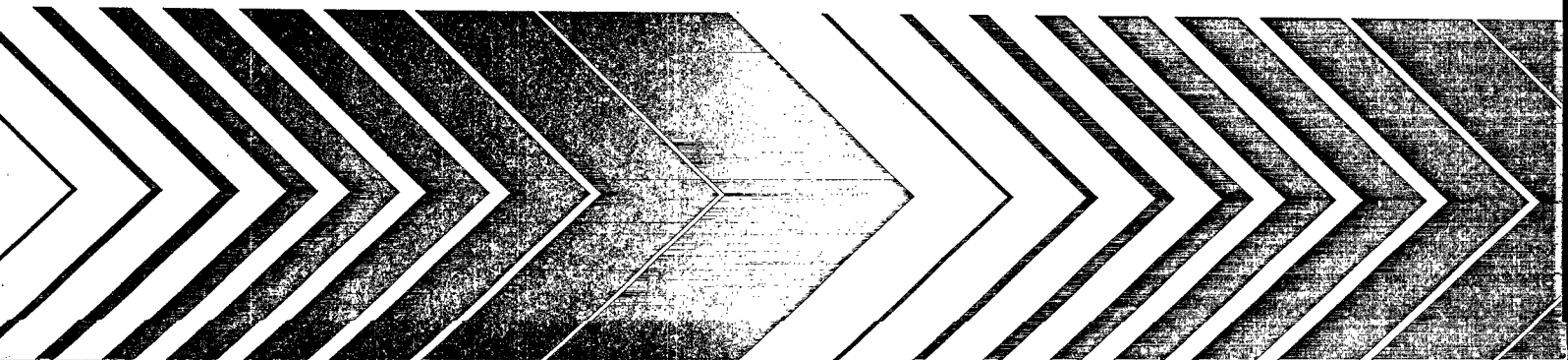
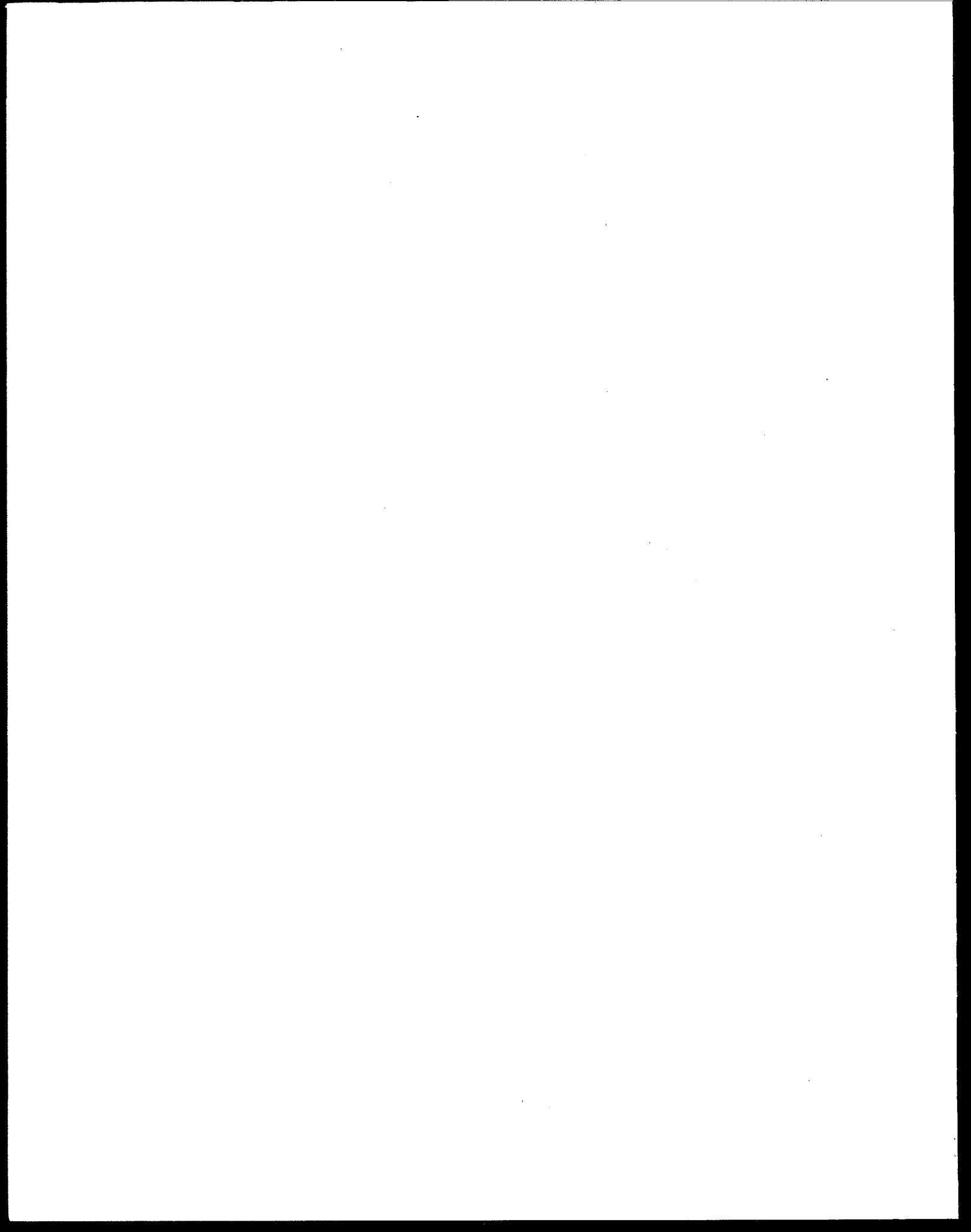




Michigan Soil Vapor Extraction Remediation (MISER) Model

A Computer Program to
Model Soil Vapor
Extraction and Bioventing of
Organic Chemicals in
Unsaturated Geological
Material





**MICHIGAN SOIL VAPOR EXTRACTION
REMEDICATION (MISER) MODEL:
A COMPUTER PROGRAM TO MODEL SOIL VAPOR EXTRACTION AND
BIOVENTING OF ORGANIC CHEMICALS IN
UNSATURATED GEOLOGICAL MATERIAL**

by

Linda M. Abriola, John Lang, and Klaus Rathfelder
Department of Civil and Environmental Engineering
The University of Michigan
Ann Arbor, Michigan 48109

Cooperative Agreement CR-822017

Project Officer

Jong Soo Cho
Subsurface Protection and Remediation Division
National Risk Management Research Laboratory
Ada, Oklahoma 74820

NATIONAL RISK MANAGEMENT RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CINCINNATI, OH 45268



DISCLAIMER

The U.S. Environmental Protection Agency through its Office of Research and Development partially funded and collaborated in the research described here under assistance agreement number CR-822017 to the University of Michigan. It has been subjected to the Agency's peer and administrative review and has been approved for publication as an EPA document. Mention of trade names or commercial products does not endorsement or recommendation for use.

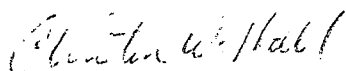
When available, the soft ware in this document is supplied on an "as-is" basis without guarantee or warranty of any kind, express or implied. Neither the United States Government (United States Environmental Protection Agency, Robert S. Kerr Environmental Research Center), the University of Michigan, nor any of the authors accept any liability resulting from use of this software.

FOREWORD

The U.S. Environmental Protection Agency is charged by Congress with protecting the Nation's land, air, and water resources. Under a mandate of national environmental laws, the Agency strives to formulate and implement actions leading to a compatible balance between activities and the ability of natural systems to support and nurture life. To meet these mandates, EPA's research program is providing data and technical support for solving environmental problems today and building a science knowledge base necessary to manage our ecological resources wisely, understand how pollutants affect our health, and prevent or reduce environmental risks in the future.

The National Risk Management Research Laboratory is the Agency's center for investigation of technological and management approaches for reducing risks from threats to human health and the environment. The focus of the Laboratory's research program is on methods for the prevention and control of pollution to air, land, water, and subsurface resources; protection of water quality in public water systems; remediation of contaminated sites and ground water; and prevention and control of indoor air pollution. The goal of this research effort is to catalyze development and implementation of innovative, cost-effective environmental technologies; develop scientific and engineering information needed by EPA to support regulatory and policy decisions; and provide technical support and information transfer to ensure effective implementation of environmental regulations and strategies.

Soil vapor extraction (SVE) and bioventing (BV) are effective and widely used in-situ remediation techniques for unsaturated soils contaminated with organic compounds, primarily petroleum hydrocarbons. Despite the effectiveness and flexibility of SVE and BV technologies, the efficiency and degree of success of those systems is controlled by a combination of physical, chemical, and biological factors. The dynamics of those interrelated processes is often incompletely understood, and consequently the performance and efficiency of specific SVE/BV systems is generally difficult to predict. This report describes the development of a numerical model for the simulation of physical, chemical, and biological interactions occurring in SVE and BV systems. The model can be used as a research tool for studying and elucidating dynamics in SVE and BV systems.



Clinton W. Hall, Director
Subsurface Protection and Remediation Division
National Risk Management Research Laboratory

ABSTRACT

This report describes the formulation, numerical development, and use of a multiphase, multicomponent, biodegradation model designed to simulate physical, chemical, and biological interactions occurring primarily in field scale soil vapor extraction (SVE) and bioventing (BV) systems. The model is entitled the Michigan Soil Vapor Extraction Remediation Model, or MISER. MISER solves the governing flow and transport equations in two space dimensions - either a cross sectional x - z domain, or an axisymmetrical r - z domain for simulating radial flow to a single well. A standard Galerkin finite element approach with linear triangular elements is employed. The coupled nonlinear equations are solved using a modular, set-iterative solution algorithm. In this approach the sets of flow, transport, and biodegradation equations are decoupled within the simulator and solved separately. The set-iterative approach substantially reduces the size of solution matrices and provides increased flexibility. Features of the model include: the ability to simulate multiphase flow, including water table coning; the simulation of multicomponent transport processes, incorporating fate-limited interphase exchange in processes of volatilization and dissolution of an entrapped organic liquid, interphase exchange between the mobile gas and aqueous phases, sorption, and biophase update; and the simulation of multicomponent biodegradation kinetics and microbial population dynamics.

This report is divided into 6 sections. Section 1 provides an overview of SVE and BV systems and a review of existing models. Section 2 presents the conceptual formulation of MISER and the associated mathematical representation for flow, transport and biotransformation processes. Section 3 describes the numerical solution approach and Section 4 presents the results of model verification analyses. Description and usage of the model is provided in Section 5, and example SVE and BV simulations are described in Section 6.

This report was submitted in fulfillment of CR-822017 by the University of Michigan under the partial sponsorship of the U.S. Environmental Protection Agency. This report covers a period from September 1993 to March 1996 and work was completed as of March 1996.

CONTENTS

DISCLAIMER	ii
FOREWORD	iii
ABSTRACT	iv
LIST OF FIGURES	x
LIST OF TABLES	xiii
LIST OF ABBREVIATIONS AND SYMBOLS	xv
1 INTRODUCTION	1
1.1 OVERVIEW AND PURPOSE	1
1.2 BACKGROUND AND LITERATURE REVIEW	2
1.2.1 Limits of Applicability	2
1.2.2 Rate Limited Mass Transfer	3
1.2.3 Biodegradation Rates	3
1.2.4 Temperature	4
1.2.5 Previous Models	5
1.3 MODEL FEATURES	6
2 MODEL FORMULATION AND THEORETICAL DEVELOPMENT	8
2.1 CONCEPTUAL MODEL	8
2.2 PHASE MASS BALANCE EQUATIONS	11
2.3 COMPONENT MASS BALANCE EQUATIONS	13
2.4 INTERPHASE MASS TRANSFER	15
2.4.1 Equilibrium Partitioning	15

2.4.2	Rate Limited Interphase Mass Transfer	17
2.4.3	Biotransformations	19
2.5	CONSTITUTIVE RELATIONS	21
2.5.1	Capillary Pressure	21
2.5.2	Relative Permeability	23
2.5.3	Gas phase density	23
2.5.4	Liquid Phase Density	23
2.5.5	Gas phase viscosity	24
2.5.6	Aqueous Phase Viscosity	24
2.5.7	Hydrodynamic Dispersion	24
2.5.8	Matrix Compressibility	25
3	NUMERICAL DEVELOPMENT	26
3.1	FINITE ELEMENT APPROACH	26
3.2	SOLUTION OF THE PHASE MASS BALANCE EQUATIONS	28
3.2.1	Pressure Based Formulation	28
3.2.2	Trial Functions and Weighted Residual Equations	29
3.2.3	Capacity Coefficients	31
3.3	MATERIAL PROPERTIES	31
3.4	VELOCITY EQUATIONS	32
3.5	COMPONENT MASS BALANCE EQUATIONS	34
3.5.1	Weighted Residual Equations in Cartesian Coordinates	35
3.5.2	Mass Exchange Terms	38
3.6	BIOLOGICAL REACTIONS	39
3.7	NAPL SATURATION	40
3.8	AXISYMMETRIC COORDINATES	40
3.9	TIME DISCRETIZATION	41
3.10	TIME STEP CONTROL	41
3.11	BOUNDARY CONDITIONS	42
3.11.1	Phase Mass Balance Boundary Conditions	42

3.11.2	Component Mass Balance Boundary Conditions	43
3.11.3	Extraction Wells	44
3.11.4	Treatment of Injection Wells	45
3.12	ITERATION SCHEME	46
3.13	SOLVER	46
4	MODEL VERIFICATION	49
4.1	MATERIAL BALANCE CALCULATION	49
4.1.1	Phase Material Balance	49
4.1.2	Component Material Balance	50
4.1.3	Calculation of Mass Balance Error	50
4.2	VERIFICATION OF THE PHASE MASS BALANCE SOLUTIONS	52
4.2.1	Comparison with One Dimensional Richards Equation	52
4.2.2	Comparison with Two Dimensional Richards Equation	54
4.2.3	Comparison with Quasi Analytical Solutions for Unsteady Radial Flow of Gas	57
4.3	VERIFICATION OF THE COMPONENT MASS BALANCE SOLUTIONS	60
4.3.1	Comparison with One Dimensional Analytical Solutions	60
4.3.2	Comparison with Two Dimensional Analytical Solutions	61
4.3.3	Verification of Biokinetics	62
4.3.4	Verification of Interphase Exchange	64
4.4	VERIFICATION OF THE COUPLED PHASE AND COMPONENT MASS BALANCE SOLUTIONS	65
5	PROGRAM DESCRIPTION AND SIMULATION SETUP	68
5.1	CODE DESCRIPTION	68
5.2	PROGRAM MODULES	68
5.3	ARRAY DIMENSIONS AND PROGRAM VARIABLES	68
5.4	DESCRIPTION OF INPUT FILES AND INPUT DATA	69
5.4.1	Data Block A – Input/Output Files and Control Options	69
5.4.2	Data Block B – General Model Control Options	69
5.4.3	Data Block C – Time Step and Iteration Control Parameters	69

5.4.4	Data Block D – Grid Information and Control Options	69
5.4.5	Data Block E – Component Chemical Properties	70
5.4.6	Data Block F – Mass Transfer Coefficients	70
5.4.7	Data Block G – Material Property Block Information	70
5.4.8	Data Block H – Sorption Parameter Data	70
5.4.9	Data Block I – Biological Parameter Data	71
5.4.10	Data Block J – Phase Parameter Data	71
5.4.11	Data Block K – Temperature Parameter Data	71
5.4.12	Data Block L – Output Control Parameters	72
5.4.13	Data Block M – Restart Identifier	72
5.4.14	Data Block N – Initial Pressure Conditions	72
5.4.15	Data Block O – Velocity Computation	73
5.4.16	Data Block P – Organic Liquid Saturation and Composition	73
5.4.17	Data Block Q – Oxygen and Nutrient Initial Conditions	73
5.4.18	Data Block R – Boundary Conditions	73
5.4.19	Data Block S – Extraction/Injection Well Conditions	74
5.4.20	Data Block T – Velocity Boundary Conditions	74
5.5	DESCRIPTION OF OUTPUT FILES	74
5.5.1	Main Output File - ‘Outpre.out’	74
5.5.2	Convergence History and Runtime Information Output File - ‘Outpre.cnv’	75
5.5.3	Error Message Output File - ‘Outpre.err’	75
5.5.4	Mass Balance Output File - ‘Outpre.mb’	75
5.5.5	Contour Plot Output File - ‘Outpre.con’	76
5.5.6	Time Series Plot Output File - ‘Outpre.plt’	76
5.5.7	Restart Output File - ‘Outpre.rst’	76
6	DEMONSTRATION OF MISER	105
6.1	SOIL VAPOR EXTRACTION	106
6.2	BIOVENTING	107
6.3	FIELD SCALE BIOVENTING	109

A	ELEMENT MATRICES FOR THE SIMULTANEOUS SOLUTION OF THE PHASE MASS BALANCE EQUATIONS	11
B	ELEMENT MATRICES FOR THE SOLUTION OF DARCY'S LAW EQUATION	12
C	ELEMENT MATRICES FOR THE SEQUENTIAL SOLUTION OF THE COMPONENT MOLE BALANCE EQUATIONS	13
D	ELEMENT MATRICES FOR THE SOLUTION OF THE ORGANIC PHASE MASS BALANCE EQUATION	129
E	Description of Major Variables	131
F	EXAMPLE MAKE FILE	139
G	EXAMPLE DATA FILES	140
H	SOURCE CODE LISTING	151
	REFERENCES	230

LIST OF FIGURES

1.1	Basic SVE/BV system configuration.	2
2.1	Conceptual model of the soil system composition.	9
2.2	Conceptual model of interphase mass transfer pathways.	10
3.1	Triangular element in global and transformed coordinates (after <i>Lapidus and Pinder</i> , 1982).	27
3.2	Variable representation in MISER.	32
3.3	Representation of an extraction well in the discretized domain.	44
4.1	Computational grid used for the numerical solution of the one dimensional Richards equation.	53
4.2	Comparison of numerical and analytical solutions for the one dimensional Richards equation. Simulation time = 6 hours; convergence tolerance = 1×10^{-4}	54
4.3	Domain configuration used in two dimensional (2D) flow simulations for comparison to SWMS_2D.	55
4.4	Numerical grid used in two dimensional flow simulations for comparison to SWMS_2D.	56
4.5	Simulated volumetric moisture content in the homogeneous domain at time 12 hrs (MISER = solid line; SWMS_2D = dashed line).	56
4.6	Simulated volumetric moisture content in the layered domain at time 6 hrs (MISER = solid line; SWMS_2D = dashed line).	57
4.7	Simulated volumetric moisture content in the layered domain at time 12 hrs (MISER = solid line; SWMS_2D = dashed line).	57
4.8	Domain configuration used in two dimensional flow simulations for comparison to quasi analytical solutions for radial gas flow.	58
4.9	Comparison of quasi analytical and numerical solutions for two dimensional radial gas flow in a uniform soil with $k = 1 \times 10^{-11} \text{ m}^2$	59
4.10	Comparison of quasi analytical and numerical solutions for one dimensional radial gas flow in a uniform soil with $k = 1 \times 10^{-14} \text{ m}^2$	59
4.11	Computational grid used for the numerical solution of the one dimensional transport equation with and without advection.	60
4.12	Comparison of MISER with the one dimensional analytical solution for diffusion driven transport.	61

4.13	Comparison of MISER with the one dimensional Ogata and Banks analytical solution for transport with dispersion and constant advection.	61
4.14	Comparison of MISER (solid lines) with a two dimensional analytical transport solution (dashed lines) at 4000 sec. Contours of normalized concentration are from left to right: 0.8, 0.6, 0.4, 0.2, 0.1, and 0.01.	62
4.15	Comparison of MISER (solid lines) with a 2D analytical transport solution including first order decay (dashed lines) at 4000 sec. Contours of normalized concentration are from left to right: 0.8, 0.6, 0.4, 0.2, 0.1, and 0.01.	63
4.16	Comparison of substrate profiles along a one dimensional column by MISER (lines) and a one dimensional numerical solution (discrete points) for biodegradation by <i>Moltz et al</i> , [1986]. 64	64
4.17	Comparison of MISER (solid lines) with a two dimensional analytical solution including linear equilibrium sorption (dashed lines) at 4000 sec. Contours of normalized concentration are from left to right: 0.8, 0.6, 0.4, 0.2, 0.1, and 0.01.	65
4.18	Comparison of MISER (lines) with a one dimensional column experiment (discrete points) for multicomponent organic liquid (benzene, TCE, toluene) volatilization under equilibrium conditions.	65
4.19	Comparison of predicted solute concentrations at time 6 hrs (MISER = solid line; SWMS_2D = dashed line).	66
4.20	Comparison of predicted solute concentrations at time 12 hrs (MISER = solid line; SWMS_2D = dashed line).	67
6.1	Problem depiction used in example simulations. Contours show the initial organic liquid distribution. The contour interval is 0.005 with levels increasing inward.	105
6.2	Predicted organic liquid saturation distribution in SVE simulations with intermediate mass transfer rates. The contour interval is 0.005 with levels increasing inward.	107
6.3	Toluene sorbed (ppm) at 20 and 100 days.	107
6.4	Toluene removal versus time.	108
6.5	Predicted organic liquid saturation (%) at selected times in the example BV simulation.	109
6.6	Predicted biomass distribution ($\text{g/l} \times 10^{-3}$) at selected times in the example BV simulation.	109
6.7	Simulation domain used in the field scale bioventing demonstration simulation.	110
6.8	Initial conditions used for the field scale bioventing demonstration simulation.	113
6.9	Predicted organic liquid distributions at specified times for the field scale bioventing demonstration simulation.	114
6.10	Predicted benzene substrate distributions at specified times for the field scale bioventing demonstration simulation.	115
6.11	Predicted biomass distributions at specified times for the field scale bioventing demonstration simulation.	116

6.12 Predicted oxygen distributions at specified times for the field scale bioventing demonstration simulation. 117

LIST OF TABLES

3.1	Summary of mass transfer expressions.	36
3.2	Summary of lumped mass exchange and bioreaction coefficients.	37
3.3	Summary of numerical scheme in MISER.	47
4.1	Comparison of global mass balance errors from numerical solutions of the one dimensional Richards equation at time 6 hrs.	54
4.2	Soil properties used in two dimensional flow simulations for comparison to SWMS_2D. . .	55
4.3	Soil properties used in two dimensional flow simulations for comparison to quasi-analytical solutions for radial gas flow.	58
5.1	MISER program modules.	77
5.2	Selected parameter variables defined in the include file 'dimen.inc.'	79
5.3	Description of input and output files.	80
5.4	Input Data in Block A – Input/Output Files and Control Options.	81
5.5	Input Data in Block B – General Model Control Options.	83
5.6	Input Data Block C – Time Step and Iteration Control Information.	84
5.7	Input Data Block D – Grid Information and Control Options Information.	85
5.8	Input Data Block E – Component Chemical Properties.	86
5.9	Input Data Block F – Mass Transfer Coefficients.	87
5.10	Input Data Block G – Material Property Block Information.	88
5.11	Input Data Block H – Sorption Parameter Data.	89
5.12	Input Data Block I – Biological Parameter Data.	90
5.13	Input Data Block J – Phase Parameter Data.	91
5.14	Input Data Block K – Temperature Parameter Data.	92
5.15	Input Data Block L – Output Control parameters.	93
5.16	Input Data Block M – Restart Identifier.	96
5.17	Input Data Block N – Initial Pressure Conditions.	97
5.18	Input Data Block O – Velocity Computation.	98

5.19	Input Data Block P – Organic Liquid Saturation and Composition.	99
5.20	Input Data Block Q – Oxygen and Nutrient Initial Conditions.	100
5.21	Input Data Block R – Boundary Conditions.	101
5.22	Input Data Block S – Extraction/Injection Well Conditions.	103
5.23	Input Data Block T – Velocity Boundary Conditions.	104
6.1	Soil properties used in example SVE and BV simulations.	106
6.2	Mass transfer coefficients used to simulate an SVE system.	106
6.3	Biotransformation parameters used in an example BV simulation.	109
6.4	Soil parameters used in the field scale bioventing demonstration simulation.	111
6.5	Fluid properties used in the field scale bioventing demonstration simulation. All values are for 20 °C.	111
6.6	Mass exchange and biokinetic parameters used in the field scale bioventing demonstration simulation.	112

LIST OF ABBREVIATIONS AND SYMBOLS

$a_{\alpha\beta}$	specific contact area between the α and β phases [L^{-1}]	$E_{\alpha\beta c}^*$	interphase mass transfer rate of component c to phase α from phase β per pore volume [$ML^{-3}T^{-1}$]
A	general mass matrix	f, F	general RHS vectors
A^e	area of a triangular element [L^2]	F_{cl}	use coefficient of component c with substrate l degradation [-]
A_{x_i}	cross-sectional area associated with node i [L^2]	f_{oc}	soil organic carbon fraction [-]
A_i	subarea of a triangular element [L^2]	F_b	boundary flux in material mass balance calculation [$ML^{-3}T^{-1}$]
b	Klinkenberg parameter [$ML^{-1}T^{-2}$]	F_e	exchange flux in material mass balance calculation [$ML^{-3}T^{-1}$]
B	general stiffness matrix	F_r	source/sink in material mass balance calculation [$ML^{-3}T^{-1}$]
$B_{1,2,3}$	material balance error measures [%]	F_s	rate of change in material mass storage for balance calculation [$ML^{-3}T^{-1}$]
B_α	net mole biological transformation rate in phase α per medium volume [$\text{mole } L^{-3}T^{-1}$].	F_μ	biological reaction rate in material mass balance calculation [$ML^{-3}T^{-1}$]
$B_{\alpha c}$	mole biological transformation rate of component c in phase α per medium volume [$\text{mole } L^{-3}T^{-1}$].	$\bar{F}_{\alpha c}$	lumped RHS mass transfer coefficient for component c in phase α [$\text{mole } L^{-3}T^{-1}$]
B_α^*	net mass biological transformation rate in phase α per volume of phase α [$ML^{-3}T^{-1}$].	g	gravitational acceleration vector [LT^{-2}]
$B_{\alpha c}^*$	mass biological transformation rate of component c in phase α per volume of phase α [$ML^{-3}T^{-1}$].	g_x	horizontal component of g [LT^{-2}]
$C_{\alpha c}$	mass concentration of component c in phase α [ML^{-3}]	g_z	vertical component of g [LT^{-2}]
C_p	capacity coefficient $\partial S_a / \partial P_c$ [$M^{-1}LT^2$]	G_α	α phase compressibility factor [-]
$D_{\alpha c}^h$	hydrodynamic dispersion tensor of component c in phase α [L^2T^{-1}]	h_{g_j}	equivalent gas phase head [L]
$D_{\alpha c}^m$	binary molecular diffusion coefficient of component c in phase α [L^2T^{-1}]	I_c	component c inhibition function [-]
E	general RHS vector	$I_{max,min}$	biomass inhibition functions [-]
E_α	net interphase mole transfer rate to phase α from all contiguous phases per pore volume [$\text{mole } L^{-3}T^{-1}$]	I_s	saturation inhibition function [-]
$E_{\alpha c}$	net interphase mole transfer rate to phase α of component c from all contiguous phases per pore volume [$\text{mole } L^{-3}T^{-1}$]	$J_{\alpha c}^d$	mass flux of component c in phase α by kinematic dispersion [$ML^{-2}T^{-1}$]
$E_{\alpha\beta c}$	interphase mole transfer rate of component c to phase α from phase β per pore volume [$\text{mole } L^{-3}T^{-1}$]	$J_{\alpha c}^m$	mass flux of component c in phase α by molecular diffusion [$ML^{-2}T^{-1}$].
E_α^*	net interphase mass transfer rate to phase α from all contiguous phases per pore volume [$ML^{-3}T^{-1}$]	k	intrinsic permeability tensor [L^2]
$E_{\alpha c}^*$	net interphase mass transfer rate of component c to phase α from all contiguous phases per pore volume [$ML^{-3}T^{-1}$]	K_d	biomass decay rate [T^{-1}]
		k_g	effective gas phase permeability [L^2]
		k_l	maximum substrate l use rate [$\text{mole } M^{-1}T^{-1}$]
		$k_{r\alpha}$	relative permeability of phase α [-]
		K_{S_l}	half saturation constant for substrate l [-]
		$k_{\alpha c}$	overall mass transfer coefficient for component c controlled by phase α [LT^{-1}]

k_{∞}	gas phase permeability at a high pressure; equivalent to the liquid permeability [L^2]	t	time [T]
K_{H_c}	Henry's Law constant for component c [$ML^{-1}T^{-2}$]	T	temperature [$^{\circ}K$]
K_{oc_c}	organic carbon - normalized partition coefficient [ML^{-3}]	u	general dependent variable
$K_{s_{ac}}^c$	Freundlich parameter for component c [L^3M^{-1}] ⁿ	\hat{u}	trial function of u
$K_{\alpha\beta_c}$	lumped mass transfer coefficient for component c between the controlling phase α and phase β [T^{-1}]	V_{α}	pore velocity of phase α [LT^{-1}]
$K_{\alpha\beta_c}^e$	equilibrium partition coefficient for component c in phase α based on phase β [-]	W_i	weighting function
\bar{K}_{α_c}	lumped LHS mass transfer and biological reaction coefficient for component c in phase α [T^{-1}]	x	horizontal spatial coordinate [L]
$L(u)$	differential operator on u	$x_{\alpha_c}^{sol}$	component c aqueous phase solubility as a mole fraction [-]
L_j	local coordinate at node j	x_c^{max}	inhibitory mole fraction of component c [-]
$L_{\alpha\Gamma}$	thickness of stagnant boundary layer in the contacting α phase [L]	x_c^{min}	minimum detectable mole fraction of component c [-]
m	van Genuchten parameter [-]	x_{α_c}	mole fraction of component c in phase α [-]
M_{α}	molecular weight of phase α [M mole ⁻¹]	$x_{\alpha\beta_c}^e$	mole fraction of component c in phase α in equilibrium with phase β [-]
M_c	molecular weight of component c [M mole ⁻¹]	$x_{\alpha_c}^o$	mole fraction of component c in contacting phase α at the boundary [-]
n	unit normal vector	X	biomass [ML^{-3}]
n	total number of nodes in the solution domain; van Genuchten parameter [-]; Freundlich parameter [-]	$X_{max,min}$	maximum and minimum biomass [ML^{-3}]
n^e	number of nodes in an element	Y_l	biomass yield coefficient for substrate l degradation [M mole ⁻¹]
N^e	number of elements in the solution domain	z	vertical spatial coordinate [L]
N_i	linear basis or shape function for triangular elements	α_{ga}	van Genuchten parameter [LT^2M^{-1}]
P_c	capillary pressure [$ML^{-1}T^{-2}$]	α_i	coefficient of the linear basis function N_i
P_{g_c}	partial pressure of component c [$ML^{-1}T^{-2}$]	$\alpha_{L,T}$	longitudinal L and transverse T dispersivities [L]
P_{nw}	pressure of the nonwetting phase [$ML^{-1}T^{-2}$]	β_i	x -direction derivative of the linear basis function N_i [-]; $2A^e \frac{\partial N_i}{\partial x}$ [-]
P_{v_c}	vapor pressure of component c [$ML^{-1}T^{-2}$]	γ_{α_c}	activity coefficient of component c in phase α [-]
P_w	pressure of the wetting phase [$ML^{-1}T^{-2}$]	γ_i	z -direction derivative of the linear basis function N_i [-]; $2A^e \frac{\partial N_i}{\partial z}$ [-]
P_{α}	pressure of phase α [$ML^{-1}T^{-2}$]	Γ	computational boundary of domain
q_{α}	specific discharge of phase α [LT^{-1}]	δ_{kl}	Kronecker delta
Q	general RHS vector	ϵ_f	convergence criteria for mobile phase balance equations [-]
Q_{α}	total discharge of phase α [L^3T^{-1}]	ϵ_i	convergence criteria for immobile component balance equations [-]
r	radial spatial coordinate [L]	ϵ_m	convergence criteria for mobile component balance equations [-]
r_c	retardation factor for component c [-]	ϵ_o	convergence criteria for NAPL saturation [-]
\bar{r}^e	radial element centroid coordinate [L]	ϵ	residual
R_{α}	internal source/sinks of phase α [T^{-1}]	λ_{α}	mobility of phase α [$M^{-1}L^3T$]
\bar{S}_a	normalized aqueous phase saturation [-]	μ	dynamic viscosity [$ML^{-1}T^{-1}$]
S_{ra}	residual aqueous phase saturation [-]	μ_{α_c}	nonlinear Monod-type rate coefficient α [mole $L^{-3}T^{-1}$]
S_{α}	saturation of phase α [-]		

$\rho_{g^0}^*$	mass density of the uncontaminated gas phase [ML^{-3}]
ρ_s^*	bulk solid phase mass density [ML^{-3}]
ρ_α	molar density of phase α [mole L^{-3}]
ρ_α^*	mass density of phase α [ML^{-3}]
$\bar{\rho}_\alpha^*$	element average phase mass density [ML^{-3}]
τ_α	tortuosity of phase α [-]
ϕ	porosity [-]
$\Phi_{c,j}$	gas phase viscosity parameter [-]
θ	variable time weighting factor [-]
$\omega_{s,c}$	adsorbed mass of component c per mass of soil [-]
Ω	computational domain

subscripts

a	aqueous phase
A	nutrient
b	biophase
c	component
g	gas phase
i, j	nodes
k	direction
l	degradable substrate; direction
L	longitudinal direction
N_2	nitrogen
o	organic phase
O_2	oxygen
r	radial direction
s	solid phase
t	total

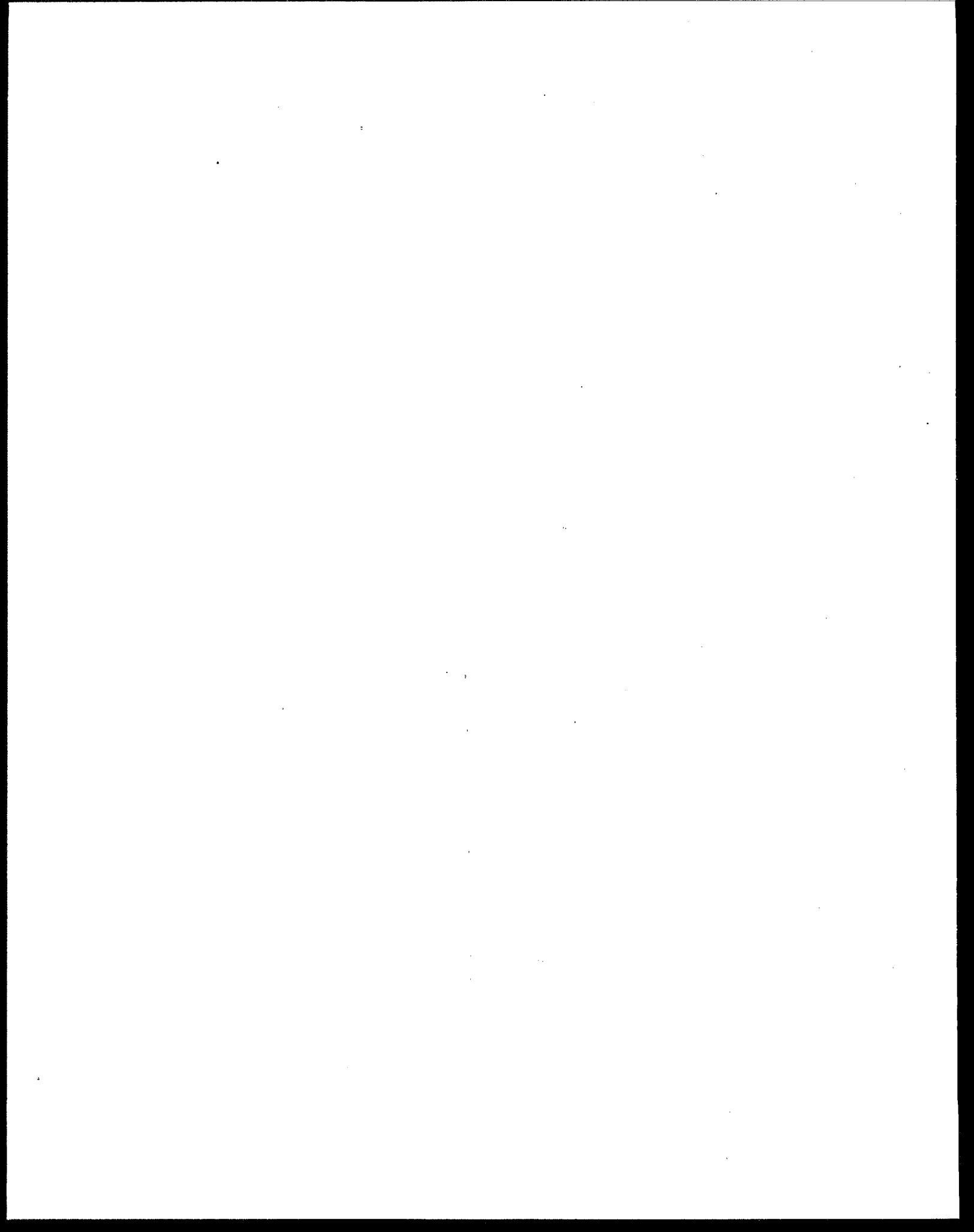
T	transverse direction
x	horizontal direction
w	water
z	vertical direction
α, β	phase (aqueous, gas, organic, solid)
γ	components of the organic liquid

superscripts

d	kinematic dispersion
e	element
h	hydrodynamic dispersion
k	iteration counter
L	mass lumped
m	molecular diffusion
o	initial time, conditions in contacting fluid, uncontaminated gas phase
t	time
$*$	mass based variable

abbreviations

BV	bioventing
LEA	local equilibrium assumption
LHS	left hand side
RHS	right hand side
SS	simultaneous solution method
SVE	soil vapor extraction



Section 1

INTRODUCTION

1.1 OVERVIEW AND PURPOSE

The migration and fate of nonaqueous phase liquid (NAPL) organic contaminants in the subsurface has been the subject of intensive investigation in the past few years. It is now generally recognized that the prevalence of NAPLs at contaminated sites is a significant impediment to aquifer restoration [Mackay and Cherry, 1989]. As a NAPL migrates through a porous formation a portion of the organic liquid is retained within the pores due to the action of capillary forces. Residual NAPL can vary from 5-40% of the pore volume, or on a volume basis, from 3-5 l/m³ in permeable soils up to 30-50 l/m³ in low permeable soils [Schwille, 1984; Hoag and Marley, 1986; Wilson et al., 1990]. These entrapped residuals cannot be mobilized by simple hydraulic flushing, and thus due to the low aqueous solubility of the NAPLs, the residuals may serve as long term sources of groundwater contamination. Conventional pump and treat remediation technologies have proven to be an ineffective and costly approach to aquifer restoration when NAPLs are present [Mackay and Cherry, 1989; Haley et al., 1991; National Research Council, 1994].

Based upon the limitations of conventional pump and treat methods, considerable effort is currently focused on the development of alternative remediation technologies. Soil vapor extraction (SVE) is an alternative remediation approach which targets the removal of volatile organic contaminants (VOCs) from the unsaturated zone. SVE involves the generation of advective vapor fluxes through the pores of the contaminated soil to induce transfer of VOCs to the air stream. Air flow is established by pumping from a system of vadose zone wells through which contaminant vapors are collected and transported above ground where they are treated if required, and discharged to the atmosphere (Figure 1.1). Since its development in the late 70's and early 80's [Texas Research Institute, 1980, 1984; Thornton and Wootan, 1982; Marley and Hoag, 1984; Crow et al., 1985, 1987] SVE applications have become widespread, with SVE now comprising up to 18% of selected remedies at Superfund sites [Travis and Macinnis, 1992]. The popularity of SVE technologies stems from their proven effectiveness for removing large quantities of VOCs from the soil, their cost competitiveness, and their relatively simple nonintrusive implementation. Numerous articles and reports document and describe SVE applications [e.g. Hutzler et al., 1989; Downey and Elliott, 1990; Gerbasi and Menoli, 1994; McCann et al., 1994].

The ability of SVE systems to enrich the unsaturated zone with oxygen and stimulate indigenous microorganisms to biodegrade organic contaminants was recognized in early feasibility studies [Thornton and Wootan, 1982; Texas Research Institute, 1984]. Enhanced biodegradation in the unsaturated zone was subsequently evaluated in laboratory treatability studies [Hinchee and Arthur, 1991; Kampbell and Wilson, 1991] and in monitored field applications [Miller, 1990; Hinchee et al., 1991; Dupont et al., 1991]. These studies helped spawn the development of engineered systems referred to as bioventing (BV) (Figure 1.1). BV is similar to SVE in that remediation is facilitated by advective vapor fluxes established through vadose zone wells. BV, however, differs fundamentally from SVE in that it is designed to maximize soil remediation by *in situ* biodegradation and to minimize contaminant volatilization and above ground recovery [Dupont,

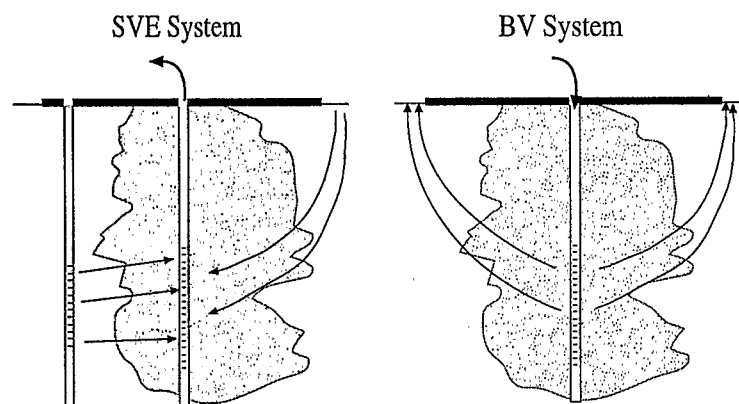


Figure 1.1: Basic SVE/BV system configuration.

1993; *Hinchee*, 1994]. BV is now recognized to be a highly effective and cost-competitive remediation alternative [*Miller et al.*, 1994] especially when treatment of off-gas is required [*Reisinger et al.*, 1994]

Because of the complexity of the processes influencing the performance of SVE/BV technologies, design and operation guidelines are frequently qualitative in nature based on experience or simple design rules [*Hutzler et al.*, 1989; *Dupont*, 1993; *Johnson et al.*, 1990, 1995]. Mathematical models are recognized as powerful tools that can be used to integrate and quantify the interaction of physical, chemical, and biological processes occurring in field scale SVE/BV systems. In addition to predicting potential mass removal, mathematical models can be used to explore alternative system designs and to investigate factors limiting successful remediation. This report describes the development of a comprehensive SVE/BV simulator, entitled the Michigan Soil Vapor Extraction Remediation Model, or MISER.

1.2 BACKGROUND AND LITERATURE REVIEW

1.2.1 Limits of Applicability

Reviews of SVE technologies indicate success of SVE is strongly correlated with contaminant volatility and the ability to generate advective gas fluxes through the contaminated soil [*Hutzler et al.*, 1989; *U.S. EPA*, 1991; *Pedersen and Curtis*, 1991; *Johnson et al.*, 1995; *Rathfelder et al.*, 1995]. SVE has been effectively applied for the removal or mass reduction of a wide variety of halogenated and nonhalogenated volatile and semivolatile organic compounds. It is not considered effective for organic compounds of low volatility, inorganic compounds, polychlorinated biphenyls (PCB), dioxins, organic pesticides, cyanides, and corrosives [*U.S. EPA*, 1991]. SVE is generally most effective in uniform soils with good conductivity, low moisture content, and low organic carbon content [*Fine and Yaron*, 1993]. Soils with low air permeability are more difficult to remediate [*U.S. EPA*, 1991], although applications in low permeability, fractured media have been reported [*Agrelot et al.*, 1985; *Gibson et al.*, 1993].

Reviews of BV design and applications [*Thomas and Ward*, 1992; *Dupont*, 1993; *Miller et al.*, 1994; *Litchfield*, 1993; *Hinchee*, 1994] indicate BV is most applicable for the removal of petroleum hydrocarbons. Chlorinated organic compounds have not been considered appropriate for BV due to their resistance to direct biodegradation, however, potential exists that these compounds can be cooxidized

during microbial growth on other hydrocarbons [English and Loehr, 1991; Speitel and Alley, 1991; Barbee, 1994; Wilson, 1994; Fuller et al., 1995]. In contrast to SVE, BV is not constrained by contaminant volatility and is therefore applicable to contaminants with moderate to low volatility [Hinchee, 1994]. Moreover, biodegradation rates are slower than volatilization processes for many hydrocarbons, and therefore BV may be well suited for application during the periods of long term, low level removal efficiency observed in traditional SVE systems. Consequently, integrated systems have been designed which employ SVE for rapid VOC recovery during early stages, followed by low cost, long term BV operations [Dupont et al., 1991; Nelson et al., 1994].

1.2.2 Rate Limited Mass Transfer

SVE systems characteristically exhibit large initial VOC recovery rates followed by a rapid drop in exhaust VOC concentrations and long term, low level removal efficiency [Crow et al., 1987; DiGiulio, 1992; Travis and Macinnis, 1992; McClellan and Gilham, 1992]. Diminished removal efficiency is attributed to several mechanisms which decrease the rate of VOC mass transfer to the mobile gas stream. Lighter, more volatile contaminant fractions are preferentially removed, leaving the heavier, less volatile components and decreasing remediation efficiency [Hoag et al., 1984; Fine and Yaron, 1993]. Secondly, removal efficiency decreases due to preferential removal of contaminants that are most accessible to the advective gas stream, leaving behind contaminants that have poor accessibility to the gas stream due to occlusion in intraaggregate or intraparticle regions [Brusseau, 1991; Gierke et al., 1992], or due to flow by passing of zones of low permeability [Kearl et al., 1991; Ho and Udell, 1992]. Diffusion controlled, rate limited interphase mass transfer, including processes of volatilization, dissolution, sorption and biotransformation, are also a potentially critical factor affecting removal efficiency [Brusseau, 1992; Armstrong et al, 1993, 1994; Wilkins et al., 1995].

Several studies indicate the rate of NAPL volatilization can be adequately modeled with the assumption of instantaneous local equilibrium [Hoag et al., 1984; Baehr et al., 1989; Berndtson and Bunge, 1991; Bloes et al., 1992; Hayden et al., 1994; Ho et al., 1994]. Other studies, however, have shown the rate of NAPL volatilization to be limited, even in relatively homogeneous materials, at high pore velocities [Rainwater et al., 1989; Kearl et al., 1991; Hoffman et al., 1993; Wilkins et al., 1995], or low constituent mole fractions [Hayden et al., 1994]. Hoffman et al. [1993] and Wilkins et al. [1995] found that measured effective mass transfer coefficients in sandy media could be well correlated with dimensionless parameters incorporating the vapor flux and mean grain size. Laboratory and field studies have also documented rate limited interphase mass transfer of VOCs between the soil water and mobile gas phases [Cho and Jaffe, 1990; Berndtson and Bunge, 1991; Gierke et al., 1992; McClellan and Gilham, 1992], and between the water and organic liquid phases [Powers et al., 1991, 1992, 1994]. Rate limited sorption/desorption on soil particles also plays an important role in the transport and retention of VOCs [Brusseau and Rao, 1989; Weber et al., 1991], and may limit bioavailability of organic substrates [Pignatello and Xing, 1996]. Recent experimental studies suggest that unsaturated zone sorption is more complex than that in the saturated zone, due to the presence of gas-liquid interfaces [Pennell et al., 1992].

1.2.3 Biodegradation Rates

Biodegradation rates in field BV systems have been assessed by the measurement of: carbon dioxide production [Hinchee and Arthur, 1991; Huesemann and Moore, 1994; van Eyk, 1994]; hydrocarbon

consumption rates [Kampbell and Wilson, 1991]; and oxygen consumption [Baehr et al., 1991; Ong et al., 1991; Hinchee, 1994; Huesemann and Moore, 1994]. Oxygen consumption is generally considered a more reliable measure of biodegradation and is typically measured by *in situ* respirometry tests [Hinchee, 1994].

Biodegradation rates in unsaturated soils have been observed in field and laboratory studies to be linked to the soil moisture content. The reason for this linkage is poorly understood. Results from laboratory studies generally show that higher rates of biotransformation occur at higher levels of soil moisture content [Fan and Scow, 1993] and that this dependence may be compound specific [Holman and Tsang, 1995]. Moisture addition in field BV applications has similarly produced an increase in degradation rates [Hinchee and Arthur, 1991; Zwick et al., 1995]. At other sites, moisture addition has been found to reduce degradation rates due to the constriction of air permeability and the resulting decrease in oxygen transport [Miller et al., 1994] or was reported to have no effect on biodegradation rates [Miller, 1990].

Enhanced biodegradation rates have also been observed in laboratory experiments following nutrient additions (e.g. nitrogen or phosphorus) indicating that nutrient limitations can constrain biodegradation rates [Hinchee and Arthur, 1991; Dupont, 1993; Baker et al., 1994; Fuller et al., 1995; Breedveld et al., 1995]. Nutrient addition in field applications is typically accomplished by water flooding [Nelson et al., 1994; Norris et al., 1994]. However, it is not clear whether there is substantial enhancement of biodegradation after nutrient addition in field studies [Miller, 1990; Miller et al., 1994; Leeson et al., 1995], partly because it is difficult to separate the effects of nutrient addition from moisture addition [Dupont et al., 1991].

Inhibition of biodegradation at high concentrations of inorganic nutrients [Baker et al., 1994] or high concentrations of organic substrates [Speitel and Alley, 1991; Huesemann and Moore, 1994; Mu and Scow, 1994] has been observed in laboratory studies. The significance of inhibition in field operations, however, has received limited attention. One study noted the possibility of observed substrate inhibition at low flow rates [Moore et al., 1995]. Thus inhibitory effects, if important, could be strongly linked to system operation and design.

Degradation rates in field conditions depend not only on the metabolic properties of the microbes, but also on the availability of substrates to the microorganisms. A host of laboratory and theoretical studies provide substantial evidence that diffusion controlled desorption of organic substrates can control the overall rate of biodegradation [Molz et al., 1986; Scow and Alexander, 1992; Scow and Hutson, 1992; Novak et al., 1993; Scow, 1993]. Thus, relatively slow rate limited desorption processes can in effect control bioremediation of subsurface systems [Mueller et al., 1989; Rijnaarts et al., 1990]. Exposure time and aging of the spill enhance the resistance to desorption and biodegradation [Novak et al., 1993; Pignatello and Xing, 1996; Fuller et al., 1995]. Consequently, modeling of biodegradation processes requires an accurate understanding and representation of sorption kinetics [Scow and Hutson, 1992; Pignatello and Xing, 1996].

1.2.4 Temperature

Subsurface temperature is an important environmental variable in SVE remediation processes. The efficiency of SVE remediation is strongly linked to the vapor pressures of the target compounds with a suggested lower limit on vapor pressure of approximately 1000 Pa [Pedersen and Curtis, 1991; Johnson et al., 1990]. Vapor pressures of organic compounds increase by a factor of approximately 3 or 4 with every 10°C rise in temperature. Experimental evidence also suggests that vapor sorption is inversely proportional to temperature [Goss, 1992]. Methods to increase SVE efficiency by increasing the subsurface temperature

have been attempted by heating and injection of ambient air [Sittler *et al.*, 1993] or by methods of steam injection [Falta *et al.*, 1992; Wilson and Clarke, 1992]. Several other novel approaches which integrate SVE with methods for soil heating have been considered [Downey and Elliott, 1990].

The effect of subsurface temperature on the design and operation of BV remediation efforts has received little attention [Sayles *et al.*, 1993, 1995]. It is generally accepted that soil temperatures at most sites are within the limits for microbial growth [Litchfield, 1993]. In Alaska, where soil temperatures are as low as 1°C, respiration rates comparable to those in temperate and subtropical regions were observed during summer months [Ong *et al.*, 1994; Kellems *et al.*, 1994]. The observed rates dropped slightly during the winter months with less effect noted at sites with either active or passive soil warming.

1.2.5 Previous Models

A number of mathematical models have been presented in the literature for the description of the SVE process. These vary greatly in level of complexity and in the processes included. The simplest models are analytical solutions for gas flow, intended to aid in the analysis of pneumatic pump tests and the design of SVE applications [Massmann, 1989; Johnson *et al.*, 1990; McWhorter, 1990; Baehr and Hult, 1991; Cho and DiGiulia, 1992; Massmann and Madden, 1994; Beckett and Huntley, 1994; Baehr *et al.*, 1995]. Numerical models of gas phase advection have also been developed for analysis of SVE systems [Welty *et al.*, 1991; Croise and Kaleris, 1992; Edwards and Jones, 1994] and BV systems [Mohr and Merz, 1995]. Because only the gas phase flow field is considered in this group of models, they are not capable of describing contaminant partitioning and migration, nor are they suitable when water movement is important. These models are generally applicable for screening purposes and simple design analyses.

A more complex group of models are those which combine constituent transport and steady state gas phase flow. Although transport processes are considered in this group of models, their applicability is restricted by assumptions on the flow field, partitioning mechanism, or domain configuration. They are, in general, not adequate for comprehensive simulation of field scale SVE/BV system, but are suitable for behavior assessment and screening purposes. The simplest models in this group couple steady state flow fields with analytical transport models [Roy and Griffin, 1991; Zaidel and Russo, 1993]. Other models incorporate steady state flow fields and the numerical solution of transport equations [Massmann and Farrier, 1992; Johnson *et al.*, 1990; Wilson *et al.*, 1988]. The model by Wilson *et al.* [1988] is extended in a series of papers to explore: the effects of impermeable caps, permeability and evaporative cooling [Gannon and Wilson, 1989]; anisotropic permeability [Mutch and Wilson, 1990]; variable permeability and soil moisture content [Gomez-Lahoz *et al.*, 1991]; system geometry [Rodriguez-Maroto *et al.*, 1991]; and spatially variant permeability [Roberts and Wilson, 1993]. Other modifications to this model have enabled the consideration of: rate limited mass transfer from fractured bedrock [Wilson, 1990] or areas of low permeability [Rodriguez-Maroto and Wilson, 1991; Osejo and Wilson, 1991]; Raoult's Law behavior of organic contaminant mixtures [Kayano and Wilson, 1992]; and departures from Darcy's Law [Clarke *et al.*, 1993]. Other models coupling steady state flow fields and numerical transport solutions have been extended to include complexities introduced by soil heterogeneities [Baehr *et al.*, 1989; Benson *et al.*, 1993; Joss, 1993].

Several numerical models incorporating transient single phase gas flow and constituent transport have been developed. Two dimensional finite element models with transport of a single volatile organic species have been developed [Metcalf and Farquhar, 1987; Stephanotos, 1988]. More complex one dimensional models encompassing single phase gas flow have been developed for analysis of nonequilibrium interphase

exchange [Brusseau, 1991; Armstrong *et al.*, 1994] and multicomponent, nonisothermal conditions [Lingineni and Dhir, 1992]. A one dimensional model developed by Gierke *et al.* [1990] includes both mobile gas and mobile aqueous phases. A two dimensional numerical model incorporating transient single phase gas flow and multi component compositional transport was developed by Rathfelder *et al.* [1991]. Nonequilibrium mass transfer was also explored.

Few models have appeared in the literature which can simulate multiphase flow, including gas and aqueous phase advection, multicomponent transport, and interphase mass exchange [Abriola, 1984, 1988; Abriola and Pinder, 1985]; . A two phase (air-water) two dimensional finite element simulator was presented by Stephamatos [1988] for the transport of a single volatile organic species. Baehr *et al.* [1989] presented a one dimensional multicomponent transport model which could predict vapor flux in a three phase (air-NAPL-water) system. Both of these models assumed equilibrium mass transfer between phases. Several more advanced models incorporate the effects of rate limited interphase mass exchange [Reeves and Abriola, 1988, 1994; Sleep and Sykes, 1989; Falta *et al.*, 1989], however, these models have not been extensively applied to SVE. Additionally, none of these models incorporates biotransformations of contaminants.

Existing models capable of simulating bioremediation processes have focused primarily on transformations and transport in the saturated zone [e.g. Chiang *et al.*, 1989]. For saturated transport models which incorporate microbial growth and transport and uptake of contaminants, nutrients, and electron acceptors, two general approaches have been used to represent microbial activity. In the first approach kinetic expressions describing the microbial consumption of a component are incorporated directly into the transport equation as a macroscopic sink term [e.g. Sykes *et al.*, 1982; Borden and Bedient, 1986; Frind *et al.*, 1990; Sleep and Sykes, 1991]. Here the bulk phase concentration of a component controls microbial consumption. In the second approach, the potential for diffusion limited transport to the biophase is accounted for by employing a macroscopic Fick's Law expression to represent the sink term [Molz *et al.*, 1986; Widdowson *et al.*, 1988; Baveye and Valocchi, 1989; Kinzelbach *et al.*, 1991; Chen *et al.*, 1992]. Of the models presented above, only Chen *et al.*, [1992] includes interphase mass transfer and transport within the gas phase. However, in this model, equilibrium interphase partitioning is assumed, residual NAPL is not considered, and transport in the gas phase is by diffusion only. The model of Sleep and Sykes [1991] includes advective flow of three fluid phases, however, no mass transfer limitations are considered, including mass transfer to the biophase.

1.3 MODEL FEATURES

A review of the relevant literature indicates that SVE/BV performance is influenced by a variety of interrelated and spatially dependent physical, chemical, and biological processes. It points to the need for a flexible simulator which can accommodate multiphase multicomponent transport and potential mass transfer limitations. Such a model must link microbial degradation with a microbial population and availability of substrates, electron acceptor, and nutrients.

The overall objective of this project is the development of a comprehensive numerical model for the simulation of multiphase flow, compositional transport, and biodegradation processes, occurring primarily in SVE and BV systems. Features in MISER include:

- the simulation of both cross sectional x - z and axisymmetric r - z domains;

- the ability to simulate the simultaneous flow of aqueous and gas phases resulting from natural and induced processes, such as: applied stresses at vadose zone extraction/injection wells, natural and artificial moisture infiltration, and density driven gas phase advection;
- the simulation of multicomponent transport processes including multicomponent organic substrates, an electron acceptor, and a limiting nutrient;
- the incorporation of rate limited interphase exchange including processes of volatilization, dissolution, sorption, and biophase uptake; and
- the simulation of multicomponent biodegradation kinetics and microbial population dynamics.

While MISER has the capability to consider both the unsaturated and saturated zones, it is not designed to simulate remediation processes primarily directed at the saturated zone such as pump and treat or air sparging. MISER is also not capable of simulating organic liquid migration nor is hysteresis considered in the movement of the gas and aqueous phases. MISER is designed to simulate aerobic microbial processes by a single population which can metabolize multiple substrates.

Section 2

MODEL FORMULATION AND THEORETICAL DEVELOPMENT

2.1 CONCEPTUAL MODEL

The following conceptual model is formulated and used as the framework for development of MISER.

Three fluid phases are modeled: (1) an entrapped organic liquid; (2) a mobile gas phase; and (3) a mobile aqueous phase. Because the organic liquid is assumed to be immobile, only the chemical and physical processes affecting the disappearance of the entrapped organic are modeled in this approach. The initial spatial distribution and composition of residual organic liquid are user defined inputs. The gas and aqueous phases are considered mobile and can flow simultaneously in response to applied stresses at extraction/injection wells, and to density gradients arising from spatial variation in phase composition.

A compositional modeling approach is employed. The transport and transformation of the following chemical species are modeled: the components comprising the organic liquid contaminant; one electron acceptor (oxygen); nitrogen (the major component of air); water (including water vapor); and one limiting nutrient (e.g. ammonia). A schematic of the conceptualized composition of the soil system is shown in Figure 2.1.

The organic liquid contaminant is considered as a mixture of γ components. Partitioning of gas and aqueous phase constituents into the organic liquid is assumed negligible. The composition of the organic liquid can vary in space and time due to mass exchange into adjacent phases. The components of the organic liquid may dissolve into the aqueous phase, but are assumed to be sparingly soluble. This assumption is valid for a wide variety of organic components at typical environmental conditions [Schwarzenbach *et al.*, 1993]. Oxygen and the limiting nutrient may also partition into the aqueous phase. Based upon available laboratory information, it is assumed that the microbes can metabolize only from the aqueous phase. To account for possible rate limited uptake by the microbes, the biophase is envisioned as a subset of the aqueous phase as depicted in Figure 2.1. The gas phase is assumed to be comprised of nitrogen and oxygen (i.e. the two major components of air), water vapor, the volatile components of the entrapped organic liquid, and the limiting nutrient. Water vapor is included so that drying effects caused by the application of SV, can be predicted. Although carbon dioxide has been monitored as an indicator of biological activity [Dupont *et al.*, 1991], this component is not modeled due to the complex geochemical considerations which impact the concentration of carbon dioxide in a soil systems. Sorption to the solid phase is limited to components of the organic liquid.

Mass transfer expressions are incorporated into the model to simulate rate limited mass exchange between phases. These expressions are used to model volatilization and dissolution of the entrapped organic liquid, mass exchange between the aqueous and gas phases, rate limited sorption, and rate limited transport to the biophase. This modeling approach is schematically illustrated in Figure 2.2.

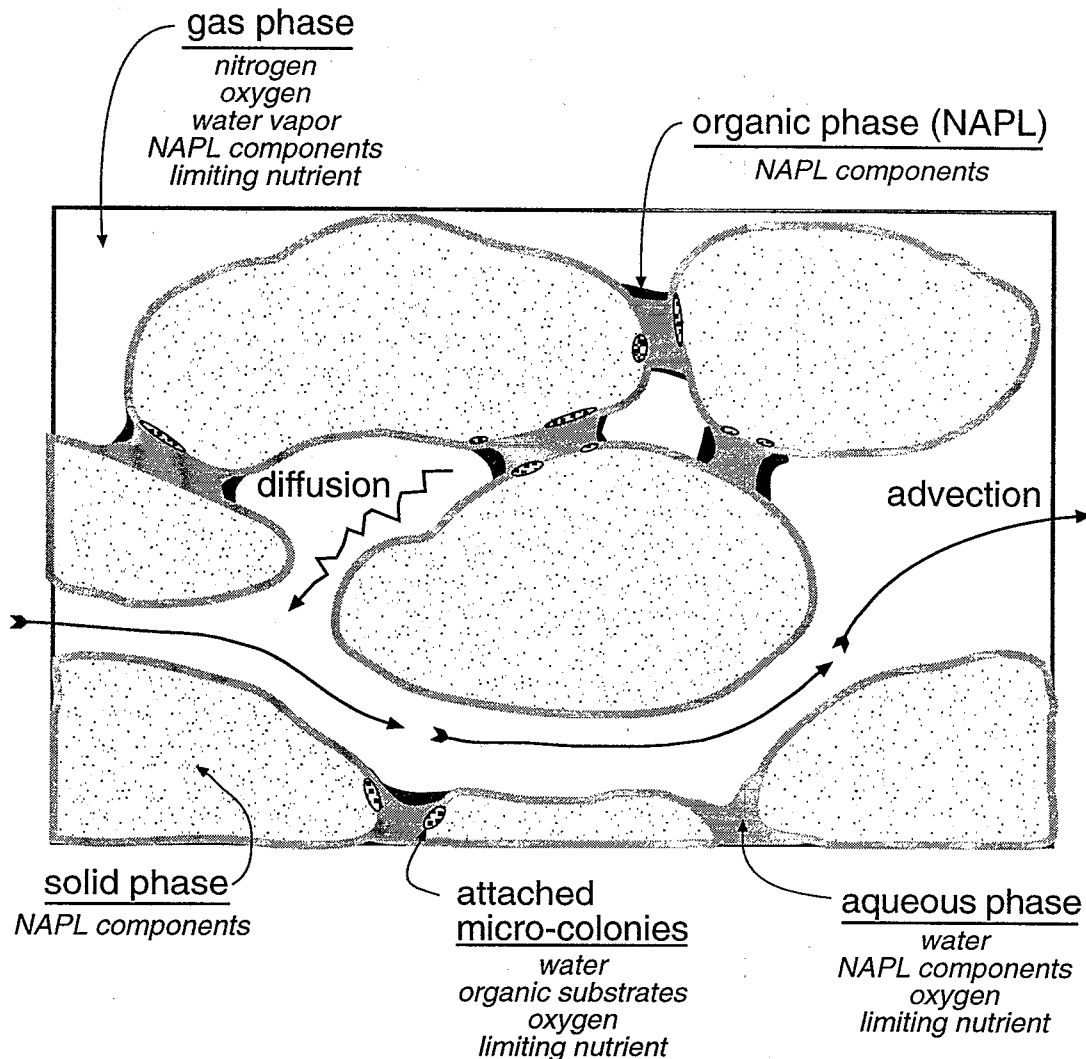


Figure 2.1: Conceptual model of the soil system composition.

The conceptual distribution of fluids in the soil system is shown in Figure 2.1 and is based on following assumptions: water is the preferential wetting fluid; gas is always the non-wetting fluid; and the organic liquid has intermediate wettability [Mercer and Cohen, 1990; Wilson, 1992]. Complete drying of the aqueous phase is not considered. Thus, the soil grains remain in continuous contact with the aqueous phase. Correspondingly, solid phase sorption occurs only through the aqueous phase. Adsorption from the vapor phase, which can be significant in dry systems [Pennell et al., 1992], is neglected.

Quantification of the biotransformation processes follows the conceptual approach of Chen et al. [1992]. Biodegradation is assumed to occur only within the aqueous phase by an indigenous, spatially homogeneous, mixed microbial population which is present as attached microcolonies. Monod-type kinetic expressions are used to model biophase utilization of substrates, electron acceptor, and limiting nutrient, as well as growth of the microbial population. Under zero substrate conditions, the microbes are not permitted to die off completely, but are maintained at a minimum concentration representative of the background

Phase Interrelationships

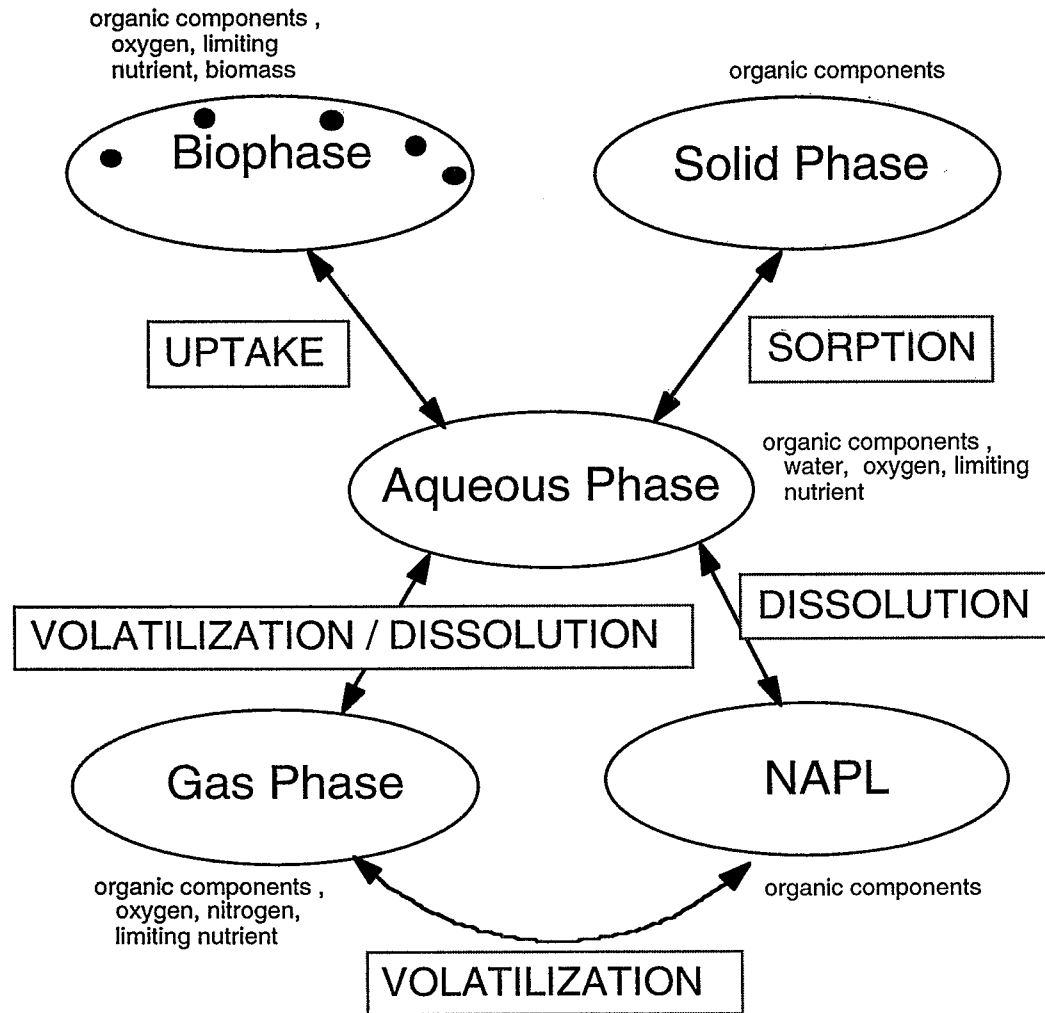


Figure 2.2: Conceptual model of interphase mass transfer pathways.

microorganism populations that are found in nearly all vadose zone environments. It is further assumed that biomass growth does not affect soil permeability and that there is no biomass transport. These two assumptions are consistent with the dominant influence of gas phase mobility during SVE and BV remediation, as well as the generally low mobility of the aqueous phase in the vadose zone. Due to the large number of required microbial transformation parameters, the number of biodegradable components is limited to three. A detailed discussion of many of the concepts mentioned above can be found in *Brock et al.* [1984].

Interphase partitioning and biological degradation processes are strongly temperature dependent. The prediction of transient temperature effects requires the solution of an energy balance equation, which substantially increases model complexity and computational requirements. To limit model complexity and computational requirements, temperature dependence is incorporated in the SVE simulator by the specification of a known (steady state) spatial distribution of temperature. This nonuniform temperature distribution is allowed to be a function of depth only.

2.2 PHASE MASS BALANCE EQUATIONS

Description of multiphase fluid movement is based on the numerical solution of phase mass balance expressions (i.e., flow equations). A general mass balance equation for a fluid phase, α , which is composed of multiple components, c , is expressed as [Abriola, 1989],

$$\frac{\partial}{\partial t} (\phi \rho_{\alpha}^* S_{\alpha}) + \nabla \cdot (\phi \rho_{\alpha}^* S_{\alpha} \mathbf{V}_{\alpha}) = \phi \sum_c \sum_{\beta} E_{\alpha\beta c}^* + \sum_c B_{\alpha c}^* + \rho_{\alpha}^* R_{\alpha} \quad (2.1)$$

where:

- $\alpha = g, a, o, s, b$ denotes the phases comprising the porous medium (g =gas, a =aqueous, o =NAPL; s =solid; b =biophase);
- c denotes components of the phase α ;
- ϕ is the matrix porosity [-];
- ρ_{α}^* is the mass density of phase α [ML^{-3}];
- S_{α} is the saturation of phase α [-];
- \mathbf{V}_{α} is the pore velocity of the phase α [LT^{-1}];
- $E_{\alpha\beta c}^*$ is the rate of interphase mass transfer of component c to the α -phase from the β phase, per unit pore volume; [$ML^{-3}T^{-1}$];
- $B_{\alpha c}^*$ is the net rate of biological transformation of component c in the phase α per unit aquifer volume [$ML^{-3}T^{-1}$]; and
- R_{α} are the internal source/sinks of phase α – volume of α phase produced per unit aquifer volume per unit time [T^{-1}].

The fluid saturations in (2.1) are subject to the constraint,

$$S_a + S_o + S_g = 1 \quad (2.2)$$

Note that S_b is neglected in 2.2. This is a consequence of the assumption that biomass growth does not affect the flow field. For mass balance purposes S_b is included in S_a .

The phase velocity is typically evaluated with an extended form of Darcy's Law which accounts for the simultaneous flow of more than one fluid [Abriola, 1989],

$$\mathbf{q}_{\alpha} = \phi S_{\alpha} \mathbf{V}_{\alpha} = -k \frac{k_{r\alpha}}{\mu_{\alpha}} (\nabla P_{\alpha} - \rho_{\alpha}^* \mathbf{g}) = -\lambda_{\alpha} (\nabla P_{\alpha} - \rho_{\alpha}^* \mathbf{g}) \quad (2.3)$$

where:

- q_α is the specific discharge of phase α [LT^{-1}];
 k is the intrinsic permeability tensor of the medium [L^2];
 $k_{r\alpha}$ is the relative permeability of the α phase [-];
 μ_α is the α phase dynamic viscosity [$ML^{-1}T^{-1}$];
 P_α is the α phase pressure [$ML^{-1}T^{-2}$];
 g is the gravitational acceleration vector [LT^{-2}]; and
 $\lambda_\alpha = k \frac{k_{r\alpha}}{\mu_\alpha}$ is the mobility tensor of phase α [$M^{-1}L^3T$].

Darcy's Law is applicable when average water and gas velocities are within an accepted laminar flow range. For groundwater flow the upper range of validity is generally not violated except for possibly near wells [Bear, 1972]. Similarly, gas fluxes generated in SVE systems are known to diminish rapidly at short distances from the extraction well and are generally within the upper range for laminar flow [Cho and DiGiulio, 1992; Beckett and Huntley, 1994]. Gas phase velocities are therefore also assumed to be within the upper range of validity, or that deviations are localized near the extraction well.

The lower range of validity of Darcy's Law becomes significant for gas flow when the mean free path approaches the diameter of the pores resulting in a gas slippage which increases the effective gas conductivity. Klinkenberg [1941] observed that gas permeability decreases with increasing pressure until it approaches the liquid permeability. Slip flow, which is also known as the Klinkenberg effect in porous media, is most significant at low air pressure in fine textured material [Corey, 1986]. Comprehensive error analyses have been conducted by several researchers [Massmann, 1989; McWhorter, 1990; Baehr and Hult, 1991] and all have found that the Klinkenberg effect is significant only in fine sands and silts at low pressures. However, a Klinkenberg correction factor which accounts for the pressure influence on gas permeability is included in MISER for two reasons: first SVE systems are being increasingly applied in fine textured materials; and secondly implementation of the correction is relatively simple with minimal effect on overall computational requirements. The correction factor has the form,

$$k_g = k_\infty \left(1 + \frac{b}{P_g} \right) \quad (2.4)$$

where:

- k_g is the effective gas phase permeability [L^2];
 k_∞ is the gas phase permeability at a high pressure and is equivalent to the liquid permeability [L^2]; and
 b is a parameter of the porous medium referred to as the Klinkenberg parameter [$ML^{-1}T^{-2}$].

An empirical relation for b was developed by Heid *et al.* [1950],

$$b = (3.98 \times 10^{-5}) k_\infty^{-0.39} \quad (2.5)$$

where b is in atmospheres and k_∞ is in cm^2 . This correlation is based on experimental measurements of air permeability in consolidated soils of 11 synthetic cores and 164 natural cores from various oil fields in the

United States. *Abu-El-Sha'r* [1993] measured air permeability in a variety of unconsolidated soils and compared them with predicted values from (2.5). He found that measured values of b from unconsolidated soils were generally within the envelope containing the measurements of *Heid et al.* [1950], with the exception of an Ottawa sand which was slightly greater than the upper limits. This finding is consistent with an analysis by *Baehr and Hult* [1991]. These researchers conclude that the correlation of *Heid et al.* [1950] may underestimate b for some unconsolidated soils, but in general can be used to approximate the Klinkenberg effect over a range of soil types.

Substituting Darcy's Law (2.3) into (2.1), the aqueous and gas phase mass balance equations are expressed as,

$$\frac{\partial}{\partial t} (\phi \rho_{\alpha}^* S_{\alpha}) - \nabla \cdot [\rho_{\alpha}^* \lambda_{\alpha} (\nabla P_{\alpha} - \rho_{\alpha}^* g)] = \phi E_{\alpha}^* + \rho_{\alpha}^* R_{\alpha} \quad \alpha = a, g \quad (2.6)$$

where $E_{\alpha}^* = \sum_c \sum_{\beta} E_{\alpha\beta_c}^*$. Here it has also been assumed that biodegradation of the organic components has no appreciable effect on the aqueous flow field. These constituents comprise a small fraction of the aqueous phase mass and the rate of their biodegradation is comparatively slow.

Changes in the NAPL saturation result solely from interphase mass transfer processes because the NAPL is assumed to be immobile and there are no internal sources and sinks of organic liquid. The NAPL mass balance equation is thus expressed as,

$$\frac{\partial}{\partial t} (\phi \rho_o^* S_o) = \phi E_o^* \quad (2.7)$$

Changes in the solid phase bulk density due to sorption and desorption are considered negligible and ignored. Also, the biophase volume is considered constant with the same properties as the aqueous phase. Therefore, phase mass balances are not required for the solid and biophases.

2.3 COMPONENT MASS BALANCE EQUATIONS

A general mass balance equation of component c within phase α (i.e. transport equation) is expressed as,

$$\frac{\partial}{\partial t} (\phi S_{\alpha} C_{\alpha_c}) + \nabla \cdot \phi S_{\alpha} (C_{\alpha_c} V_{\alpha} + J_{\alpha_c}^d + J_{\alpha_c}^m) = \phi \sum_{\beta} E_{\alpha\beta_c}^* + B_{\alpha_c}^* \quad (2.8)$$

where:

C_{α_c} is the mass concentration of component c in phase α [ML^{-3}];

$J_{\alpha_c}^d$ is the mass flux of component c in phase α by kinematic dispersion [$ML^{-2}T^{-1}$]; and

$J_{\alpha_c}^m$ is the mass flux of component c in phase α by molecular diffusion [$ML^{-2}T^{-1}$].

The dispersive and diffusive mass fluxes are typically combined and expressed in a Fickian form, e.g.

$$J_{\alpha_c}^d + J_{\alpha_c}^m = -D_{\alpha_c}^h \nabla C_{\alpha_c} \quad (2.9)$$

where $D_{\alpha c}^h$ is the hydrodynamic dispersion tensor of component c in phase α [$L^2 T^{-1}$]. The validity of the Fickian approach for gas transport in the subsurface was rigorously studied by *Fen* [1993] through comparisons with a dusty gas model. The latter model is considered to be more general because it integrates a number of flux mechanisms for multicomponent gas transport which are not captured in the Fickian approach. *Fen* [1993] found the Fickian approach is generally valid when advection is the dominant mass flux mechanism. In systems where diffusive fluxes are significant the Fickian approach coupled with the flow equations was found to inaccurately represent induced pressure gradients due to nonequimolar effects. Errors were found to be reduced when the mass balance equations were expressed in molar form, and when the organic contaminants were of low volatility or their molecular weights were similar to the ambient gases.

Dispersive and diffusive mass fluxes are evaluated in MISER with the Fickian approach expressed by (2.9). This approach is used for several reasons. The Fickian approach is simpler and more expedient to use because the dusty gas model is conceptually complex and numerically difficult to implement. Secondly, advective fluxes of equal magnitude to diffusive fluxes are obtained from very small pressure gradients [*Thorstenson and Pollock*, 1989]. Thus, advection dominance can be reasonably expected throughout a majority of the domain during SVE/BV operations in typical systems. Diffusive fluxes are expected to become significant only at large distances from the extraction/injection wells, in tight formations, or during shut down periods.

To reduce possible errors in the modeling of nonequimolar fluxes in regions of diffusion dominance, the component mass balance equations are converted to molar form [*Fen*, 1993]. Mass concentration may be expanded as,

$$C_{\alpha c} = \rho_{\alpha} M_c x_{\alpha c} \quad (2.10)$$

where ρ_{α} is the phase molar density [$\text{mole } L^{-3}$]. Substituting (2.9) and (2.10) into (2.8), and defining $M_c E_{\alpha c} = \sum_{\beta} E_{\alpha\beta c}^*$ and $M_c B_{\alpha c} = \sum_{\beta} B_{\alpha\beta c}^*$, the general molar based transport equation is expressed as,

$$\frac{\partial}{\partial t} (\phi S_{\alpha} \rho_{\alpha} x_{\alpha c}) + \nabla \cdot \phi S_{\alpha} (\rho_{\alpha} x_{\alpha c} \mathbf{V}_{\alpha} - \rho_{\alpha} D_{\alpha c}^h \nabla x_{\alpha c}) = \phi E_{\alpha c} + B_{\alpha c} \quad (2.11)$$

where:

- $x_{\alpha c}$ is the mole fraction of component c in phase α [-];
- $E_{\alpha c}$ is the net rate of moles of component c transferred to the α phase from all contiguous phases per unit pore volume [$\text{mole } L^{-3} T^{-1}$]; and
- $B_{\alpha c}$ is the net rate of biological transformation of moles of component c in phase α per unit aquifer volume [$\text{mole } L^{-3} T^{-1}$].

The component mole fractions in (2.11) are subject to the constraint,

$$\sum_c x_{\alpha c} = 1 \quad (2.12)$$

Five equations of the form (2.11) are developed for each component – one for each fluid phase, one for the solid phase, and one for the biophase.

Constituents of the gas and aqueous phases are subject to transport by advective and dispersive processes. Constituents of the aqueous phase may also be subject to biotransformation when a separate

biophase is not considered. The constituent mass balance equations for these phases are expressed as,

$$\frac{\partial}{\partial t}(\phi S_g \rho_g x_{gc}) + \nabla \cdot \phi S_g (\rho_g x_{gc} V_g - \rho_g D_{gc}^h \nabla x_{gc}) = \phi E_{gc} \quad c = \gamma, w, O_2, A \quad (2.13a)$$

$$\frac{\partial}{\partial t}(\phi S_a \rho_a x_{ac}) + \nabla \cdot \phi S_a (\rho_a x_{ac} V_a - \rho_a D_{ac}^h \nabla x_{ac}) = \phi E_{ac} + B_{ac} \quad c = \gamma, O_2, A \quad (2.13b)$$

where the denoted components are: γ = organic liquid constituents; w = water or water vapor; O_2 = oxygen; A = nutrient. Note that constituent mass balance equations are not needed for nitrogen (N_2) in the gas phase and water in the aqueous phase due to use of the mole fraction constraint (2.12).

Constituents in the organic phase, solid phase, and biophase are not subject to transport by advection. It is further assumed that diffusion in these phases is negligible. Diffusion in the organic liquid is restricted by the assumed disconnected nature of entrapped residuals. In the solid phase, surface diffusion of sorbed components is considered negligible. The transport equations for the organic and solid phases are then given by:

$$\frac{\partial}{\partial t}(\phi S_o \rho_o x_{oc}) = \phi E_{oc} \quad c = \gamma \quad (2.14)$$

$$\frac{\partial}{\partial t} \left[\frac{\rho_s^* \omega_{sc}}{M_c} \right] = \phi E_{sc} \quad c = \gamma \quad (2.15)$$

where ω_{sc} is the sorbed mass of component c per mass of soil, and ρ_s^* is the bulk solid phase mass density. Diffusion is similarly neglected in the biophase because the constituent distribution is assumed to be dominated by exchange with the aqueous phase and by biotransformation. Due to difficulties in calculating the molar density of subsurface microorganisms, the biophase density is assumed to be equal to the aqueous phase molar density. The transport equation for the biophase is,

$$\frac{\partial}{\partial t}(\phi S_b \rho_a x_{bc}) = \phi E_{bc} + B_{bc} \quad c = \gamma, O_2, A \quad (2.16)$$

where S_b is the biophase saturation. Since aqueous flow is assumed to be unaffected by biomass growth, S_b is considered constant, representing the fraction of pore volume suitable to microbial growth. It is evaluated by,

$$S_b = \frac{X_{\max}}{\phi \rho_a^*} \quad (2.17)$$

where X_{\max} [$M L^{-3}$] is the maximum biomass concentration.

2.4 INTERPHASE MASS TRANSFER

2.4.1 Equilibrium Partitioning

In the absence of phase partitioning rate data the local equilibrium assumption (LEA) has been employed in SVE models to simplify the description of phase partitioning. The LEA enables the use of partition coefficients to relate constituent phase concentrations. These partitioning coefficients are typically developed under the assumption of ideal fluid behavior. Detailed developments of the partition coefficients are found in: *Lyman et al.* [1982]; *Baehr* [1984]; and *Schwarzenbach et al.* [1993].

Equilibrium partitioning between the organic and gas phases is expressed with Raoult's Law, which states,

$$P_{gc} = \gamma_{oc} x_{oc} P_{vc} \quad (2.18)$$

where:

- P_{gc} is the partial pressure of the component c [$ML^{-1}T^{-2}$];
- γ_{oc} is the activity coefficient of component c in the organic phase [-]; and
- P_{vc} is the vapor pressure of component c as a pure substance [$ML^{-1}T^{-2}$].

The activity coefficient is a relative measure of nonideal behavior due to interactions of dissimilar molecules. Ideal behavior of the organic phase (i.e. $\gamma_{oc} = 1$) can be readily assumed for pure NAPLs and many common mixtures of hydrocarbons which are composed of chemically similar components (e.g. gasoline and petroleum hydrocarbons) [Baehr, 1984; Schwarzenbach *et al.*, 1993; Adenekan *et al.*, 1993]. The component vapor pressure in (2.18) also depends on the curvature of the fluid surface, however, the influence of capillarity has been shown to be negligible in ordinary soils of gravel, sand, and silt [Baehr, 1984]. The partial pressure of component c may be expressed with the ideal gas law,

$$P_{gc} = \rho_g x_{gc} RT \quad (2.19)$$

With the foregoing assumptions and employing (2.19), Raoult's Law is expressed as,

$$\rho_g x_{gc} = \frac{x_{oc} P_{vc}}{RT} \quad (2.20)$$

This equation is rearranged to obtain an expression of the organic-gas equilibrium partition coefficient $K_{go_c}^e$,

$$K_{go_c}^e = \frac{x_{gc}}{x_{oc}} = \frac{P_{vc}}{\rho_g RT} \quad (2.21)$$

Dissolution of organic constituents in the aqueous phase is also assumed to occur under ideal fluid behavior. This assumption is valid for many common hydrophobic organic compounds which have small aqueous solubility such that their activity coefficients are constant over the range of possible concentrations (roughly zero to 1000 ppm) [Schwarzenbach *et al.*, 1993; Adenekan *et al.*, 1993]. Equilibrium aqueous-organic partitioning is then expressed by,

$$K_{ao_c}^e = \frac{x_{ac}}{x_{oc}} = x_{ac}^{sol} \quad (2.22)$$

where $K_{ao_c}^e$ is the aqueous-organic equilibrium partition coefficient, and x_{ac}^{sol} is the aqueous phase solubility limit for component c expressed as a mole fraction. Implicit in the assumption that activity coefficients are constant is that co-solvents have negligible influence on the activity coefficients and dissolution capacity. This assumption is well supported for slightly soluble organic compounds [Schwarzenbach *et al.*, 1993].

The aqueous-gas equilibrium partition coefficient is obtained by combining (2.21) and (2.22),

$$K_{ag_c}^e = \frac{x_{ac}}{x_{gc}} = \frac{K_{ao_c}^e}{K_{go_c}^e} \quad (2.23)$$

which can be expanded as,

$$P_{g_c} = \left(\frac{P_{V_c}}{x_{a_c}^{\text{sol}}} \right) x_{a_c} \quad (2.24)$$

Equation (2.24) is a statement of Henry's Law where $K_H = P_{V_c}/x_{a_c}^{\text{sol}}$ is the Henry's Law constant. There is substantial evidence that K_H is unaffected by solute-solute interactions for slightly, or even, moderately soluble organic compounds [Schwarzenbach *et al.*, 1993].

Equilibrium sorption capacity is commonly related to the aqueous concentration by a Freundlich isotherm [Weber *et al.*, 1991],

$$\omega_{s_{a_c}}^e = K_{s_{a_c}}^e (C_{a_c})^n \quad (2.25)$$

where:

$K_{s_{a_c}}^e$ is the aqueous/solid equilibrium partition coefficient [(mass adsorbed / mass soil) / (mass solute / volume solution)]ⁿ [(L³ M⁻¹)ⁿ]; and

n is an empirically derived constant [-].

In the case where $n = 1$, a retardation factor may be defined as,

$$r_c = \left(1 + \frac{\rho_s^* K_{s_{a_c}}^e}{\phi} \right) \quad (2.26)$$

where r_c is then inserted directly into (2.11) for the aqueous phase,

$$\frac{\partial}{\partial t} (\phi r_c S_a \rho_a x_{a_c}) + \nabla \cdot \phi S_a (\rho_a x_{a_c} V_a - \rho_a D_{a_c}^h \nabla x_{a_c}) = \phi E_{a_c} + B_{a_c} \quad (2.27)$$

Sorption is assumed to be both linear and at equilibrium when using retardation for a given component, c . Retardation can be considered for all the components of the aqueous phase.

In the absence of measured data, $K_{s_{a_c}}^e$ can be estimated from commonly used correlations with the soil organic content [Lyman, 1982],

$$K_{s_{a_c}}^e = \frac{\omega_{s_c}}{C_{a_c}} = K_{oc} f_{oc} \quad (2.28)$$

where:

K_{oc} is the organic carbon - normalized partition coefficient (mass sorbed / (mass organic carbon) / (mass) / volume solution); and

f_{oc} is the fraction of organic carbon in the soil [-].

Additionally, $n = 1$ is assumed when this correlation is used.

2.4.2 Rate Limited Interphase Mass Transfer

Nonequilibrium interphase mass transfer processes are often represented by the dual resistance model [Weber and DiGiano, 1996]. This model assumes: (1) mass transfer is controlled by the rate of diffusion on each side of the interface, and (2) no resistance is encountered at the interface. The diffusional resistance on

one side of the interface is frequently considered dominant such that the mass transfer rate can be described with an overall mass transfer coefficient [Weber and DiGianno, 1996]. Generally, use of overall mass transfer coefficients are strictly applicable for the measured system only and should be extended to other systems with caution.

The rate of organic phase volatilization is assumed to be controlled by gas phase resistance and is evaluated with a linear driving force expression [Weber and DiGianno, 1996],

$$E_{go_c} = \rho_g K_{go_c} (x_{go_c}^e - x_{g_c}) \quad (2.29)$$

where:

- K_{go_c} is the lumped gas-organic mass transfer coefficient [T^{-1}]; and
- $x_{go_c}^e$ is the gas phase mole fraction of component c in equilibrium with the organic phase mole fraction of component c [-].

The lumped mass transfer coefficient is a function of the fluid saturation, phase velocity, and properties of the porous media. It can be expressed as,

$$K_{go_c} = k_{g_c} a_{go} \quad (2.30)$$

where:

- k_{g_c} is the overall mass transfer coefficient based on gas phase control (moles transferred / (time)(interfacial contact area)($\Delta\rho_g x$)) [LT^{-1}]; and
- a_{go} is the gas phase specific contact area with the organic phase (gas-organic interfacial contact area / pore volume) [L^{-1}].

The linear driving force model is completed by relating the equilibrium gas phase mole fraction of component c in (2.29) with the corresponding organic phase mole fraction concentration by (2.21),

$$E_{go_c} = \rho_g K_{go_c} (K_{go_c}^e x_{o_c} - x_{g_c}) \quad (2.31)$$

A similar linear driving force model is used for the rate of nonequilibrium organic phase dissolution. In this case, the mass transfer resistance is assumed to be contained within the aqueous phase. Equation (2.22) completes the expression by relating the equilibrium aqueous phase mole fraction of component c to the corresponding organic phase mole fraction,

$$E_{ao_c} = \rho_a K_{ao_c} (K_{ao_c}^e x_{o_c} - x_{a_c}) \quad (2.32)$$

Mass transfer across the aqueous-gas interface is commonly assumed to be controlled by resistance in the aqueous phase. This assumption is generally valid for sparingly soluble organic compounds, however, gas phase resistance becomes more significant for moderately soluble compounds [Munz and Roberts, 1984; Roberts et al., 1985]. The aqueous/gas transfer term is expressed as,

$$E_{ag_c} = \rho_a K_{ag_c} (K_{ag_c}^e x_{g_c} - x_{a_c}) \quad (2.33)$$

Here the equilibrium aqueous phase mole fraction of component c is related to the corresponding gas phase mole fraction with (2.23).

The rate of mass transfer between the aqueous and solid phases is also assumed to be controlled by resistance in the aqueous phase,

$$E_{as_c} = \rho_a K_{as_c} (x_{as_c}^e - x_{a_c}) \quad (2.34)$$

where $x_{as_c}^e$ is the aqueous phase mole fraction in equilibrium with the solid phase loading as computed by (2.25).

Rate limited mass transfer into the biophase is assumed to occur through an immobile liquid film adjacent to the biomass. Diffusional mass transfer resistance into the biophase can be represented with,

$$E_{ab_c} = \rho_a K_{ab_c} (x_{b_c} - x_{a_c}) \quad (2.35)$$

where K_{ab_c} is a lumped biophase/aqueous phase mass exchange coefficient for component c .

2.4.3 Biotransformations

The biodegradation module of MISER can model up to six spatially heterogeneous variables; three biodegradable organic substrates (contaminants), oxygen as the electron acceptor, a limiting nutrient, and the biomass. Monod-type kinetics are used to describe the microbial consumption and growth processes. This approach assumes that the conversion of contaminant into biomass is not instantaneous. Additionally, microbial activity may be limited by the diffusional mass transfer resistance described with (2.35) above. In this approach biotransformation is coupled to the aqueous phase component balance equations (2.13b), by the exchange term E_{ab_c} .

No mass transfer resistance internal to the biophase is considered due to a lack of information and the difficulty in determining the additional parameters needed to represent such processes. The biophase mole fraction profiles of the contaminants, oxygen and the limiting nutrient are therefore assumed to be uniform within the biophase. This assumption is based on two concepts. First, that microbial populations in the subsurface are generally low and hence that microcolonies can be modeled as fully penetrated biofilms. Secondly, that microbial kinetics within the biofilm are fast relative to the time scales of the physical processes in bioventing and thus biophase concentration profiles can be considered to be at "pseudo-steady state" during a time step. By assuming a uniform concentration within the biophase (i.e. a fully penetrated biofilm or a quasi steady state consumption of the substrate l) the mass transfer expression (2.35) can be related to biophase utilization by a Monod-type kinetics expression [Williamson and McCarty, 1976] as,

$$\frac{\partial}{\partial t} (\phi S_b \rho_a x_{b_c}) = \phi E_{b_c} - F_{cl} k_l X \left(\frac{x_{b_l}}{K_{s_l} + x_{b_l}} \right) \left(\frac{x_{b_{O_2}}}{K_{s_{O_2}} + x_{b_{O_2}}} \right) \left(\frac{x_{b_A}}{K_{s_A} + x_{b_A}} \right) I_l I_{O_2} I_A I_S \quad (2.36)$$

where:

- F_{cl} is the use coefficient of component c with substrate l degradation [(mole c)(mass l)⁻¹], $F_{cl} = M_c^{-1}$ when $c = l$. Note that F_{cl} is input in units of gm c /gm l and is converted internally;
- k_l is the maximum specific substrate utilization rate of substrate l [(mass l)(biomass T)⁻¹];
- X is the active biomass concentration [ML^{-3}] expressed on a media volume basis;

- K_{s_l} is the half-saturation coefficient of component l [(mole l)(mole biophase) $^{-1}$]. Note that K_{s_l} is input in units of gm l /liter and is converted internally;
 I_l is a substrate inhibition function described below [-];
 I_{O_2} is an oxygen inhibition function described below [-];
 I_A is a nutrient inhibition function described below [-]; and
 I_S is a saturation inhibition function described below [-].

When $c = O_2$ or A , the Monod expression on the right hand side of (2.36) must be summed over all the substrates, l . Equations of the form (2.36) comprise a system of coupled nonlinear equations.

A second approach used to model biodegradation is to assume that constituents in the aqueous and biophases are in equilibrium. The biotransformation sinks are then directly inserted into the aqueous phase component balance equations (2.13b) using,

$$B_{ac} = -F_{cl}k_l X \left(\frac{x_{a_l}}{K_{s_l} + x_{a_l}} \right) \left(\frac{x_{a_{O_2}}}{K_{s_{O_2}} + x_{a_{O_2}}} \right) \left(\frac{x_{a_A}}{K_{s_A} + x_{a_A}} \right) I_l I_{O_2} I_A I_S \quad (2.37)$$

Because the organic substrates may potentially have inhibitory effects on biodegradation at high concentrations [Speitel and Alley, 1991; Huesemann and Moore, 1994; Mu and Scow, 1994], equations (2.36) and (2.37) are modified to include inhibition kinetics. The following expressions for inhibition are currently used,

$$I_l = \left(1 - \frac{x_l^{min}}{x_l} \right) \left(1 - \frac{x_l}{x_l^{max}} \right) \quad (2.38)$$

where x_l^{min} represents the minimum detectable mole fraction of l (currently 1 ppb on a mass basis) and x_l^{max} is the inhibitory mole fraction of substrate l . In the presence of several substrates, inhibition may be a function of the total substrate mole fraction. In this case, x_l in (2.38) is the sum of all the substrate mole fractions. x_l^{min} remains the same and x_l^{max} can have different values for each substrate. Equation (2.38) is also used to represent nutrient inhibition by replacing l with A and setting $x_A^{min} = 0$.

Microbial activity is assumed to be restricted by a threshold oxygen concentration below which aerobic metabolism ceases. This effect is modeled with an inhibition function of the form,

$$I_{O_2} = \left(1 - \frac{x_{O_2}^{min}}{x_{O_2}} \right) \quad (2.39)$$

where $x_{O_2}^{min}$ is the oxygen mole fraction below which aerobic metabolism ceases.

Microbial activity in unsaturated systems may also depend on moisture content [Fan and Scow, 1993; Holman and Tsang, 1995]. Insufficient data, however, are available to accurately predict such effects. As a preliminary means to investigate the importance of saturation dependency, an untested saturation inhibition function is incorporated in MISER,

$$I_S = \left(0.1 + 0.9 \frac{S_a - S_{ra}}{1 - S_{ra}} \right) \quad (2.40)$$

where S_{ra} is the residual aqueous saturation. Here, metabolic activity is allowed to decrease one order of magnitude as the aqueous phase goes from fully saturated ($S_a = 1$) to residual saturation ($S = S_{ra}$).

Both the equilibrium and nonequilibrium descriptions of biotransformation discussed above require additional expressions describing the growth and decay of the attached biomass. Here, an ordinary differential equation based on Monod kinetics is used,

$$\frac{dX}{dt} = \left[\sum_l Y_l k_l \left(\frac{x_{b_l}}{K_{s_l} + x_{b_l}} \right) \left(\frac{x_{b_{O_2}}}{K_{s_{O_2}} + x_{b_{O_2}}} \right) \left(\frac{x_{b_A}}{K_{s_A} + x_{b_A}} \right) I_l I_{O_2} I_A I_S I_{max} - K_d I_{min} \right] X \quad (2.41)$$

where:

- Y_l is the biomass yield coefficient for the metabolism of substrate l [(biomass)(mole l)⁻¹]. Note that Y_l is input in units of (biomass)(gm l)⁻¹ and is converted internally;
- K_d is the microorganism decay coefficient [T^{-1}];
- I_{max} is a function which prevents the biomass from exceeding a maximum concentration given by,

$$I_{max} = \left(1 - \frac{X}{X_{max}} \right); \quad (2.42)$$

- X_{max} is the maximum allowable biomass concentration
- I_{min} is a function which maintains a minimum concentration reflecting the indigenous population present in uncontaminated subsurface environments,

$$I_{min} = \left(1 - \frac{X_{min}}{X} \right); \text{ and} \quad (2.43)$$

- X_{min} is the background or indigenous concentration of biomass.

The detachment or sloughing of the attached biofilm is not considered in this equation due to a lack of information regarding the necessary parameters to model this process.

2.5 CONSTITUTIVE RELATIONS

2.5.1 Capillary Pressure

Capillary pressure is defined as,

$$P_c = P_{nw} - P_w \quad (2.44)$$

where P_{nw} and P_w are the phase pressure in the non-wetting and wetting fluids, respectively. In soil systems containing only two fluid phases (e.g. gas and aqueous), capillary pressure data is routinely measured and related to fluid saturation. A common fitting function for such two phase gas-aqueous capillary retention data was developed by *van Genuchten* [1980],

$$\bar{S}_a = [1 + (\alpha_{ga} P_c)^n]^{-m} \quad (2.45)$$

where:

$$\bar{S}_a = \frac{S_a - S_{ra}}{1 - S_{ra}} \text{ is the normalized aqueous saturation [-];}$$

$$S_{ra} \text{ is the residual aqueous saturation [-];}$$

$$\alpha_{ga} \text{ fitting variable [}LT^2M^{-1}\text{]; and}$$

$$n, m = 1 - 1/n \text{ are additional fitting variables [-].}$$

Here, the wetting fluid is the aqueous phase and gas phase is the nonwetting fluid.

In contrast to two phase systems, capillary pressure behavior in a porous medium containing three fluid phases (gas, aqueous, and organic) is difficult to measure. Consequently, three phase behavior is typically estimated from two phase capillary pressure data. A parametric model developed by *Parker et al.* [1987] is frequently used to estimate three phase capillary pressure. This model assumes the aqueous phase is the preferential wetting fluid, gas is the nonwetting fluid, and the organic phase retains intermediate wettability and completely separates the gas and aqueous phases. Thus, gas-aqueous interfaces do not occur in the three phase system. Under such conditions the two phase organic-aqueous relation is assumed to control the aqueous phase saturation, and the two phase gas-organic relation is assumed to control the total liquid saturation regardless of the proportion of aqueous and organic liquids. Extensions of the three phase estimation model have been developed for hysteretic behavior and capillary entrapment of organic liquid and gas [*Kaluvarachchi and Parker, 1992*].

The parametric model described above is mathematically expedient, but requires the explicit evaluation of an organic liquid pressure. Tracking the pressure of an organic liquid which is assumed to be at immobile residual saturation is a difficult prospect that is complicated by limitations of the three phase parametric model. The parametric model predicts organic liquid entrapment by the water phase, but not by the gas phase. Thus, evaluation of the residual organic liquid pressure is uncertain in a system containing residual water, and can only be exacerbated by the effects of volatilization from the organic phase. The parametric model also predicts a discontinuity in capillary pressure due to the appearance and disappearance of the organic phase which can potentially lead to numerical difficulties. Lastly, the assumption that the organic phase completely partitions the gas and aqueous phases may be invalid for nonspreading organic liquids [*McBride et al., 1992; Wilson, 1992*], leading to further difficulties in the tracking of organic liquid pressures.

Due to the conceptual difficulty of tracking the pressure of an organic liquid which is assumed to be at immobile residual saturation, MISER assumes that all capillary behavior is independent of the organic liquid pressure. The aqueous phase saturation is related to the two phase gas-aqueous capillary pressure data (2.45), regardless of the proportion of aqueous and organic phase, or the spreading characteristic of the organic phase. Hysteretic behavior in this relation is also neglected. Under this assumption of effects of air entrapment on fluid distribution is neglected. While these effects may be significant under some conditions, neglecting hysteretic behavior due to air entrapment should not pose a significant limitation in modeling field scale SVE/BV systems due the typically large uncertainty in quantifying distributed capillary and relative permeability parameters.

With the assumptions described above, eq. (2.45) is used to determine aqueous saturation from predicted gas and aqueous pressures. Subsequently, gas phase saturation is calculated from (2.2). In the case when the total liquid saturation is greater than one, then S_g is set to zero and $S_a = 1 - S_o$.

2.5.2 Relative Permeability

Relative permeability expressions for the three phase system are obtained from the model of *Parker et al.* [1987]. This model employs correlations developed by *Mualem* [1976] to relate effective permeability with $P_c(S)$ data. The functional forms for the aqueous and gas phases are,

$$k_{ra} = \bar{S}_a^{1/2} [1 - (1 - \bar{S}_a^{1/m})^m]^2 \quad (2.46a)$$

$$k_{rg} = (1 - \bar{S}_t)^{1/2} (1 - \bar{S}_t^{1/m})^{2m} \quad (2.46b)$$

where $S_t = S_a + S_o$. Because the organic phase is assumed immobile throughout SVE operations,

$$k_{ro} = 0 \quad (2.47)$$

As discussed above, hysteretic behavior in the relative permeability relations due to gas entrapment is neglected.

2.5.3 Gas phase density

At common environmental conditions (0-50 °C, ≈ 1 atm.), the gas phase molar density can be accurately estimated with the ideal gas law [*Lyman*, 1982],

$$\rho_g = \frac{P_g}{RT} \quad (2.48)$$

The gas phase mass density is then given by,

$$\rho_g^* = \frac{P_g}{RT} \left(\sum_c x_{gc} M_c \right) \quad (2.49)$$

2.5.4 Liquid Phase Density

Compressibility of the aqueous and organic phases is considered negligible for environmental pressures expected during typical venting operations. Liquid density is computed as a function of composition and temperature.

The molar density of liquid mixtures at constant temperature and pressure is estimated with Amagat's Law [*Reid et al.*, 1977],

$$\rho_a = \frac{1}{\sum_c x_{ac} \left(\frac{M_c}{\rho_c^*} \right)} \quad (2.50)$$

where ρ_c^* is the mass density of pure component c at the mixture temperature [ML^{-3}]. Amagat's Law assumes the volumes of the mixture components are additive. This assumption is valid for mixtures of similar components which are at low to moderate pressures and temperatures not close to the critical point of the mixture. These conditions are readily met for common organic contaminants in typical environmental settings.

2.5.5 Gas phase viscosity

Gas viscosity is independent of pressures at the relatively low pressure encountered in environmental settings (below 10 atmospheres) [Welty *et al.*, 1984]. Under these conditions the gas phase mixture viscosity can be estimated as a function of composition using the semi-empirical formula [Reid *et al.*, 1977],

$$\mu_g = \sum_c \frac{x_{gc} \mu_c}{\sum_j x_{gj} \Phi_{cj}} \quad (2.51a)$$

$$\Phi_{cj} = \frac{[1 + (\mu_c/\mu_j)^{1/2} (M_j/M_c)^{1/4}]^2}{[8(1 + M_j/M_c)]^{1/2}} \quad (2.51b)$$

where μ_c and μ_j are the viscosity of components c and j in the pure state at the system temperature.

2.5.6 Aqueous Phase Viscosity

The aqueous phase viscosity is assumed to be independent of composition and pressure. For steady state temperature conditions the aqueous phase viscosity is constant in time.

2.5.7 Hydrodynamic Dispersion

Dispersive fluxes are assumed to be significant in the mobile aqueous and gas phases only; diffusion within the immobile NAPL and biophase is neglected, as is surface diffusion within the solid phase. A traditional groundwater modeling approach is used in applying Fick's Law to evaluate combined processes of mechanical dispersion and molecular diffusion. For an isotropic medium the dispersion tensor is evaluated by [Bear, 1972],

$$\phi S_\alpha D_{\alpha c LT}^h = \alpha_T |q_\alpha| \delta_{kl} + (\alpha_L - \alpha_T) \frac{q_{\alpha L} q_{\alpha T}}{|q_\alpha|} + \phi S_\alpha \tau_\alpha D_{\alpha c}^m \delta_{LT} \quad (2.52)$$

where:

- α_L and α_T are the coefficients of longitudinal and transverse dispersivity [L];
- $q_{\alpha L}$, $q_{\alpha T}$ are components of the α phase Darcy velocity in the longitudinal and transverse directions [LT^{-1}];
- δ_{LT} is the Kronecker delta;
- $D_{\alpha c}^m$ is the binary molecular diffusion coefficient for component c in phase α [L^2T^{-1}]; and
- τ_α is the porous medium tortuosity factor in phase α [-].

The tortuosity factor is computed as a function of the fluid content using the relationship of Millington and Quirk [1961],

$$\tau_\alpha = \frac{(\phi S_\alpha)^{7/3}}{\phi^2} \quad (2.53)$$

2.5.8 Matrix Compressibility

Changes in the soil porosity are assumed to be negligible under pressure changes induced in typical SVE/BV systems. Thus,

$$\frac{\partial \phi}{\partial P} = 0 \quad (2.54)$$

Section 3

NUMERICAL DEVELOPMENT

The mathematical model of SVE/BV processes developed in the preceding section consists of a number of coupled nonlinear partial differential equations. A numerical solution of these equations is developed using the Galerkin finite element method. This approach is well suited for the simulation of SVE/BV scenarios and has advantages over other methods in the ability to accurately represent boundary conditions and source/sinks, as well as flexible discretization of irregular and heterogeneous domains. The governing equations are solved in two space dimensions; either a cross sectional x - z domain, or an axisymmetrical r - z domain. This section describes details of the numerical algorithm implemented in MISER.

3.1 FINITE ELEMENT APPROACH

The Galerkin finite element method has been widely used to solve groundwater flow and transport equations. Detailed descriptions of this method are found in several reference texts [*Strang and Fix, 1973; Lapidus and Pinder, 1982; Huyakorn and Pinder, 1983; Zienkiewicz and Taylor, 1991*].

The finite element method is based on a weighted residual technique to approximate the solution, u , of the general differential equation represented by,

$$L(u) - f = 0 \quad (3.1)$$

The solution domain, Ω , is first discretized into a network of elements which are connected at discrete nodal points. Within each element the dependent variable u is approximated by a trial function \hat{u} ,

$$u \simeq \hat{u} = \sum_{j=1}^{n^e} N_j(x, z) u_j(t) \quad (3.2)$$

where N_j are the basis functions; u_j are values of the dependent variables at nodes of the element, and n^e is the number of nodes in the element. Substituting \hat{u} into (3.1) results in an error or residual. This residual is weighted and integrated over the domain, Ω , developing the weighted residual equations as,

$$\int_{\Omega} (L(\hat{u}) - f) W_i d\Omega = 0 \quad (3.3)$$

where W_i is an arbitrary weighting function. In the Galerkin finite element method the weighting functions are chosen to be the basis functions. Combining (3.2) and (3.3) leads to a system of n simultaneous equations which are solved for the all the unknowns, u_i , at the n nodes in Ω .

Application of the Galerkin finite element method in MISER employs triangular elements for spatial discretization and linear basis functions. Triangular elements are advantageous for the discretization of irregular and heterogeneous domains. They also simplify the evaluation of integrals in the weighted

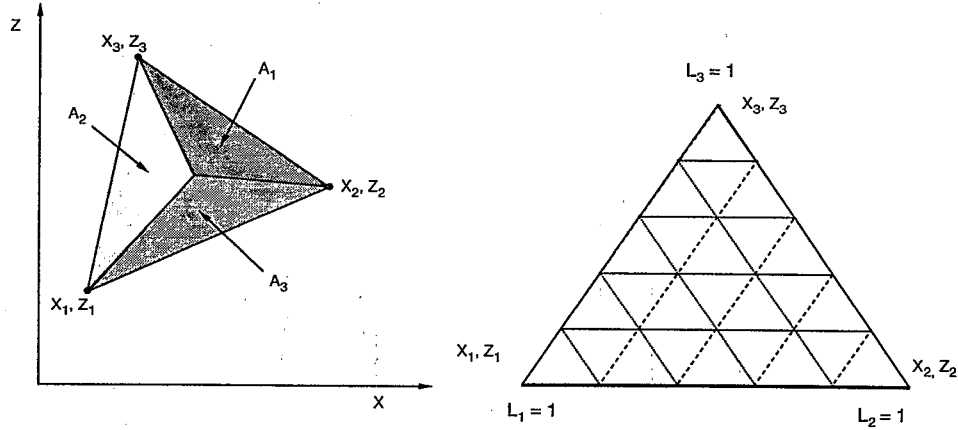


Figure 3.1: Triangular element in global and transformed coordinates (after *Lapidus and Pinder*, 1982).

residual equation by the use of local area coordinates. Figure 3.1 shows an arbitrary triangular element in a two-dimensional Cartesian domain and the transformed element in local area coordinates.

Linear basis functions for triangular elements may be developed in global coordinates as [*Lapidus and Pinder*, 1982; *Huyakorn and Pinder*, 1983],

$$N_j = \frac{1}{2A^e} (\alpha_j + \beta_j x + \gamma_j z) \quad j = 1, 3 \quad (3.4)$$

where,

$$\begin{aligned} \alpha_1 &= x_2 y_3 - x_3 y_2; & \beta_1 &= y_2 - y_3; & \gamma_1 &= x_3 - x_2 \\ \alpha_2 &= x_3 y_1 - x_1 y_3; & \beta_2 &= y_3 - y_1; & \gamma_2 &= x_1 - x_3 \\ \alpha_3 &= x_1 y_2 - x_2 y_1; & \beta_3 &= y_1 - y_2; & \gamma_3 &= x_2 - x_1 \end{aligned} \quad (3.5)$$

and,

$$A^e = \frac{1}{2} \det \begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_2 & z_2 \\ 1 & x_3 & z_3 \end{vmatrix} = \text{area of the triangle} \quad (3.6)$$

Transformation from the global coordinates (x_j, z_j) to the local area coordinates (L_j) is obtained by,

$$L_j = \frac{A_j}{A^e} \quad (3.7)$$

where the subarea A_j is defined by the point of interest within a given element and the element vertices (Figure 3.1). Since $A_1 + A_2 + A_3 = A$ then $L_1 + L_2 + L_3 = 1.0$, and it can be shown that the local coordinate variables L_j are equivalent to the basis function N_j [*Lapidus and Pinder*, 1982; *Huyakorn and Pinder*, 1983]. The advantage of using the local area coordinates is that simple integration formulas have been developed for linear basis functions [*Lapidus and Pinder*, 1982],

$$\int_{\Omega^e} L_1^{m_1} L_2^{m_2} L_3^{m_3} d\Omega^e = \frac{m_1! m_2! m_3!}{(m_1 + m_2 + m_3 + 2)!} 2A \quad (3.8)$$

where Ω^e is the domain occupied by element e . Differentiation of the basis function yields,

$$\frac{\partial N_j}{\partial x} = \frac{\partial L_j}{\partial x} = \frac{\beta_j}{2A^e} \quad (3.9a)$$

$$\frac{\partial N_j}{\partial z} = \frac{\partial L_j}{\partial z} = \frac{\gamma_j}{2A^e} \quad (3.9b)$$

After transformation to local area coordinates the weighted residual equation (3.3) can be expressed as,

$$\sum_{e=1}^{N^e} \int_{\Omega^e} (L(\hat{u}) - f) N_i d\Omega^e = 0 \quad (3.10)$$

where N^e is the number of elements in Ω .

3.2 SOLUTION OF THE PHASE MASS BALANCE EQUATIONS

The aqueous and gas phase flow equations (2.6) are solved by the simultaneous solution (SS) method [Aziz and Settari, 1979]. In this approach fluid pressures are selected as the primary dependent variable, and the flow equations are solved simultaneously for P_a and P_g . Saturations are subsequently updated from capillary pressure-saturation relations (2.45).

3.2.1 Pressure Based Formulation

The simultaneous solution scheme is developed by first recasting the flow equations in terms of the primary pressure variables. The time derivatives in (2.6) are expanded as,

$$\frac{\partial}{\partial t} (\phi \rho_\alpha^* S_\alpha) = \rho_\alpha^* S_\alpha \frac{\partial \phi}{\partial t} + \phi S_\alpha \frac{\partial \rho_\alpha^*}{\partial t} + \phi \rho_\alpha^* \frac{\partial S_\alpha}{\partial t} \quad \alpha = a, g \quad (3.11)$$

The first term on the right hand side (RHS) is the change in mass storage due to matrix compressibility, which is assumed to be negligible.

The middle term represents the temporal change in phase density. Density of the aqueous phase is assumed to depend on composition only. Moreover, the temporal change of this term is assumed to have a minor influence on the aqueous flow field such that it can be lagged by a single time step and moved to the RHS. The density of the gas phase, however, depends on composition and pressure. The temporal change of gas density may be expanded as,

$$\phi S_g \frac{\partial \rho_g^*}{\partial t} = \phi S_g \frac{\partial}{\partial t} \left[\frac{M_g P_g}{RT} \right] = \phi S_g \frac{M_g}{RT} \frac{\partial P_g}{\partial t} + \phi S_g \frac{P_g}{RT} \frac{\partial M_g}{\partial t} \quad (3.12)$$

Similar to the aqueous phase, changes in the gas phase density due to temporal changes in composition (i.e. the last term in (3.12)) are assumed to be small such that they can be lagged by a time step.

The last term on the RHS of (3.11) accounts for change in phase mass storage due to change in fluid saturation. This term is expanded in terms of capillary pressure, obtaining,

$$\phi \rho_\alpha^* \frac{\partial S_\alpha}{\partial t} = \phi \rho_\alpha^* \frac{\partial S_\alpha}{\partial P_c} \frac{\partial P_c}{\partial t} = \phi \rho_\alpha^* C_{P\alpha} \left(\frac{\partial P_g}{\partial t} - \frac{\partial P_a}{\partial t} \right) \quad (3.13)$$

where $C_{p\alpha} = \partial S_\alpha / \partial P_c$ is the capacity coefficient. Note $C_p \equiv C_{p_a} = \partial S_a / \partial P_c = -\partial S_g / \partial P_c$.

The pressure based aqueous and gas flow equations are developed in two dimensional coordinates by substituting (3.12) and (3.13) into (2.6), dividing through by ρ_a^* , and expanding the spatial derivatives,

$$\begin{aligned} \phi C_p \left(\frac{\partial P_g}{\partial t} - \frac{\partial P_a}{\partial t} \right) - \frac{\partial}{\partial x} \left[\lambda_{a_{xx}} \left(\frac{\partial P_a}{\partial x} - \rho_a^* g_x \right) \right] - \frac{\partial}{\partial z} \left[\lambda_{a_{zz}} \left(\frac{\partial P_a}{\partial z} - \rho_a^* g_z \right) \right] \\ - \lambda_{a_{xx}} \left(\frac{1}{\rho_a^*} \frac{\partial P_a}{\partial x} - g_x \right) \frac{\partial \rho_a^*}{\partial x} - \lambda_{a_{zz}} \left(\frac{1}{\rho_a^*} \frac{\partial P_a}{\partial z} - g_z \right) \frac{\partial \rho_a^*}{\partial z} \\ = \frac{\phi}{\rho_a^*} \left(E_a^* - S_a \frac{\partial \rho_a^*}{\partial t} \right) + R_a \end{aligned} \quad (3.14a)$$

$$\begin{aligned} \frac{\phi S_g}{RT \rho_g} \frac{\partial P_g}{\partial t} - \phi C_p \left(\frac{\partial P_g}{\partial t} - \frac{\partial P_a}{\partial t} \right) \\ - \frac{\partial}{\partial x} \left[\lambda_{g_{xx}} \left(\frac{\partial P_g}{\partial x} - \rho_g^* g_x \right) \right] - \frac{\partial}{\partial z} \left[\lambda_{g_{zz}} \left(\frac{\partial P_g}{\partial z} - \rho_g^* g_z \right) \right] \\ - \lambda_{g_{xx}} \left(\frac{1}{\rho_g^*} \frac{\partial P_g}{\partial x} - g_x \right) \frac{\partial \rho_g^*}{\partial x} - \lambda_{g_{zz}} \left(\frac{1}{\rho_g^*} \frac{\partial P_g}{\partial z} - g_z \right) \frac{\partial \rho_g^*}{\partial z} \\ = \frac{\phi}{\rho_g^*} \left(E_g^* - \frac{S_g P_g}{RT} \frac{\partial M_g}{\partial t} \right) + R_g \end{aligned} \quad (3.14b)$$

where g_x and g_z are components of the gravitational acceleration vector. Division by the phase density is performed to simplify the specification of boundary conditions.

3.2.2 Trial Functions and Weighted Residual Equations

Standard trial functions are employed for the primary variable, P_α , and for secondary variables ρ_α^* , E_α , and λ_α ,

$$P_\alpha(x, y, t) \simeq \sum_{j=1}^3 P_{\alpha_j}(t) N_j(x, y) \quad (3.15a)$$

$$\rho_\alpha^*(x, y, t) \simeq \sum_{j=1}^3 \rho_{\alpha_j}^*(t) N_j(x, y) \quad (3.15b)$$

$$E_\alpha^*(x, y, t) \simeq \sum_{j=1}^3 E_{\alpha_j}^*(t) N_j(x, y) \quad (3.15c)$$

$$\lambda_\alpha(x, y, t) \simeq \sum_{j=1}^3 \lambda_{\alpha_j}(t) N_j(x, y) \quad (3.15d)$$

Based on the work of *Abriola and Rathfelder* [1993] and *Rathfelder and Abriola* [1994], nontraditional finite element approaches are used in defining trial functions involving products of dependent variables and coefficients. In this work it was found that computational efficiency and material balance properties are

enhanced when products are expanded collectively with a single basis function. The following trial functions for the terms in (3.14) are defined,

$$\frac{\phi S_g}{RT \rho_g} \frac{\partial P_g}{\partial t} \simeq \frac{\phi^e}{R} \sum_{j=1}^3 \frac{S_{g_j}}{\rho_{g_j} T_j} \frac{\partial P_{g_j}}{\partial t} N_j \quad (3.16a)$$

$$\phi C_p \left(\frac{\partial P_g}{\partial t} - \frac{\partial P_a}{\partial t} \right) \simeq \phi^e \sum_{j=1}^3 C_{p_j} \left(\frac{\partial P_{g_j}}{\partial t} - \frac{\partial P_{a_j}}{\partial t} \right) N_j \quad (3.16b)$$

$$\lambda_{\alpha_{xx}} \left(\frac{1}{\rho_{\alpha}^*} \frac{\partial P_{\alpha}}{\partial x} - g_x \right) \simeq \left(\sum_{j=1}^3 \frac{\lambda_{\alpha_{xxj}}}{\rho_{\alpha_j}^*} N_j \right) \left(\sum_{j=1}^3 P_{\alpha_j} \frac{\partial N_j}{\partial x} \right) - g_x \left(\sum_{j=1}^3 \lambda_{\alpha_{xxj}} N_j \right) \quad (3.16c)$$

$$\frac{\phi}{\rho_{\alpha}^*} E_{\alpha}^* \simeq \phi^e \sum_{j=1}^3 \left(\frac{1}{\rho_{\alpha_j}^*} E_{\alpha_j}^* \right) N_j \quad (3.16d)$$

$$\phi \frac{S_a}{\rho_a^*} \frac{\partial \rho_a^*}{\partial t} \simeq \phi^e \sum_{j=1}^3 \left(\frac{S_{a_j}}{\rho_{a_j}^*} \frac{\partial \rho_{a_j}^*}{\partial t} \right) N_j \quad (3.16e)$$

$$\frac{\phi S_g P_g}{\rho_g^* RT} \frac{\partial M_g}{\partial t} \simeq \frac{\phi^e}{R} \sum_{j=1}^3 \left(\frac{S_{g_j} P_{g_j}}{\rho_{g_j}^* T_j} \frac{\partial M_{g_j}}{\partial t} \right) N_j \quad (3.16f)$$

where ϕ^e is the porosity in element e .

Substituting the trial functions (3.15) and (3.16) into (3.14) and applying Galerkin's method and Green's theorem leads to the weak form of the weighted residual equations. For the aqueous and gas phases, respectively, the weak form is expressed as,

$$\begin{aligned} & \sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \phi^e \left[C_{p_j} \left(\frac{\partial P_{g_j}}{\partial t} - \frac{\partial P_{a_j}}{\partial t} \right) N_j \right] N_i \right. \\ & \quad + \lambda_{a_{xxj}} N_j \left(P_{a_j} \frac{\partial N_j}{\partial x} - \rho_{a_j}^* g_x N_j \right) \frac{\partial N_i}{\partial x} + \lambda_{a_{zzj}} N_j \left(P_{a_j} \frac{\partial N_j}{\partial z} - \rho_{a_j}^* g_z N_j \right) \frac{\partial N_i}{\partial z} \\ & \quad - \left[\frac{\lambda_{a_{xxj}}}{\rho_{a_j}^*} N_j \left(P_{a_j} \frac{\partial N_j}{\partial x} \right) - g_x \lambda_{a_{xxj}} N_j \right] \rho_{a_j}^* \frac{\partial N_j}{\partial x} N_i \\ & \quad \left. - \left[\frac{\lambda_{a_{zzj}}}{\rho_{a_j}^*} N_j \left(P_{a_j} \frac{\partial N_j}{\partial z} \right) - g_z \lambda_{a_{zzj}} N_j \right] \rho_{a_j}^* \frac{\partial N_j}{\partial z} N_i \right\} d\Omega^e \quad (3.17a) \end{aligned}$$

$$\begin{aligned} & = \sum_{e=1}^{N^e} \int_{\Gamma^e} \left\{ \lambda_{a_{xxj}} N_j \left(P_{a_j} \frac{\partial N_j}{\partial x} - \rho_{a_j}^* g_x N_j \right) + \lambda_{a_{zzj}} N_j \left(P_{a_j} \frac{\partial N_j}{\partial z} - \rho_{a_j}^* g_z N_j \right) \right\} n N_i d\Gamma^e \\ & \quad + \sum_{e=1}^{N^e} \phi_e \int_{\Omega^e} \left\{ \frac{1}{\rho_{a_j}^*} E_{a_j}^* N_j - \frac{S_{a_j}}{\rho_{a_j}^*} \frac{\partial \rho_{a_j}^*}{\partial t} N_j + R_{a_j} N_j \right\} N_i d\Omega^e \end{aligned}$$

$$\begin{aligned} & \sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \phi^e \left[\frac{S_{g_j}}{RT_j \rho_{g_j}} \frac{\partial P_{g_j}}{\partial t} N_j - C_{p_j} \left(\frac{\partial P_{g_j}}{\partial t} - \frac{\partial P_{a_j}}{\partial t} \right) N_j \right] N_i \right. \\ & \quad \left. + \lambda_{g_{xxj}} N_j \left(P_{g_j} \frac{\partial N_j}{\partial x} - \rho_{g_j}^* g_x N_j \right) \frac{\partial N_i}{\partial x} + \lambda_{g_{zzj}} N_j \left(P_{g_j} \frac{\partial N_j}{\partial z} - \rho_{g_j}^* g_z N_j \right) \frac{\partial N_i}{\partial z} \right\} \end{aligned}$$

$$\begin{aligned}
& - \left[\frac{\lambda_{gxxj}}{\rho_{gj}^*} N_j \left(P_{gj} \frac{\partial N_j}{\partial x} \right) - g_x \lambda_{gxxj} N_j \right] \rho_{gj}^* \frac{\partial N_j}{\partial x} N_i \\
& - \left[\frac{\lambda_{gzzj}}{\rho_{gj}^*} N_j \left(P_{gj} \frac{\partial N_j}{\partial z} \right) - g_z \lambda_{gzzj} N_j \right] \rho_{gj}^* \frac{\partial N_j}{\partial z} N_i \Big\} d\Omega^e \quad (3.17b) \\
& = \sum_{e=1}^{N^e} \int_{\Gamma^e} \left\{ \lambda_{gxxj} N_j \left(P_{gj} \frac{\partial N_j}{\partial x} - \rho_{gj}^* g_x N_j \right) + \lambda_{gzzj} N_j \left(P_{gj} \frac{\partial N_j}{\partial z} - \rho_{gj}^* g_z N_j \right) \right\} n_i N_i d\Gamma^e \\
& + \sum_{e=1}^{N^e} \phi^e \int_{\Omega^e} \left\{ \frac{1}{\rho_{gj}^*} E_{gj}^* N_j - \frac{\phi^e S_{gj} P_{gj}}{\rho_{gj}^* RT_j} \frac{\partial M_{gj}}{\partial t} N_j + R_{gj} N_j \right\} N_i d\Omega^e
\end{aligned}$$

where the summation convention is used for the repeated subscript j , Γ^e is the domain boundary of domain associated with element e , and \mathbf{n} is an outward unit normal vector.

Performing the integration over each element and assembling the resulting equations yields a global system of time dependent ordinary differential equations with nonlinear coefficients,

$$[A] \frac{\partial \{P\}}{\partial t} + [B] \{P\} = \{F\} + \{E\} + \{Q\} \quad (3.18)$$

where $\{P\}$ is the vector of pressures at all nodes and is ordered with alternating aqueous and gas pressures, $[A]$ is the mass matrix, $[B]$ is the stiffness matrix, and $\{F\}$, $\{E\}$, and $\{Q\}$ are the RHS matrices. MISER also includes an option to lump the mass matrix. Detailed development of the element matrices is given in Appendix A.

3.2.3 Capacity Coefficients

Treatment of the capacity coefficient is based on the work of *Abriola and Rathfelder* [1993] who investigated mass balance accuracy in two-phase flow problems. They showed that finite element solution with the SS scheme is mass conservative when the accumulation term is approximated with the nontraditional trial function in (3.16b) and the capacity coefficient is evaluated with the standard chord slope approximation,

$$C_{pi} = \frac{S_{a_i}^{k,t+1} + S_{a_i}^t}{P_{c_i}^{k,t+1} + P_{c_i}^t} \quad (3.19)$$

where k is an iteration counter. C_{pi} is set to a dummy minimum value (1×10^{-7}) at the first iteration ($k = 1$), and additionally if the calculated value of C_{pi} is less than the minimum value.

3.3 MATERIAL PROPERTIES

The representation of heterogeneous soil properties is essential for accurate representation of field processes. Discontinuities in soil properties produce discontinuities in dependent variables such as saturation, mobility, and mass exchange. Since these parameters influence transport and degradation processes, the method used to represent discontinuous material properties can influence simulation results and interpretation. Discontinuities are typically handled by averaging adjacent, but different material properties [Voss, 1984; Simunek *et al.*, 1994]. This can introduce a smearing or dispersion effect that

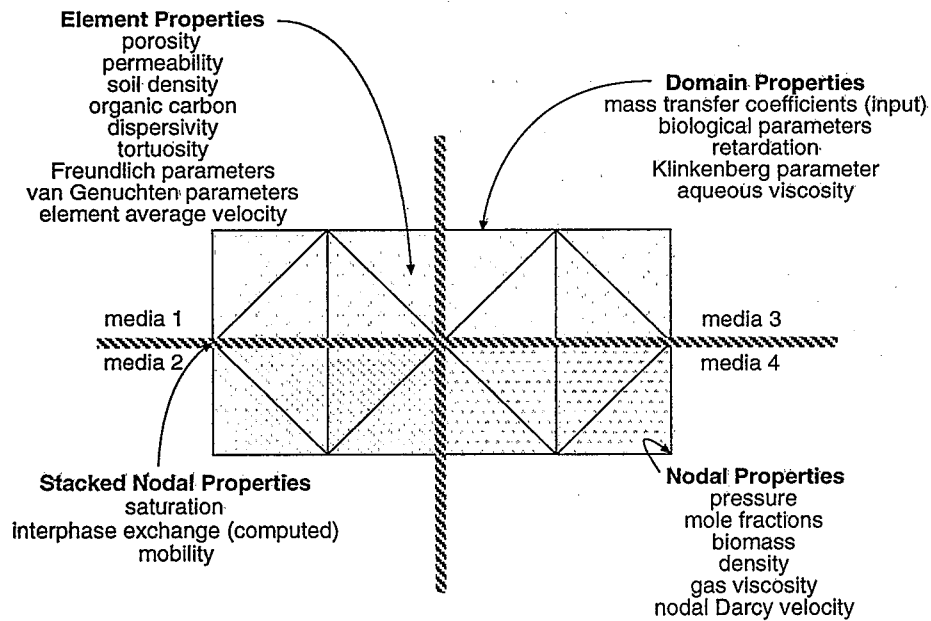


Figure 3.2: Variable representation in MISER.

necessitates grid refinement for the accurate representation of interfaces. Such refinement, however, may not always be possible for the simulation of large and/or strongly heterogeneous domains. MISER incorporates a unique data structure for the representation of discontinuous properties at soil interfaces. MISER maintains and numerically tracks discontinuous variables at material property interfaces, including: saturation, mobility, and interphase exchange. These variables are termed 'stacked nodal variables' (Figure 3.2) because multiple values can exist at a single node. A maximum of four material property blocks may be contiguous at a given node.

Other variables are treated as either continuous nodal properties (material independent), element constant variables (material dependent), or domain properties (spatially invariant). Figure 3.2 identifies variables in each group.

3.4 VELOCITY EQUATIONS

MISER includes two optional approaches for evaluating aqueous and gas phase specific discharges from Darcy's Law (2.3). The first approach generates element wise constant fluxes by solving (2.3) directly,

$$q_{\alpha_x}^e = -\lambda_{\alpha_{xx}}^e \left(\sum_{j=1}^3 P_{\alpha_j} \frac{\partial N_j}{\partial x} - \rho_{\alpha}^{*e} g_x \right) \quad (3.20a)$$

$$q_{\alpha_z}^e = -\lambda_{\alpha_{zz}}^e \left(\sum_{j=1}^3 P_{\alpha_j} \frac{\partial N_j}{\partial z} - \rho_{\alpha}^{*e} g_z \right) \quad (3.20b)$$

where:

- $q_{\alpha_x}^e$ is the element constant specific discharge of phase α in the x direction [LT^{-1}];
 $q_{\alpha_z}^e$ is the element constant specific discharge of phase α in the z direction [LT^{-1}];
 $\lambda_{\alpha_{xx}}^e$ is the element averaged mobility of phase α in the x direction [$M^{-1}L^3T$];
 $\lambda_{\alpha_{zz}}^e$ is the element averaged mobility of phase α in the z direction [$M^{-1}L^3T$]; and
 ρ_{α}^e is the element averaged phase mass density [ML^{-3}].

The element averaged mobilities are computed from,

$$\lambda_{\alpha_{ll}}^e = \sum_{j=1}^3 \frac{\lambda_{\alpha_{llj}}}{3} \quad (3.21)$$

Averaging of the phase density is done in a manner that is consistent with the spatial variability of the pressure term [Voss, 1984],

$$\rho_{\alpha}^{*e} = \frac{\sum_{j=1}^3 \rho_{\alpha_j}^* \left| \frac{N_j}{\partial z} \right|}{\sum_{j=1}^3 \left| \frac{N_j}{\partial z} \right|} \quad (3.22)$$

The second option is the calculation of nodal specific fluxes by solving Darcy's Law (2.3) with the finite element method. This approach eliminates discontinuities in the velocity field that are present when using element average velocities. The use of a continuous nodal velocity field has been shown to yield significant improvements in mass balance for the transport equation [Yeh, 1981]. After substituting the trial functions and applying Galerkin's method to (2.3), the weighted residual equations for the aqueous phase are,

$$\sum_{e=1}^{N^e} \int_{\Omega^e} q_{a_x} N_j N_i d\Omega^e = - \sum_{e=1}^{N^e} \int_{\Omega^e} \lambda_{a_{xx}} N_j \left(P_{a_j} \frac{\partial N_j}{\partial x} - \rho_{a_j}^* g_x N_j \right) N_i d\Omega^e \quad (3.23a)$$

$$\sum_{e=1}^{N^e} \int_{\Omega^e} q_{a_z} N_j N_i d\Omega^e = - \sum_{e=1}^{N^e} \int_{\Omega^e} \lambda_{a_{zz}} N_j \left(P_{a_j} \frac{\partial N_j}{\partial z} - \rho_{a_j}^* g_z N_j \right) N_i d\Omega^e \quad (3.23b)$$

where the summation convention is used for the repeated subscript j .

For the gas phase, a modified version of (2.3) is used which is expressed in terms of the equivalent head [Mendoza and Frind, 1990]. The weighted residual equations for the gas phase are,

$$\sum_{e=1}^{N^e} \int_{\Omega^e} q_{g_x} N_j N_i d\Omega^e = - \sum_{e=1}^{N^e} \int_{\Omega^e} \lambda_{g_{xx}} \rho_{g^o}^* g N_j \left[h_{g_j} \frac{\partial N_j}{\partial x} - \left(\frac{\rho_{g_j}^*}{\rho_{g^o}^*} - 1 \right) N_j \right] N_i d\Omega^e \quad (3.24a)$$

$$\sum_{e=1}^{N^e} \int_{\Omega^e} q_{g_z} N_j N_i d\Omega^e = - \sum_{e=1}^{N^e} \int_{\Omega^e} \lambda_{g_{zz}} \rho_{g^o}^* g N_j \left[h_{g_j} \frac{\partial N_j}{\partial z} - \left(\frac{\rho_{g_j}^*}{\rho_{g^o}^*} - 1 \right) N_j \right] N_i d\Omega^e \quad (3.24b)$$

where the summation convention is used for the repeated subscript j , $\rho_{g^o}^*$ is the mass density of the uncontaminated gas phase, and the equivalent head h_{g_j} is defined by,

$$h_{g_j} = \frac{P_{g_j}}{\rho_{g^o}^* g} + z_j \quad (3.25)$$

Additionally, nodal gas phase velocity is set to zero when the gas phase saturation is less than a user specified value, currently set at 5%.

3.5 COMPONENT MASS BALANCE EQUATIONS

The transport equations are solved in sequence [Reeves, 1993; Reeves and Abriola, 1994]. They are first modified to eliminate the divergence of velocity which is not defined when using a linear interpolation space to approximate the pressure field.

The general form of the component balance equation from Section 2, eq. (2.11) in molar form is,

$$\frac{\partial}{\partial t}(\phi S_\alpha \rho_\alpha x_{\alpha c}) + \nabla \cdot \phi S_\alpha (\rho_\alpha x_{\alpha c} V_\alpha - \rho_\alpha D_{\alpha c}^h \nabla x_{\alpha c}) = \phi E_{\alpha c} + B_{\alpha c} \quad (3.26)$$

The first two terms in (3.26) are expanded using the chain rule,

$$\begin{aligned} x_{\alpha c} \frac{\partial}{\partial t}(\phi S_\alpha \rho_\alpha) + \phi S_\alpha \rho_\alpha \frac{\partial}{\partial t}(x_{\alpha c}) + x_{\alpha c} \nabla \cdot (\phi S_\alpha \rho_\alpha V_\alpha) + \phi S_\alpha \rho_\alpha V_\alpha \cdot \nabla(x_{\alpha c}) - \\ \nabla \cdot (\phi S_\alpha \rho_\alpha D_{\alpha c}^h \nabla x_{\alpha c}) = \phi E_{\alpha c} + B_{\alpha c} \end{aligned} \quad (3.27)$$

Equation (3.26) is next summed over all the components c in phase α and multiplied by $x_{\alpha c}$,

$$x_{\alpha c} \frac{\partial}{\partial t}(\phi \rho_\alpha S_\alpha) + x_{\alpha c} \nabla \cdot (\phi \rho_\alpha S_\alpha V_\alpha) = x_{\alpha c} (\phi E_\alpha + B_\alpha) \quad (3.28)$$

Substituting (3.28) into (3.27), the divergence in velocity term is eliminated, obtaining,

$$\begin{aligned} \phi S_\alpha \rho_\alpha \frac{\partial}{\partial t}(x_{\alpha c}) + \phi S_\alpha \rho_\alpha V_\alpha \cdot \nabla(x_{\alpha c}) - \nabla \cdot (\phi S_\alpha \rho_\alpha D_{\alpha c}^h \nabla x_{\alpha c}) \\ = \phi E_{\alpha c} + B_{\alpha c} - x_{\alpha c} (\phi E_\alpha + B_\alpha) \end{aligned} \quad (3.29)$$

The nonequilibrium mass exchange terms are evaluated with the linear driving force expressions described in the preceding chapter. These expressions have the general form,

$$E_{\alpha\beta c} = \rho_\alpha K_{\alpha\beta c} (K_{\alpha\beta c}^e x_{\beta c} - x_{\alpha c}) \quad (3.30)$$

where $E_{\alpha\beta c}$ is the rate of moles of component c transferred to the α -phase from the β phase per pore volume, and $K_{\alpha\beta c}^e$ is the equilibrium phase partitioning coefficient for the α phase in contact with the β phase.

The nonlinear biotransformation reaction terms, $B_{\alpha c}$, are developed in detail in Section 3.6. Presently these terms are represented by,

$$B_{\alpha c} = \mu_{\alpha c} x_{\alpha c} \quad (3.31)$$

where $\mu_{\alpha c}$ represents a Monod-type nonlinear reaction coefficient [mole $L^{-3}T^{-1}$].

The foregoing expressions are used to develop a general form of the transport equation. Substituting (3.30) and (3.31) into (3.29) and rearranging, the general form of the transport equation is,

$$\begin{aligned} \phi S_\alpha \rho_\alpha \frac{\partial}{\partial t}(x_{\alpha c}) + \phi S_\alpha \rho_\alpha V_\alpha \cdot \nabla(x_{\alpha c}) - \nabla \cdot (\phi S_\alpha \rho_\alpha D_{\alpha c}^h \nabla x_{\alpha c}) \\ + x_{\alpha c} \left(\phi \rho_\alpha \sum_{\beta} K_{\alpha\beta c} + \phi E_\alpha + B_\alpha - \mu_{\alpha c} \right) \\ = + \phi \rho_\alpha \sum_{\beta} K_{\alpha\beta c} K_{\alpha\beta c}^e x_{\beta c} - \phi \sum_{\beta} \rho_\beta K_{\beta\alpha c} (K_{\beta\alpha c}^e x_{\alpha c} - x_{\beta c}) \end{aligned} \quad (3.32)$$

Note that the last term on the right hand side of (3.32) reflects those mass transfer terms for which the mass transfer resistance is assumed to be present in the β phase. In (3.32) the coefficients included in the mass exchange terms reflect which phase controls the equilibrium partitioning and nonequilibrium mass exchange. To simplify notation and to facilitate the development of general finite element equations, the following lumped coefficients are defined,

$$\bar{K}_{\alpha_c} = \phi \rho_\alpha \sum_{\beta} K_{\alpha\beta_c} + \phi E_\alpha + B_\alpha - \mu_{\alpha_c} \quad (3.33a)$$

$$\bar{F}_{\alpha_c} = \phi \rho_\alpha \sum_{\beta} K_{\alpha\beta_c} K_{\alpha\beta_c}^e x_{\beta_c} - \phi \sum_{\beta} \rho_\beta K_{\beta\alpha_c} \left(K_{\beta\alpha_c}^e x_{\alpha_c} - x_{\beta_c} \right) \quad (3.33b)$$

These terms contain the mass exchange and bioreaction information and may be different for each phase and component. Tables 3.1 and 3.2 summarize the phase and component dependencies of these lumped coefficients. Substituting (3.33) into (3.32), the final form of the general transport equation in molar form is,

$$\phi S_\alpha \rho_\alpha \frac{\partial}{\partial t} (x_{\alpha_c}) + \phi S_\alpha \rho_\alpha V_\alpha \cdot \nabla (x_{\alpha_c}) + \nabla \cdot \left(\phi S_\alpha \rho_\alpha D_{\alpha_c}^h \nabla x_{\alpha_c} \right) + x_{\alpha_c} \bar{K}_{\alpha_c} = \bar{F}_{\alpha_c} \quad \alpha = g, a \quad (3.34)$$

Equation (3.34) is used to model transport of component c in the two mobile phases. The transport equation for the immobile phases (organic, solid, biophase) is derived directly from (3.34) by neglecting the advective and dispersive terms, obtaining,

$$\phi S_\alpha \rho_\alpha \frac{\partial}{\partial t} (x_{\alpha_c}) + x_{\alpha_c} \bar{K}_{\alpha_c} = \bar{F}_{\alpha_c} \quad \alpha = o, s, b \quad (3.35)$$

Note that for the solid phase, the principal variable is the mass loading, ω_{s_c} , the bulk solid phase mass density, ρ_s^* , is used on the LHS in place of $\phi S_\alpha \rho_\alpha$, and the RHS lumped coefficient is also expressed on a mass basis.

3.5.1 Weighted Residual Equations in Cartesian Coordinates

The primary dependent variables in (3.34) are the component mole fractions. These variables are expanded with standard weighting functions developed in Section 3.1. The nontraditional approaches described in Section 3.2.2 are used in defining trial functions involving products of dependent variables and coefficients such as $S_\alpha \rho_\alpha$, \bar{K}_{α_c} , and \bar{F}_{α_c} . Expressing (3.34) in a two dimensional Cartesian coordinates and substituting the weighting functions, the weighted residual equation for the (3.34) using element average velocities is,

$$\begin{aligned} \sum_{e=1}^{N_e} \int_{\Omega^e} \left\{ \phi^e S_{\alpha_j} \rho_{\alpha_j} N_j \left[\frac{\partial x_{\alpha_c j}}{\partial t} N_j + V_{\alpha_x}^e x_{\alpha_c j} \frac{\partial N_j}{\partial x} + V_{\alpha_z}^e x_{\alpha_c j} \frac{\partial N_j}{\partial z} \right] \right. \\ \left. + \frac{\partial}{\partial x} \left[\phi^e S_{\alpha_j} \rho_{\alpha_j} N_j \left(D_{\alpha_{c x x}}^{h^e} x_{\alpha_c j} \frac{\partial N_j}{\partial x} + D_{\alpha_{c x z}}^{h^e} x_{\alpha_c j} \frac{\partial N_j}{\partial z} \right) \right] \right. \\ \left. + \frac{\partial}{\partial z} \left[\phi^e S_{\alpha_j} \rho_{\alpha_j} N_j \left(D_{\alpha_{c z x}}^{h^e} x_{\alpha_c j} \frac{\partial N_j}{\partial x} + D_{\alpha_{c z z}}^{h^e} x_{\alpha_c j} \frac{\partial N_j}{\partial z} \right) \right] \right. \\ \left. + x_{\alpha_c j} \bar{K}_{\alpha_c j} N_j N_j \right\} N_i d\Omega^e = \sum_{e=1}^{N_e} \int_{\Omega^e} \bar{F}_{\alpha_c j} N_j N_i d\Omega^e \end{aligned} \quad (3.36)$$

Component	Phase pair	Mass exchange expression
organic	gas-organic	$E_{go_\gamma} = \rho_g K_{go_\gamma} (K_{go_\gamma}^e x_{o_\gamma} - x_{g_\gamma})$
	aqueous-gas	$E_{ag_\gamma} = \rho_a K_{ag_\gamma} (K_{ag_\gamma}^e x_{g_\gamma} - x_{a_\gamma})$
	aqueous-organic	$E_{ao_\gamma} = \rho_a K_{ao_\gamma} (K_{ao_\gamma}^e x_{o_\gamma} - x_{a_\gamma})$
	aqueous-solid	$E_{as_\gamma} = \rho_a K_{as_\gamma} (K_{as_\gamma}^e x_{s_\gamma} - x_{a_\gamma})$
	aqueous-biophase	$E_{ab_\gamma} = \rho_a K_{ab_\gamma} (x_{b_\gamma} - x_{a_\gamma})$
	all others	$E_{\alpha\beta_\gamma} = 0$
water	aqueous-gas	$E_{ga_w} = \rho_g K_{ga_w} (K_{ga_w}^e x_{a_w} - x_{g_w})$
	all others	$E_{\alpha\beta_w} = 0$
oxygen	aqueous-gas	$E_{ag_{O_2}} = \rho_a K_{ag_{O_2}} (K_{ag_{O_2}}^e x_{g_{O_2}} - x_{a_{O_2}})$
	aqueous-biophase	$E_{ab_{O_2}} = \rho_a K_{ab_{O_2}} (x_{b_{O_2}} - x_{a_{O_2}})$
	all others	$E_{\alpha\beta_{O_2}} = 0$
nutrient	aqueous-gas	$E_{ag_A} = \rho_a K_{ag_A} (K_{ag_A}^e x_{g_A} - x_{a_A})$
	aqueous-biophase	$E_{ab_A} = \rho_a K_{ab_A} (x_{b_A} - x_{a_A})$
	all others	$E_{\alpha\beta_A} = 0$

Table 3.1: Summary of mass transfer expressions.

After applying Green's Theorem the weak form of the weighted residual equation (3.36) is,

$$\begin{aligned}
& \sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \phi^e S_{\alpha_j} \rho_{\alpha_j} N_j \left[\frac{\partial x_{\alpha_{c_j}}}{\partial t} N_j + V_{\alpha_x}^e x_{\alpha_{c_j}} \frac{\partial N_j}{\partial x} + V_{\alpha_z}^e x_{\alpha_{c_j}} \frac{\partial N_j}{\partial z} \right] + x_{\alpha_{c_j}} \bar{K}_{\alpha_{c_j}} N_j N_j \right\} N_i d\Omega^e \\
& + \sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \phi^e S_{\alpha_j} \rho_{\alpha_j} N_j \left[D_{\alpha_{cx}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial x} + D_{\alpha_{cz}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial z} \right] \right\} \frac{\partial N_i}{\partial x} d\Omega^e \\
& + \sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \phi^e S_{\alpha_j} \rho_{\alpha_j} N_j \left[D_{\alpha_{cx}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial x} + D_{\alpha_{cz}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial z} \right] \right\} \frac{\partial N_i}{\partial z} d\Omega^e \quad (3.37) \\
& = \sum_{e=1}^{N^e} \int_{\Omega^e} \bar{F}_{\alpha_{c_j}} N_j N_i d\Omega^e + \sum_{e=1}^{N^e} \int_{\Gamma^e} \left\{ \phi^e S_{\alpha_j} \rho_{\alpha_j} N_j \left[D_{\alpha_{cx}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial x} \right. \right. \\
& \left. \left. + D_{\alpha_{cz}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial z} + D_{\alpha_{cx}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial x} + D_{\alpha_{cz}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial z} \right] \right\} n N_i d\Gamma^e
\end{aligned}$$

When retardation is considered in the aqueous phase, the time derivative term in (3.37) becomes,

$$\phi^e r_c S_{\alpha_j} \rho_{\alpha_j} N_j \frac{\partial x_{\alpha_{c_j}}}{\partial t} N_j \quad (3.38)$$

Retardation may be considered for any component of the aqueous phase. When retardation is included, nonequilibrium mass exchange to the solid phase is not allowed for any aqueous phase component.

Phase	Component	Lumped Mass Exchange and Bioreaction Terms	
organic	organic	$\bar{K}_{o\gamma} = \phi E_o$	
		$\bar{F}_{o\gamma} = -\phi E_{ao\gamma} - \phi E_{go\gamma}$	
Aqueous	organic	$\bar{K}_{a\gamma} = \phi E_a + B_a + \phi \rho_a (K_{ao\gamma} + K_{ag\gamma} + K_{ab\gamma} + K_{as\gamma})$	
		$\bar{F}_{a\gamma} = \phi \rho_a (K_{ao\gamma} K_{ao\gamma}^e x_{o\gamma} + K_{ag\gamma} K_{ag\gamma}^e x_{g\gamma} + K_{ab\gamma} x_{b\gamma} + K_{as\gamma} K_{as\gamma}^e x_{s\gamma}) - \mu_{a\gamma}$	
	O_2	$\bar{K}_{ao_2} = \phi E_a + B_a + \phi \rho_a (K_{agO_2} + K_{abO_2}) - \mu_{aO_2}$	
		$\bar{F}_{ao_2} = \phi \rho_a (K_{agO_2} K_{agO_2}^e x_{gO_2} + K_{abO_2} x_{bO_2})$	
	water	$\bar{K}_{aw} = \phi E_a$	
		$\bar{F}_{aw} = -\phi E_{ga_w}$	
	nutrient	$\bar{K}_{aA} = \phi E_a + B_a + \phi \rho_a (K_{agA} + K_{abA}) - \mu_{aA}$	
		$\bar{F}_{aA} = \phi \rho_a (K_{agA} K_{agA}^e x_{gA} + K_{abA} x_{bA})$	
	Gas	organic	$\bar{K}_{g\gamma} = \phi E_g + \phi \rho_g K_{go\gamma}$
			$\bar{F}_{g\gamma} = \phi \rho_g K_{go\gamma} K_{go\gamma}^e x_{o\gamma} - \phi E_{ag\gamma}$
water		$\bar{K}_{gw} = \phi E_g + \phi \rho_g K_{ga_w}$	
		$\bar{F}_{gw} = \phi \rho_g K_{ga_w} K_{ga_w}^e x_{aw}$	
O_2		$\bar{K}_{gO_2} = \phi E_g$	
		$\bar{F}_{gO_2} = -\phi E_{agO_2}$	
nutrient		$\bar{K}_{gA} = \phi E_g$	
		$\bar{F}_{gA} = -\phi E_{agA}$	
Solid	organic	$\bar{K}_{s\gamma}^* = 0$	
		$\bar{F}_{s\gamma}^* = -\phi E_{as\gamma}^*$	
Biophase	organic	$\bar{K}_{b\gamma} = -\mu_{b\gamma}$	
		$\bar{F}_{b\gamma} = -\phi E_{ab\gamma}$	
	oxygen	$\bar{K}_{bO_2} = -\mu_{bO_2}$	
		$\bar{F}_{bO_2} = -\phi E_{abO_2}$	
	nutrient	$\bar{K}_{bA} = -\mu_{bA}$	
$\bar{F}_{bA} = -\phi E_{abA}$			

Table 3.2: Summary of lumped mass exchange and bioreaction coefficients.

Equation (3.37) applies to the mobile aqueous and gas phases. The weighted residual equations for the immobile phases are developed from (3.37) by dropping the advective and dispersive terms. For the organic phase the weak form of the weighted residual equation is,

$$\sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \phi^e S_{o_j} \rho_{o_j} N_j \frac{\partial x_{o_{c_j}}}{\partial t} N_j + \bar{K}_{o_{c_j}} N_j x_{o_{c_j}} N_j \right\} N_i d\Omega^e = \sum_{e=1}^{N^e} \int_{\Omega^e} \bar{F}_{o_{c_j}} N_j N_i d\Omega^e \quad (3.39)$$

A similar equation can be developed for the solid phase. Note that since mass exchange is assumed to be

negligible compared to the mass of the solid phase, $E_s^* = 0$,

$$\sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \rho_s^* \frac{\partial \omega_{s_{c_j}}}{\partial t} N_j + \bar{K}_{s_{c_j}}^* N_j \omega_{s_{c_j}} N_j \right\} N_i d\Omega^e = \sum_{e=1}^{N^e} \int_{\Omega^e} \bar{F}_{s_{c_j}}^* N_j N_i d\Omega^e \quad (3.40)$$

Here, ω_{s_c} is the mass fraction of component c in the solid phase and ρ_s^* is the solid phase bulk density. The mass exchange terms for aqueous-solid interphase mass transfer are also written on a mass basis. When sorption is modeled as a retardation process for a given organic component (3.40) is not solved. When the transport of only one organic component is being considered, MISER has the capability to treat sorption as a two compartment rate limited process. Both compartments can have different Freundlich parameters and mass transfer coefficients. Typically this is used to simulate sorption where one compartment has fast kinetics and is considered to follow equilibrium or near-equilibrium partitioning and the other compartment is substantially rate limited. When using this option the solid mass is divided into two fractions, one for each set of kinetics.

The weighted residual equation for the biophase is expressed as,

$$\sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \phi^e S_{b_j} \rho_{a_j} N_j \frac{\partial x_{b_{c_j}}}{\partial t} N_j + x_{b_{c_j}} \bar{K}_{b_{c_j}} N_j N_j \right\} N_i d\Omega^e = \sum_{e=1}^{N^e} \int_{\Omega^e} \bar{F}_{b_{c_j}} N_j N_i d\Omega^e \quad (3.41)$$

Note that aqueous phase molar density is used for the biophase. Also, S_b is fixed to the volume occupied by the maximum allowable biomass which is calculated by assuming that the biomass and aqueous phase density are equivalent. This implies that $E_b = 0$ and that $B_b = 0$.

3.5.2 Mass Exchange Terms

Nonequilibrium mass exchange is modeled with the linear driving force expression (3.30). The controlling phase for each component is designated based on the relative volatility or solubility of the component in the gas or aqueous phases. For gas-aqueous phase mass exchange of the organic components, oxygen, and the limiting nutrient, the aqueous phase is the controlling phase. Mass exchange of water between the gas and aqueous phases is controlled by the gas phase. Similarly, the aqueous phase is assumed to control sorption rates and aqueous-biophase interactions. For exchange with the organic phase, the adjacent phase will always be the controlling. Table 3.1 summarizes the mass exchange expressions for all components.

The exchange terms presented in Table 3.1 are subject to the following restrictions:

1. Numerical difficulties are experienced when the mass transfer coefficients are larger than required to approximate equilibrium partitioning. To diminish the effect of these difficulties, the mass transfer coefficients are adjusted downward when they exceed the following nodal criterion derived from an analytic solution for one dimension transport with advection and nonequilibrium exchange [Wilkins *et al.*, 1995],

$$K_{\alpha\beta c} \leq -\frac{|q_\alpha|}{\sqrt{2A_{\max}^e}} \ln \left(1 - \frac{x_{\alpha c}^{\text{target}}}{x_{\alpha c}^e} \right) \quad (3.42)$$

where q_α is the effective specific discharge at the node taken as the greater of the advective or diffusive flux over the element, A_{\max}^e is the area of the largest element of which the given node is a member of, and $\left(1 - \frac{x_{\alpha c}^{\text{target}}}{x_{\alpha c}^e} \right)$ is the minimum deviation from equilibrium allowed. Note that this

restriction applies to a single element. Over several elements a much closer approach to equilibrium is possible with MISER (see Section 4.3.4).

2. When the organic phase is present in a given element, organic mass exchange is not considered between the aqueous and gas phases, the aqueous and solid phases, or between the aqueous and biophase.
3. Exchange into the gas phase from the organic phase is not allowed when the gas phase saturation falls below a specified value, (currently 0.05).
4. The exchange terms for organic phase volatilization are adjusted downward when the predicted exchanged component mass for an element is greater than the organic phase component mass present in the element. The exchange terms are reduced by the ratio of the total component mass present in the element to the original predicted component exchange mass.
5. Mass is not allowed to partition from the aqueous or gas phases into the organic phase when only one organic component is present.
6. Oxygen mass transfer from the aqueous phase to the biophase is allowed only when the aqueous phase oxygen mole fraction is positive.
7. Nutrient mass transfer from the aqueous phase to the biophase is only allowed when the aqueous phase nutrient mole fraction is positive.

3.6 BIOLOGICAL REACTIONS

Biological activity is described with Monod-type kinetic expressions. As shown in Table 3.2 these expressions can be inserted directly into the aqueous phase component transport equations or into a separate biophase component transport equations when mass transfer rate limitations to the biomass are to be considered. The nonlinear rate coefficient $\mu_{\alpha i}$ takes three forms as follows, one form for the degradation of organic substrates, one for the utilization of oxygen, and the last for the utilization of nutrient,

$$\mu_{\alpha i} = k_l X \left(\frac{1}{K_{s_i} + x_{\alpha i}} \right) \left(\frac{x_{\alpha O_2}}{K_{s_{O_2}} + x_{\alpha O_2}} \right) \left(\frac{x_{\alpha A}}{K_{s_A} + x_{\alpha A}} \right) I_l I_{O_2} I_A I_S \quad (3.43a)$$

$$\mu_{\alpha O_2} = \sum_l F_{O_2 l} k_l X \left(\frac{x_{\alpha i}}{K_{s_i} + x_{\alpha i}} \right) \left(\frac{1}{K_{s_{O_2}} + x_{\alpha O_2}} \right) \left(\frac{x_{\alpha A}}{K_{s_A} + x_{\alpha A}} \right) I_l I_{O_2} I_A I_S \quad (3.43b)$$

$$\mu_{\alpha A} = \sum_l F_{A l} k_l X \left(\frac{x_{\alpha i}}{K_{s_i} + x_{\alpha i}} \right) \left(\frac{x_{\alpha O_2}}{K_{s_{O_2}} + x_{\alpha O_2}} \right) \left(\frac{1}{K_{s_A} + x_{\alpha A}} \right) I_l I_{O_2} I_A I_S \quad (3.43c)$$

Biomass growth, maintenance, and decay is governed by an ordinary differential equation,

$$\frac{dX}{dt} = \left(\sum_l (Y_l \mu_{\alpha l} x_{\alpha l} I_{max}) - K_d I_{min} \right) X \quad (3.44)$$

In order to reduce negative mole fractions in either the biophase transport equations when considering a separate biophase, or in the aqueous phase transport equations when they have the bioreaction term inserted

directly, the size of the bioreaction term is limited in some cases. If the calculated bioreaction term over the course of the current time step will consume more of the component to which it applies than is available, the bioreaction term is reduced by the ratio of available component mass to projected required component mass. The available component mass is defined as the mass of component present in the appropriate phase plus the projected amount of component to be delivered to the appropriate phase through interphase mass exchange. Other affected bioreaction terms are also reduced. For instance, if a organic component reaction rate is reduced, that component's contribution to oxygen and nutrient usage, as well as to biomass production, must be accounted for. A reduction in oxygen or nutrient reaction rate affects all the organic components and biomass production. In addition, Monod terms are zero when the corresponding component is below a preselected minimum detectable value, currently set to 1 ppb by mass.

3.7 NAPL SATURATION

The numerical formulation is completed with an expression describing the change in organic phase saturation. Since the organic phase is assumed to be immobile, changes in organic saturation result solely from interphase mass transfer as indicated in (2.7). Expanding the time derivative term of this equation and substituting the summation over the phase exchange terms as previously defined gives,

$$\phi S_o \frac{\partial \rho_o^*}{\partial t} + \phi \rho_o^* \frac{\partial S_o}{\partial t} = \phi E_o^* \quad (3.45)$$

Because the organic phase is assumed to be incompressible, $\frac{\partial \rho_o^*}{\partial t}$ is strictly a function of composition. Furthermore, since the organic phase saturation is updated after convergence of the phase compositions, the density derivative term is known and can be moved to the right hand side along with the exchange terms,

$$\phi \rho_o^* \frac{\partial S_o}{\partial t} = \phi E_o^* - \phi S_o \frac{\partial \rho_o^*}{\partial t} \quad (3.46)$$

The weighted residual equation for (3.46) is then developed as,

$$\sum_{e=1}^{N^e} \int_{\Omega^e} \left\{ \phi^e \rho_{o_j}^* N_j \frac{\partial S_{o_j}}{\partial t} N_j \right\} N_i d\Omega^e = \sum_{e=1}^{N^e} \int_{\Omega^e} \phi \left\{ E_{o_j}^* N_j - S_{o_j} N_j \frac{\partial \rho_{o_j}^*}{\partial t} N_j \right\} N_i d\Omega^e \quad (3.47)$$

The solution of (3.47) is obtained by mass lumping the left hand side. When the organic phase consists of only one component, the resulting finite element equation is explicit in organic phase saturation since the density derivative term is zero. When more than one organic phase component is present, (3.47) is solved iteratively because the organic phase saturation appears on both sides of the equation. Additionally, the organic phase saturation is never allowed to fall below zero.

3.8 AXISYMMETRIC COORDINATES

An axisymmetric coordinate system (r - z) is simulated by multiplying the element matrices by [Huyakorn and Pinder, 1983],

$$\bar{r}_e = \frac{2\pi}{3} (r_1 + r_2 + r_3) \quad \text{for } (r\text{-}z) \quad (3.48)$$

where r_i is the radial coordinate of node i . The horizontal coordinate x is then taken to represent the radial distance r . When simulating a cross sectional (x - z) domain, \bar{r}_e is set to unity,

$$\bar{r}_e = 1 \quad \text{for } (x-z) \quad (3.49)$$

3.9 TIME DISCRETIZATION

Consider the following generic finite element equation,

$$[A] \frac{\partial \{u\}}{\partial t} + [B] \{u\} = \{F\} \quad (3.50)$$

A standard finite difference approach is used to discretize the time derivative in all the finite element equations developed above [Huyakorn and Pinder, 1983],

$$\frac{\partial \{u\}}{\partial t} = \frac{\{u\}^{k+1,t+1} - \{u\}^t}{\Delta t} \quad (3.51)$$

where k is an iteration counter; Δt is the time step; and the superscript $t + 1$ represents the $t + \Delta t$ time level. A variable time weighting factor, θ , is included in MISER and is defined as,

$$\{u\}^{k+1,t+1} = \theta \{u\}^{k+1,t+1} + (1 - \theta) \{u\}^t \quad (3.52)$$

where $\theta = 1$ is used for fully implicit time stepping and $\theta = 1/2$ is used for Crank-Nicholson time stepping. Substituting (3.51) and (3.52) into (3.50) yields,

$$\left[\frac{1}{\Delta t} [A] + \theta [B] \right] \{u\}^{k+1,t+1} = \left[\frac{1}{\Delta t} [A] - (1 - \theta) [B] \right] \{u\}^t + \{F\}^{k,t+1} \quad (3.53)$$

To reduce potential errors due to limited computer precision, the dependent variable is expressed as a difference over the time step, Δu^{t+1} ,

$$\{u\}^{k+1,t+1} = \{u\}^t + \{\Delta u\}^{k+1,t+1} \quad (3.54)$$

and substituting (3.54) into (3.53) yields,

$$\left[\frac{1}{\Delta t} [A] + \theta [B] \right] \{\Delta u\}^{k+1,t+1} = -[B] \{u\}^t + \{F\}^{k,t+1} \quad (3.55)$$

Equation (3.55) is not used for the calculation of the velocities when (3.23) and (3.24) are solved. In this case difference equation (3.53) is employed.

3.10 TIME STEP CONTROL

The time discretization starts with a prescribed time step size. This time step is adjusted automatically in accordance with the following set of rules:

1. The time step size cannot be smaller than a prescribed minimum value.

2. The time step size cannot be larger than a prescribed maximum value.
3. During a given time step, if the number of iterations required for convergence of the flow equations and for convergence of the transport equations is less than prescribed numbers, the time step is increased by multiplying the current time step by a prescribed constant (> 1). Different values of the prescribed number of iterations may be specified for the flow and transport equations.
4. During a given time step, if convergence of the flow equations or of the transport equations is not attained within a specified maximum number of iterations, the time step is decreased by multiplying the current time step by a constant (< 1). Different values of the maximum number of iterations may be specified for the flow and transport equations. If convergence is repeatedly not attained, the simulation will terminate when a minimum time step size is reached.
5. Solution of the flow equations may be skipped for a specified number of time steps. The transport equations are always solved on every time step. Adjustment of the time step size proceeds as described above, however, the use of small multipliers for increasing the time step size is advised (< 1.1).

3.11 BOUNDARY CONDITIONS

3.11.1 Phase Mass Balance Boundary Conditions

Boundary conditions must be specified for the aqueous and gas phase mass balance equations as either constant specified pressure (type I) or constant specified flux (type II).

Constant pressure conditions may be specified at any node in the computation domain. This condition is implemented by modifying the appropriate row in the global matrix equation (3.18) to,

$$\Delta P_{\alpha_i}^{t+1} = 0 \quad (3.56)$$

where i is the node at which constant pressure conditions are specified.

Specified boundary fluxes are introduced through the source/sink terms. The flow across a boundary segment of element e is represented by the surface integrals on the RHS of (3.17) and is expressed as,

$$Q_{\alpha_i} = - \sum_{e=1}^{N^e} \int_{\Gamma^e} q_{\alpha_e} N_i d\Gamma^e = - \sum_{e=1}^{N^e} q_{\alpha_i} \frac{A_{x_i}}{2} \quad (3.57)$$

where q_{α_i} is the boundary flux [LT^{-1}] of phase α at the boundary node i and Q_{α_i} is the total discharge at node i [L^3T^{-1}]. No flow boundaries ($q_{\alpha} = 0$) are the natural finite element boundary condition. The boundary flux is assumed to be uniform over the cross sectional area, A_{x_i} , associated with the boundary node i . This cross sectional area depends on the coordinate system and thus the specified flow must properly reflect the cross sectional area.

Due to compressibility effects the specified volumetric flux of the gas phase is referenced to free surface conditions (fs) conditions which are assumed to 1 atmosphere of pressure at 20 °C. The free surface

conditions are related to the reservoir conditions (rc) by a compressibility factor, G_g [-], which is evaluated from the ideal gas law (2.49),

$$Q_{g_{rc}} = Q_{g_{fs}} G_g = Q_{g_{fs}} \left(\frac{P_{g_{fs}} T_{rc}}{P_{g_{rc}} T_{fs}} \right) \quad (3.58)$$

3.11.2 Component Mass Balance Boundary Conditions

Boundary conditions are required for the gas and aqueous phase component balance equations. These may be specified as either: constant specified mole fraction (type I); specified diffusional flux (type II),

$$\phi D_{\alpha_c jj}^h \frac{\partial x_{\alpha_c}}{\partial x_j} n_j = \frac{D_{\alpha_c}^m}{L_{\alpha\Gamma}} (x_{\alpha_c}^o - x_{\alpha_c}) \quad (3.59)$$

where $L_{\alpha\Gamma}$ is the thickness of the stagnant boundary layer in the contacting α phase fluid and $x_{\alpha_c}^o$ is the specified value of the component mole fraction in the contacting fluid; or mixed (type III) where the composition of an incoming fluid is specified along a boundary section,

$$-\phi D_{\alpha_c jj}^h \frac{\partial x_{\alpha_c}}{\partial x_j} n_j + q_{\alpha_j} n_j x_{\alpha_c} = q_{\alpha_j} n_j x_{\alpha_c}^o \quad (3.60)$$

When the boundary is impermeable or water flow is directed out of the domain, (3.60) reduces to (3.59). Eqs. (3.59) and (3.60) are expressed at all nodes to which a given boundary condition applies.

Constant mole fraction boundary conditions may be specified at any node in the computational domain. This condition is implemented by modifying the appropriate row in the global matrix equation (3.55) to,

$$\Delta x_{\alpha_c i}^{t+1} = 0 \quad (3.61)$$

where i is the node at which constant mole fraction conditions are specified.

Zero diffusive flux type II boundaries are the natural finite element boundary condition for the transport equation, i.e.,

$$\phi D_{\alpha_c jj}^h \frac{\partial x_{\alpha_c}}{\partial x_j} n_j = 0 \quad (3.62)$$

This boundary condition corresponds to setting the surface integral term on the RHS of (3.37) to 0. This condition is used to represent those boundaries where advective transport is directed outward and there is no mole fraction gradient at the boundary. Since the NAPL, solid, and biophase component transport equations do not consider advective or diffusive flux no additional specification of boundary conditions is required beyond the natural condition. This is also true for the NAPL saturation equation.

Nonzero type II and type III boundary fluxes are introduced through the surface integral on the RHS of (3.37). For type III boundaries, (3.60) becomes,

$$\sum_{e=1}^{N^e} \int_{\Gamma^e} \left\{ \phi^e S_{\alpha_j} \rho_{\alpha_j} D_{\alpha_c jj}^{h^e} \frac{\partial x_{\alpha_c}}{\partial x_j} N_j N_j \right\} N_i n_j d\Gamma^e = Q_{\alpha_j} n_j x_{\alpha_c} - Q_{\alpha_j} n_j x_{\alpha_c}^o \quad (3.63)$$

The first term on the RHS of (3.63) represents the advective flux and is inserted into the matrix $[B]$ in (3.55). The second term is the total material flux and is added to the vector $\{F\}$ in (3.55). The boundary flux is evaluated in two ways. When the flow equations are solved, the boundary flux calculated for the

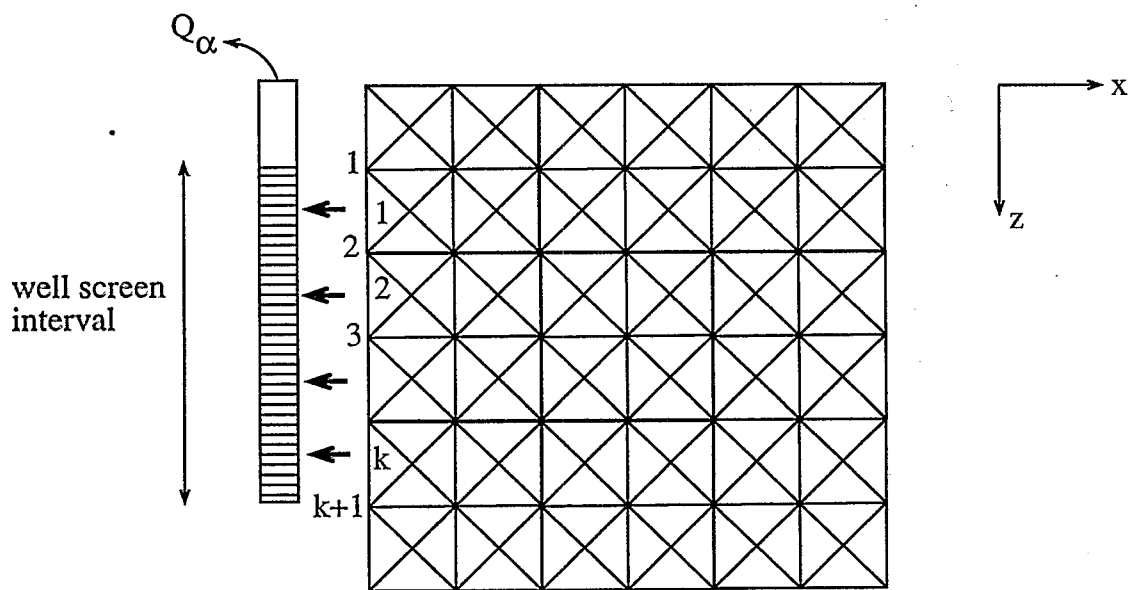


Figure 3.3: Representation of an extraction well in the discretized domain.

phase material balance calculation, or imposed as a boundary condition for the flow equations is used to evaluate Q_α . Otherwise the boundary flux is evaluated using the Darcy flux in the following fashion. The boundary flux is assumed to be uniform over the cross sectional area, A_{x_i} , associated with the boundary node i . The cross sectional area depends on the coordinate system and is evaluated by,

$$A_{x_i} = l_i \quad \text{for } (x-z) \quad (3.64a)$$

$$A_{x_i} = 2\pi\bar{r}_i l_i \quad \text{for } (r-z) \quad (3.64b)$$

Here $l_i = \frac{1}{2}(l_{e1} + l_{e2})$ is the length of the boundary segment associated with node i and l_{e1} and l_{e2} are the lengths of the boundary segments of elements $e1$ and $e2$ which are connected at node i . The average radius of the boundary segment associated with node i is $\bar{r}_i = \frac{1}{4}(r_{i-1} + 2r_i + r_{i+1})$. Here r_{i-1} , r_i , and r_{i+1} are the radii of the three nodes spanning $e1$ and $e2$. Nonzero type II boundaries are treated in the same manner as nonzero type III boundaries with the RHS of (3.63) being replaced by the RHS of (3.59) after evaluation of the surface integral.

3.11.3 Extraction Wells

MISER incorporates the option to simulate radial flow to an extraction well positioned in the center of axisymmetric ($r-z$) domains. Consider the problem of simulating an extraction well along the left boundary of the domain shown in Figure 3.3. The well screen spans k elements and $k + 1$ nodes, and is assumed to be parallel to the z -coordinate.

Vadose zone extraction wells operate in regions of variable saturation and therefore typically remove both aqueous and gas phases during operation. MISER requires the specification of the total combined (aqueous and gas) extraction rate and then apportions the flow of each phase at each node along the well

screen. The flow of each phase is apportioned along the well screen by [Aziz and Settari, 1979],

$$Q_{\alpha_{is}} = \sum_{e=1}^k \left[\frac{\frac{1}{2} A_{x_i} \lambda_{\alpha_i} (\nabla P_{\alpha_i} - \rho_{\alpha_i}^* g \nabla z)}{\sum_{e=1}^k \sum_{j=1}^2 \frac{1}{2} A_{x_j} [\lambda_{a_j} (\nabla P_{a_j} - \rho_{a_j}^* g \nabla z) + \lambda_{g_j} (\nabla P_{g_j} - \rho_{g_j}^* g \nabla z)]} \right] Q_{ttfs} \quad (3.65)$$

where Q_{ttfs} is the specified combined discharge at free surface conditions, and j denotes the nodes along the well screen in element e . The cross sectional area associated with each node along the well screen is computed by,

$$A_{x_i} = 2\pi r_{well} l_i \quad (3.66)$$

where r_{well} is the radius of the well, $l_i = \frac{1}{2}(l_{e1} + l_{e2})$ is the vertical length of well segment associated with node i , and l_{e1} and l_{e2} are the lengths of the element $e1$ and $e2$ which are connected at node i . Here it is assumed that well is parallel with the z -coordinate.

Further simplification is made by assuming that the potential gradient for each phase is the same in all layers. The flows may then be apportioned solely by mobility,

$$Q_{\alpha_{irc}} = \sum_{e=1}^k \left[\frac{A_{x_i} G_{\alpha_i} \lambda_{\alpha_{x_i}}}{\sum_{e=1}^k \sum_{j=1}^2 A_{x_j} (\lambda_{a_{x_j}} + \lambda_{g_{x_j}})} \right] Q_{ttfs} \quad (3.67)$$

where the aqueous compressibility is set to 1, and the gas phase compressibility is computed from (3.58). This 'transmissibility allocation' method is generally valid when variation in permeability along the well is small. However, the method "may give erroneous results in the case of large vertical heterogeneity and especially when non-communicating layers exist" [Aziz and Settari, 1979].

The appropriate boundary condition for the transport equations at an extraction well is type II with zero diffusive flux.

3.11.4 Treatment of Injection Wells

Presently MISER is configured to consider the injection of gas phase only. Thus, under injection conditions the specified total discharge, Q_{ttfs} , is comprised solely of gas phase which is similarly apportioned along the well screen by the transmissibility allocation methods, using,

$$Q_{g_{irc}} = \sum_{e=1}^k \left[\frac{A_{x_i} B_{g_i} \lambda_{g_{x_i}}}{\sum_{e=1}^k \sum_{j=1}^2 A_{x_j} \lambda_{g_{x_j}}} \right] Q_{gtfs} \quad (3.68)$$

In the case when the denominator in (3.68) is zero (i.e. injection below the water table), then Q_{ttfs} is distributed uniformly over the well screen.

The appropriate boundary condition for the transport equations at an injection well is type III which specifies the composition of the incoming fluid.

3.12 ITERATION SCHEME

An iteration scheme is necessary to update secondary variables that are functions of the primary solution variables. A Picard iteration approach is employed in MISER [Huyakorn and Pinder, 1983]. This approach is implemented by simply lagging the secondary variables and iterating within a time step until a convergence criterion is satisfied.

A relative maximum differential is used to test convergence. For the flow equations convergence is established by the change in pressure, evaluated by,

$$\frac{\max_j |P_\alpha^{k+1} - P_\alpha^k|}{\max_j |P_\alpha^{k+1}|} \leq \epsilon_f \quad (3.69)$$

where ϵ_f is the convergence tolerance for solution of the flow equations.

Mole fractions, x_{α_i} , are used to control convergence of the transport equations. The transport equations for the gas, aqueous, organic, solid and biophases are solved sequentially until convergence is achieved for each equation. Two values of the convergence criteria are used, one for the gas and aqueous phase transport equations, ϵ_m , and one for the immobile phase transport equations, ϵ_i . Convergence is measured by,

$$\frac{\max_j |x_{\alpha_j}^{k+1} - x_{\alpha_j}^k|}{\max_j |x_{\alpha_j}^{k+1}|} \leq \epsilon_m \text{ or } \epsilon_i \quad (3.70)$$

For the solid phase, convergence is measured using solid phase mass loadings, ω_{s_c} , otherwise the expression is the same as (3.70).

After convergence has been obtained for all the transport equations (3.47) is solved for the organic phase saturations. An iterative solution is not required if the organic phase is composed of only a single component. When more than one organic phase component is present, convergence is measured using the organic phase saturations,

$$\frac{\max_j |S_{o_j}^{k+1} - S_{o_j}^k|}{\max_j |S_{o_j}^{k+1}|} \leq \epsilon_o \quad (3.71)$$

where ϵ_o is the convergence criterion for the organic phase saturation. When a nodal organic phase mole fraction falls below a user specified value (currently 10^{-3}), convergence is not measured for that nodal mole fraction.

Table 3.3 summarizes the numerical iteration scheme employed in MISER.

3.13 SOLVER

The nonlinear discretized flow and gas and aqueous phase transport equations are solved using the Harwell Sparse Matrix Package [Duff, 1979]. The specific flux equations are solved directly when calculating element fluxes. When calculating nodal velocities the Harwell package is used, unless the mass matrix is lumped. In that case, the specific fluxes are solved for directly. The NAPL and solid phase transport equations can also be solved using the Harwell Sparse Matrix Package. The NAPL and solid phase

Numerical Solution Scheme		
Sequence	Description	Section Reference
	Problem Setup	
1.	Read input files and initialize variables as needed.	5.4
2.	Calculate element areas.	3.1
3.	Calculate initial mass in domain.	4.1
	Begin Iteration Over Time	
4.	Update phase molecular and mass densities.	2.5.3 and 2.5.4
	Solve Flow Equations	
5.	Update gas phase viscosity.	2.5.5
6.	Begin iterative solution of flow equations.	3.2
6a.	Update capacity coefficients and mobilities.	3.2.3, 2.5.2
6b.	Update well terms.	3.11.3, 3.11.4
6c.	Solve flow equations after imposing boundary conditions.	3.11.1
6d.	Update gas and aqueous phase saturations.	2.5.1
6e.	Update gas phase mole and mass density.	2.5.3
6f.	Check convergence. Iterate to 6. if not converged.	3.12
7.	Update boundary fluxes for material balance and transport equation boundary conditions	4.1.1
8.	Update phase material balance.	4.1
	Solve Transport Equations	
9.	Update velocity field.	3.4
10.	Update bioreaction terms and biomass.	3.6
	Only when a separate biophase is not present.	
11.	Begin iterative solution of transport equations.	3.5
11a.	Update biophase mole fractions and biomass.	3.6
	Only when a separate biophase is present.	
11b.	Update phase molecular and mass densities.	2.5.3 and 2.5.4
11c.	Update mass exchange terms.	3.5
11d.	Update dispersion coefficients.	2.5.7
11e.	Solve gas and aqueous phase transport equations after imposing boundary conditions.	3.5.1 and 3.11.2
11f.	Solve organic phase transport equations.	3.5.1
11g.	Solve solid phase transport equations.	3.5.1
11h.	Solve organic phase saturation equation.	3.7
11i.	Update gas and aqueous saturations if the flow equations are not included.	
11j.	Check convergence. Iterate to 11. if not converged.	3.12
	Increment to Next Time Level	
12.	Update component material balance.	4.1.2
13.	Print output if desired.	
14.	Adjust time step if indicated and proceed to next time level.	3.10

Table 3.3: Summary of numerical scheme in MISER.

transport equations only have the time derivative term on the left hand side and are solved directly when mass lumped. In contrast, when considering a separate biophase, the biophase transport equations have exchange and bioreaction terms on the left hand side and hence are not diagonal after mass lumping. Thus, the biophase transport equations are always solved using the Harwell package. The NAPL saturation equation is always mass lumped and solved directly.

Section 4

MODEL VERIFICATION

4.1 MATERIAL BALANCE CALCULATION

A material balance calculation is included as an option in MISER. This option calculates a mass balance error as a measure of the material balance in the numerical solution of the flow and transport equations. The mass balance error is calculated over the entire computational region Ω .

4.1.1 Phase Material Balance

Mass balance error in solutions to the flow equations is obtained by integrating (2.6) over Ω and applying the divergence theorem,

$$\int_{\Omega} \frac{\partial}{\partial t} (\phi \rho_{\alpha}^* S_{\alpha}) d\Omega + \int_{\Gamma} [\rho_{\alpha}^* \lambda_{\alpha} (\nabla P_{\alpha} - \rho_{\alpha}^* g \nabla z)] \cdot n d\Gamma - \int_{\Omega} \phi E_{\alpha}^* d\Omega - \int_{\Omega} R_{\alpha} d\Omega = 0 \quad (4.1)$$

where Γ is the boundary of Ω .

If the numerical solution is substituted into (4.1) then the mass balance residual of phase α may be calculated by,

$$\varepsilon_{\alpha} = F_{s_{\alpha}} + F_{b_{\alpha}} + F_{e_{\alpha}} + F_{r_{\alpha}} \quad (4.2)$$

where,

$$F_{s_{\alpha}} = \int_{\Omega} \frac{\partial}{\partial t} (\phi \rho_{\alpha}^* S_{\alpha}) d\Omega \approx \sum_{e=1}^{N^e} \int_{\Omega} \frac{\phi^e}{\Delta t} (\rho_{\alpha_i}^{*t+1} S_{\alpha_i}^{t+1} - \rho_{\alpha_i}^{*t} S_{\alpha_i}^t) N_i d\Omega \quad (4.3a)$$

$$F_{b_{\alpha}} = \int_{\Gamma} [\rho_{\alpha}^* \lambda_{\alpha} (\nabla P_{\alpha} - \rho_{\alpha}^* g \nabla z)] \cdot n d\Gamma \quad (4.3b)$$

$$F_{e_{\alpha}} = - \int_{\Omega} \phi E_{\alpha}^* d\Omega \approx - \sum_{e=1}^{N^e} \int_{\Omega} \phi^e E_{\alpha_i}^* N_i d\Omega \quad (4.3c)$$

$$F_{r_{\alpha}} = - \int_{\Omega} R_{\alpha} d\Omega \approx - \sum_j \rho_{\alpha_j}^* Q_{\alpha_j} \quad (4.3d)$$

Here mass lumping is employed to evaluate the volume integrals. The boundary integrals are evaluated by back substitution of the predicted pressures into the finite element equations [Huyakorn and Pinder, 1983].

4.1.2 Component Material Balance

Mass balance error in solutions to the transport equations is obtained by integrating (2.11) over Ω , applying the divergence theorem, and substituting $\mu_{\alpha_c} x_{\alpha_c}$ for B_{α_c} . The resulting equation is summed over the phases giving,

$$M_c \sum_{\alpha} \int_{\Omega} \frac{\partial}{\partial t} (\phi \rho_{\alpha} S_{\alpha} r_c x_{\alpha_c}) d\Omega - M_c \sum_{\alpha} \int_{\Gamma} (\phi \rho_{\alpha} S_{\alpha} D_{\alpha_{c ij}}^h \nabla x_{\alpha_c}) \cdot n d\Gamma - M_c \sum_{\alpha} \int_{\Omega} \mu_{\alpha_c} x_{\alpha_c} d\Omega = 0 \quad (4.4)$$

where M_c is the component molecular weight. Retardation ($r_c \neq 1$) is included only when $\alpha = a$ and nonequilibrium sorption is not considered.

If the numerical solution is substituted into (4.4) then the component residual for the entire domain may be calculated by,

$$\varepsilon_c = F_{s_c} + F_{b_c} + F_{\mu_c} \quad (4.5)$$

where,

$$F_{s_c} = M_c \sum_{\alpha} \int_{\Omega} \frac{\partial}{\partial t} (\phi \rho_{\alpha} S_{\alpha} r_c x_{\alpha_c}) d\Omega \approx M_c \sum_{\alpha} \sum_{e=1}^{N_e} \int_{\Omega} \frac{\phi^e}{\Delta t} (\rho_{\alpha_i}^{t+1} S_{\alpha_i}^{t+1} r_c x_{\alpha_{c_i}}^{t+1} - \rho_{\alpha_i}^t S_{\alpha_i}^t r_c x_{\alpha_{c_i}}^t) N_i d\Omega \quad (4.6a)$$

$$F_{b_c} = M_c - \sum_{\alpha} \int_{\Gamma} (\phi \rho_{\alpha} S_{\alpha} D_{\alpha_{c ij}}^h \nabla x_{\alpha_c}) \cdot n d\Gamma \quad \text{for } \Gamma_i = \text{Type I} \quad (4.6b)$$

$$\approx M_c \sum_{\alpha} \sum_{i=1}^{N_{II}} \rho_{\alpha_i} Q_{\alpha}^{\Gamma_i} x_{\alpha_{c_i}} N_i \quad \text{for } \Gamma_i = \text{Type II}$$

$$\approx M_c \sum_{\alpha} \sum_{i=1}^{N_{III}} \int_{\gamma} \rho_{\alpha_i} Q_{\alpha}^{\Gamma_i} x_{\alpha_{c_i}}^o N_i \quad \text{for } \Gamma_i = \text{Type III}$$

$$F_{\mu_c} = -M_c \sum_{\alpha} \int_{\Omega} \mu_{\alpha_{c_i}} x_{\alpha_{c_i}} d\Omega \approx -M_c \sum_{\alpha} \sum_{e=1}^{N_e} \mu_{\alpha_{c_i}} x_{\alpha_{c_i}} N_i \quad (4.6c)$$

The type I boundary integrals are evaluated by back substitution of the predicted mole fractions into the finite element equations [Huyakorn and Pinder, 1983]. When $r_c \neq 1$, the change in component mass determined with (4.6a) includes the change in sorbed mass.

4.1.3 Calculation of Mass Balance Error

The accuracy of the numerical scheme is evaluated by three measures of mass balance error. The first two are relative errors, and the third represents an absolute error. The expressions used to calculate these errors

are respectively,

$$B_{\alpha 1}^{\Delta t} = 100 \left(\frac{|\varepsilon_{\alpha} \Delta t|}{\sum_{e=1}^{N^e} \phi^e \int_{\Omega} \rho_{\alpha i}^{*0} S_{\alpha i}^0 N_i d\Omega} \right) \quad (4.7a)$$

$$B_{\alpha 2}^{\Delta t} = 100 \left(\frac{|\varepsilon_{\alpha}|}{\max [(|F_{b_{\alpha}}| + |F_{e_{\alpha}}| + |F_{r_{\alpha}}|), |F_{s_{\alpha}}|] } \right) \quad (4.7b)$$

$$B_{\alpha 3}^{\Delta t} = 100 \left(1 - \left| \frac{F_{b_{\alpha}} + F_{e_{\alpha}} + F_{r_{\alpha}}}{F_{s_{\alpha}}} \right| \right) \quad (4.7c)$$

$$B_{c 1}^{\Delta t} = 100 \left(\frac{|\varepsilon_c \Delta t|}{M_c \sum_{\alpha} \sum_{e=1}^{N^e} \phi^e \int_{\Omega} \rho_{\alpha i}^0 S_{\alpha i}^0 x_{\alpha c_i}^0 N_i d\Omega} \right) \quad (4.7d)$$

$$B_{c 2}^{\Delta t} = 100 \left(\frac{|\varepsilon_c|}{\max [(|F_{b_c}| + |F_{\mu_c}|), |F_{s_c}|] } \right) \quad (4.7e)$$

$$B_{c 3}^{\Delta t} = 100 \left(1 - \left| \frac{F_{b_c} + F_{\mu_c}}{F_{s_c}} \right| \right) \quad (4.7f)$$

where the denominators in (4.7a) and (4.7d) are the mass storages of phase α and component c respectively at the start of the simulation. $B_{\alpha 1}$, $B_{\alpha 2}$, $B_{\alpha 3}$, $B_{c 1}$, $B_{c 2}$, and $B_{c 3}$ are set to zero if the denominator is zero.

The error measures above represent the percentage mass balance error over a single time step. Cumulative mass balance errors are also computed using,

$$B_{\alpha 1}^t = 100 \left(\frac{\left| \frac{\sum \varepsilon_{\alpha} \Delta t}{\Delta t} \right|}{\sum_{e=1}^{N^e} \phi^e \int_{\Omega} \rho_{\alpha i}^{*0} S_{\alpha i}^0 N_i d\Omega} \right) \quad (4.8a)$$

$$B_{\alpha 2}^t = 100 \left(\frac{\left| \frac{\sum \varepsilon_{\alpha} \Delta t}{\Delta t} \right|}{\max \left[\left(\left| \sum_{\Delta t} F_{b_{\alpha}} \Delta t \right| + \left| \sum_{\Delta t} F_{e_{\alpha}} \Delta t \right| + \left| \sum_{\Delta t} F_{r_{\alpha}} \Delta t \right| \right), \left| \sum_{\Delta t} F_{s_{\alpha}} \Delta t \right| \right]} \right) \quad (4.8b)$$

$$B_{\alpha 3}^t = 100 \left(1 - \left| \frac{\sum (F_{b_{\alpha}} + F_{e_{\alpha}} + F_{r_{\alpha}}) \Delta t}{\sum F_{s_{\alpha}} \Delta t} \right| \right) \quad (4.8c)$$

$$B_{c 1}^t = 100 \left(\frac{\left| \frac{\sum \varepsilon_c \Delta t}{\Delta t} \right|}{M_c \sum_{\alpha} \sum_{e=1}^{N^e} \phi^e \int_{\Omega} \rho_{\alpha i}^0 S_{\alpha i}^0 x_{\alpha c_i}^0 N_i d\Omega} \right) \quad (4.8d)$$

$$B_{c2}^t = 100 \left(\frac{\left| \sum_{\Delta t} \varepsilon_c \Delta t \right|}{\max \left[\left(\left| \sum_{\Delta t} F_{b_c} \Delta t \right| + \left| \sum_{\Delta t} F_{\mu_c} \Delta t \right| \right), \left| \sum_{\Delta t} F_{s_c} \Delta t \right| \right]} \right) \quad (4.8e)$$

$$B_{c3}^t = 100 \left(1 - \frac{\left| \sum_{\Delta t} (F_{b_c} + F_{\mu_c}) \Delta t \right|}{\sum_{\Delta t} F_{s_c} \Delta t} \right) \quad (4.8f)$$

4.2 VERIFICATION OF THE PHASE MASS BALANCE SOLUTIONS

4.2.1 Comparison with One Dimensional Richards Equation

Numerical solutions of the flow equations were compared with analytical solutions of the Richards equation. This equation describes the movement of a constant density aqueous phase in variably saturated media under the assumption that the gas phase does not impede the liquid migration; i.e. the gas pressure is static. Thus, comparisons with solutions of Richards equation provides verification for the solution of the aqueous flow equation only, even though both aqueous and gas flow equations are solved.

The one dimensional vertical form of Richards equation is expressed as,

$$\phi \frac{\partial S_a}{\partial t} - \frac{\partial}{\partial z} \left[\frac{k k_{ra}}{\mu_a} \left(\frac{\partial P_a}{\partial z} - \rho_a^* g \right) \right] = 0 \quad (4.9)$$

Semi-analytical solutions of (4.9) developed by Philip [1969] were compared to MISER. The test problem considered is for vertical moisture infiltration under constant surface ponding into a soil with an initial moisture content close to residual. Conditions of the test problem were obtained from Celia *et al.*, [1990]. The asymptotic characteristic of the capacity coefficient in the region of residual water saturation creates computational difficulties for numerical simulators. Therefore, this test problem provides a rigorous test of MISER for typical moisture infiltration conditions in the unsaturated zone. The hydraulic properties of the test problem are,

$$S_a = \frac{1 - S_{ra}}{[1 + (\alpha P_{c_{ga}})^n]^m} + S_{ra} \quad (4.10)$$

$$k_{ra} = \frac{\{1 - (\alpha P_{c_{ga}})^{n-1} [1 + (\alpha P_{c_{ga}})^n]^{-m}\}^2}{[1 + (\alpha P_{c_{ga}})^n]^{m/2}} \quad (4.11)$$

where $\phi = 0.368$; $S_{ra} = 0.2772$; $n = 2$; $m = 0.5$; $\alpha = 3.415 \times 10^{-4} \text{ Pa}^{-1}$; and $k = 9.43435 \times 10^{-12} \text{ m}^2$. The initial and boundary conditions are:

$$\begin{array}{lll} P_a(x, z, t = 0) = -1000 \text{ cm} = -98071 \text{ Pa} & P_g(x, z, t = 0) = 0 \text{ Pa} & (S_a = 0.299) \\ P_a(x, z = 0, t) = -75 \text{ cm} = -7355.325 \text{ Pa} & P_g(x, z = 0, t) = 0 \text{ Pa} & (S_a = 0.544) \\ P_a(x, z = L, t) = -1000 \text{ cm} = -98071 \text{ Pa} & P_g(x, z = L, t) = 0 \text{ Pa} & (S_a = 0.299) \\ \partial P_a(x = 0, z, t) / \partial x = 0 & \partial P_g(x = 0, z, t) / \partial x = 0 & \\ \partial P_a(x = 4 \text{ cm}, z, t) / \partial x = 0 & \partial P_g(x = 4 \text{ cm}, z, t) / \partial x = 0 & \end{array}$$

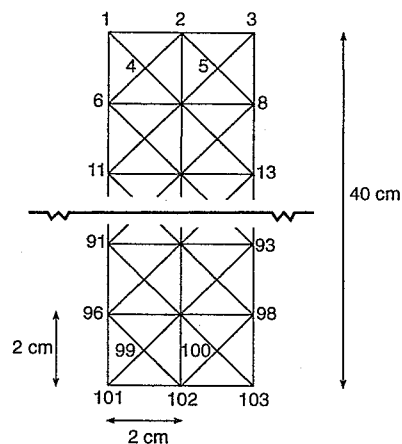


Figure 4.1: Computational grid used for the numerical solution of the one dimensional Richards equation.

Numerical simulation of the moisture infiltration problem was obtained on a symmetric “union jack” grid consisting of 160 elements and 103 nodes (Figure 4.1). Nodes were uniformly spaced in the x and z directions at 2 cm. The gas phase was simulated with three components: nitrogen, oxygen, and water. Since transport equations were not solved, the gas phase composition was fixed, with a mass density of 1.24 g/l. To reduce the effect of gas pressure transients on aqueous migration, the vapor viscosity of the three components were reduced to an artificially small value of 1×10^{-7} Pa-s. The aqueous phase was simulated as pure water.

Figure 4.2 compares the predicted and analytical solutions at time 6 hrs. Close agreement was generally obtained for pressure and saturation distributions. Pronounced oscillations at the toe of the sharp front are evident in the pressure distribution when the consistent form of the mass matrix is used. The oscillations are eliminated when the mass matrix is lumped. This behavior is consistent with that commonly observed in the numerical solution of Richards equation [Milly, 1985; Celia *et al.*, 1990; Rathfelder and Abriola, 1994]. The magnitude of oscillations increases as $C_a = dS_a/dP_c$ approaches zero (i.e. as S_a approaches S_{ra}), and the effect of the oscillations can be to severely limit computational efficiency and numerical accuracy. Therefore, mass lumping is frequently recommended to eliminate oscillatory behavior [Milly, 1985; Celia *et al.*, 1990;], at the expense of some loss in numerical accuracy [Huyakorn and Pinder, 1983; Zienkiewicz and Taylor, 1991].

Global mass balance errors at varying convergence tolerances are listed in Table 4.1. Small mass balance errors further confirm the accuracy of the numerical solutions. Mass balance errors increase slightly with increasing convergence tolerance.

No grid effects were observed in that all solutions were identical along nodes in the horizontal plane. Furthermore, identical solutions were obtained when the vertical direction was numerically reversed (i.e. the grid was rotated by 90 degrees and gravity components were set to $g_z=0$ and $g_x=9.81 \text{ m/s}^2$). Collectively these results indicate MISER is correctly solving the aqueous phase mass balance equation in cartesian coordinates.

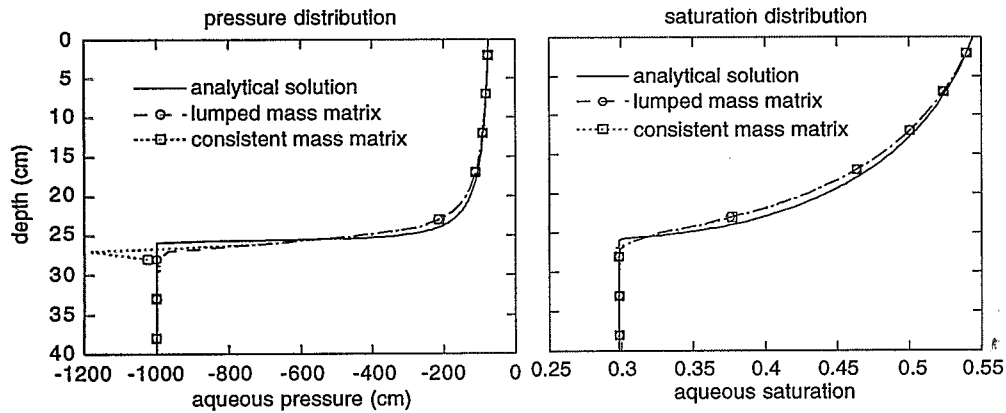


Figure 4.2: Comparison of numerical and analytical solutions for the one dimensional Richards equation. Simulation time = 6 hours; convergence tolerance = 1×10^{-4} .

convergence tolerance	mass matrix	residual ϵ_a (kg)	B_{1a}^t (%)	B_{2a}^t (%)
1.0×10^{-2}	consistent	2.66×10^{-4}	3.00×10^{-3}	5.77×10^{-2}
1.0×10^{-2}	lumped	1.76×10^{-4}	1.98×10^{-3}	3.81×10^{-2}
1.0×10^{-3}	consistent	-4.64×10^{-5}	5.21×10^{-4}	1.00×10^{-2}
1.0×10^{-3}	lumped	-4.58×10^{-6}	5.14×10^{-5}	9.93×10^{-4}
1.0×10^{-4}	consistent	5.35×10^{-7}	6.02×10^{-6}	1.16×10^{-4}
1.0×10^{-4}	lumped	-5.70×10^{-6}	6.41×10^{-5}	1.23×10^{-3}

Table 4.1: Comparison of global mass balance errors from numerical solutions of the one dimensional Richards equation at time 6 hrs.

4.2.2 Comparison with Two Dimensional Richards Equation

The capability of MISER to simulate a two dimensional variably saturated axisymmetric flow was tested by comparison to numerical solutions of the two dimensional Richards equation obtained from the SWMS_2D model [Simunek *et al.*, 1994]. The scenario under consideration is described as example problem 4 [Section 7.4, Simunek *et al.*, 1994] and involves moisture infiltration from a single-ring infiltrometer. The axisymmetric domain is shown in Figure 4.3 and the associated soil properties are listed in Table 4.2. The radius of the ring infiltrometer is 20 cm.

SWMS_2D numerically solves the two dimensional Richards equation using the Galerkin-type linear finite element scheme. For comparisons presented herein, SWMS_2D was run using quadrilateral elements. The grid, shown in Figure 4.4, consists of 342 elements and 380 nodes; it is identical to that described in the users manual [Section 7.4, Simunek *et al.*, 1994]. No flow conditions were prescribed along all boundaries, except at the five nodes along the top left boundary where constant pressure conditions were

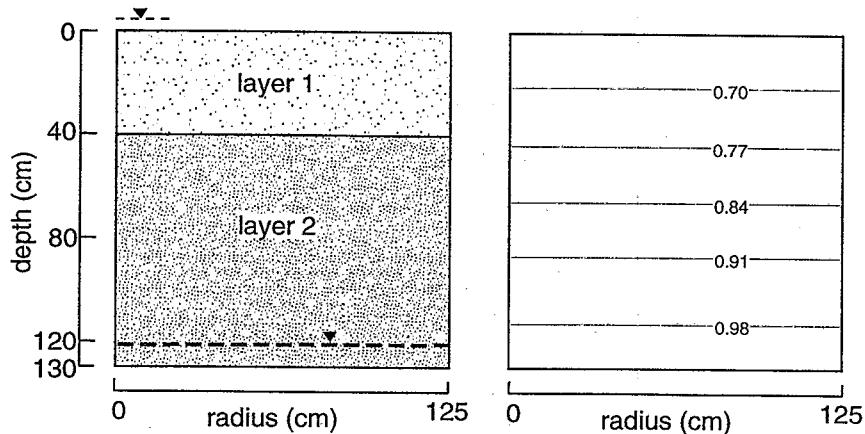


Figure 4.3: Domain configuration used in two dimensional (2D) flow simulations for comparison to SWMS_2D.

Parameter	Layer 1	Layer 2
ϕ	0.399	0.339
$k_x = k_z$ (m ²)	3.9598×10^{-13}	6.0327×10^{-13}
S_{rw}	2.51×10^{-4}	2.95×10^{-4}
n	1.376	1.603
α (1/Pa)	1.77×10^{-4}	1.42×10^{-4}

Table 4.2: Soil properties used in two dimensional flow simulations for comparison to SWMS_2D.

specified ($P_a = 1$ atm; $S_w = 1$).

The two dimensional moisture infiltration problem was simulated with MISER using the same nodal structure as in Figure 4.4. Each quadrilateral element was subdivided into two triangular elements resulting in 684 elements and 380 nodes. Boundary conditions for the gas phase were specified as first type, constant atmospheric gas pressure along the entire top boundary and no flow conditions on remaining boundaries. Other specifications required to conform to assumptions inherent in Richards equation were identical to those used in the one-dimensional comparisons: the transport equations were not solved; the gas phase composition was fixed; and the vapor viscosity was set to an artificially small value to eliminate effects of gas pressure transients on aqueous migration.

Intermodel comparisons were initially made for a homogeneous problem, employing the soil properties for layer 1 over the entire domain. Very close agreement was obtained in numerical predictions of moisture content, as demonstrated by near indistinguishable contour lines shown in Figure 4.5. Mass balance computations from MISER were on the order of 10^{-6} and $10^{-3}\%$ for the relative and absolute error measures, respectively, at a convergence tolerance of 10^{-3} . Collectively, these results indicate MISER is correctly solving the aqueous phase flow equation in a two dimensional axisymmetric domain.

The second comparison was for the layered domain shown in Figure 4.3. Results shown in Figures 4.6 and 4.7 show very close agreement in the upper layer and moderate discrepancies near the region of the soil interface (depth = 40 cm). These discrepancies are attributed to differences in the way that discontinuities

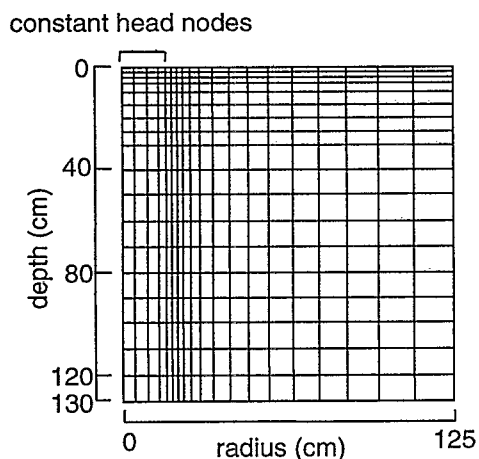


Figure 4.4: Numerical grid used in two dimensional flow simulations for comparison to SWMS_2D.

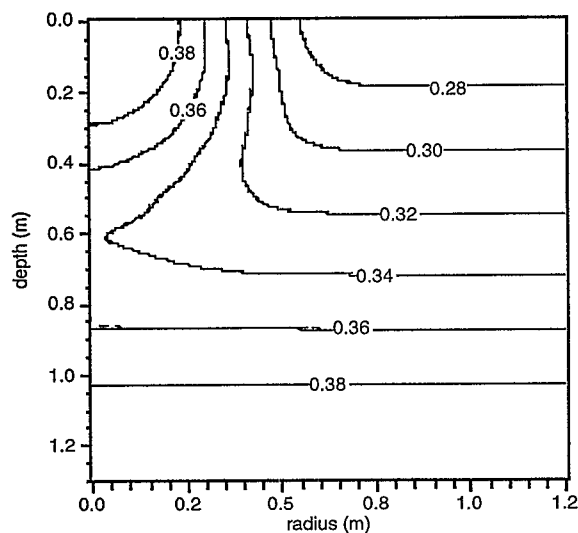


Figure 4.5: Simulated volumetric moisture content in the homogeneous domain at time 12 hrs (MISER = solid line; SWMS_2D = dashed line).

at the material property interface are treated in the two numerical models. In MISER, discontinuities are preserved numerically by tracking separate nodal material property parameters and saturation values within contiguous elements spanning the interface. Moisture profiles from MISER exhibit a sharp contrast at the interface (depth = 40 cm). In SWMS_2D, the nodal material property parameters are averaged. Moisture profiles from SWMS_2D exhibit a sharp contrast at depth of approximately 45 cm, or approximately the distance of one-half element below interface. Despite discrepancies near the interface, moisture profiles in Figure 4.7 show good agreement at depth below the interface.

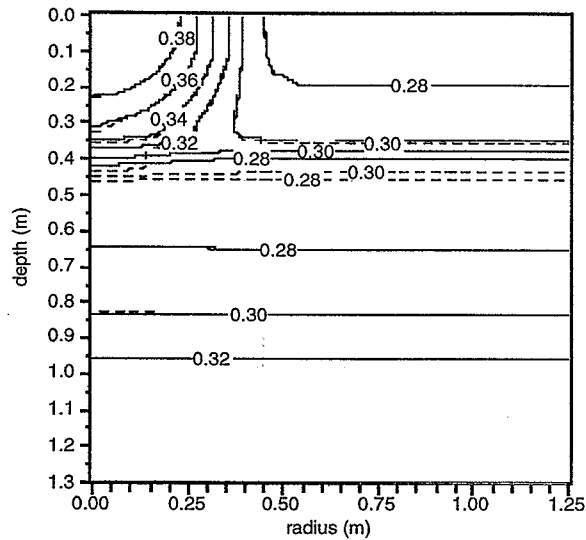


Figure 4.6: Simulated volumetric moisture content in the layered domain at time 6 hrs (MISER = solid line; SWMS_2D = dashed line).

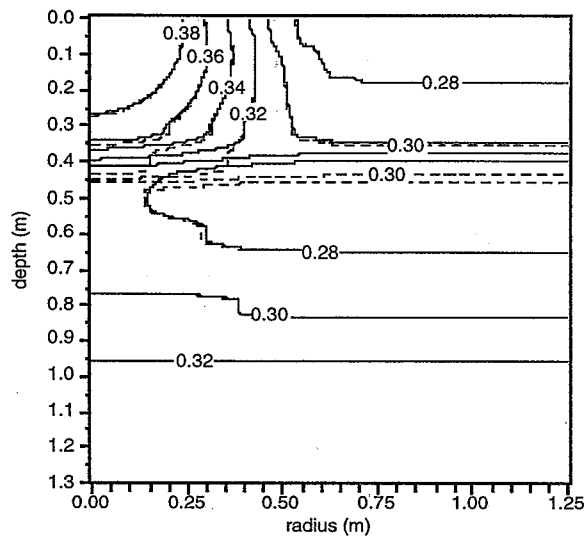


Figure 4.7: Simulated volumetric moisture content in the layered domain at time 12 hrs (MISER = solid line; SWMS_2D = dashed line).

4.2.3 Comparison with Quasi Analytical Solutions for Unsteady Radial Flow of Gas

The capability of MISER to simulate axisymmetric flow of gas to vadose zone extraction or injection wells was tested by comparison to quasi analytical solutions developed by *McWhorter* [1994]. The quasi analytical solutions represent unsteady one dimensional radial gas flow and account for nonlinearities stemming from pressure dependent density (compressibility) and permeability (Klinkenberg effect). To conform with conditions of the analytical solution, MISER was used to simulate confined radial gas flow to

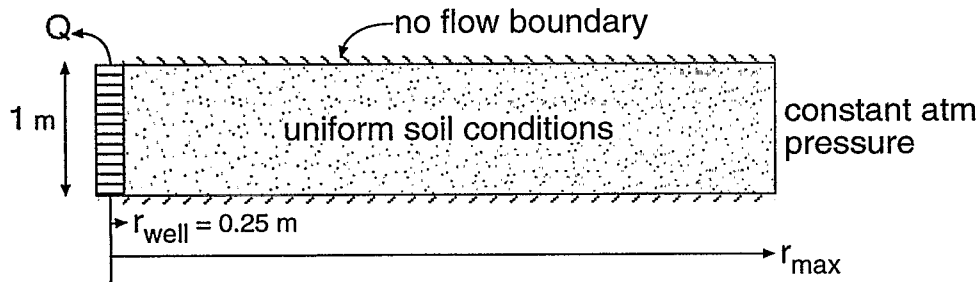


Figure 4.8: Domain configuration used in two dimensional flow simulations for comparison to quasi analytical solutions for radial gas flow.

Parameter	Case 1	Case 2
ϕ	0.33	0.33
$k_x = k_z$ (m ²)	1×10^{-11}	1×10^{-14}
S_{rw}	0.12	0.12
n	7.0	7.0
α (1/Pa)	0.002	0.002
b (atm)	0.0	0.316

Table 4.3: Soil properties used in two dimensional flow simulations for comparison to quasi-analytical solutions for radial gas flow.

a fully penetrating extraction/injection well (Figure 4.8). All soil properties were homogeneous (Table 4.3), with water present at the immobile residual level. Properties of nitrogen gas were used to represent the gas phase in both the numerical and analytical solutions.

Comparisons were made for varying conditions of gas injection and extraction using two values of intrinsic permeability. The first case involves a relatively conductive soil under conditions in which slip flow phenomena are negligible. For this problem the domain was discretized into 505 nodes (5 vertical by 101 horizontal) and 800 elements. Nodal spacing was uniform in the vertical direction and nonuniform in the radial direction, evaluating nodal coordinates from [Aziz and Settari, 1979],

$$\frac{r_{i+1}}{r_i} = \left(\frac{r_{\max}}{r_{\text{well}}} \right)^{1/(N-1)} \quad (4.12)$$

where $n = 101$ is the number of nodes in the radial direction, and $r_{\max} = 1000$ m is the radial coordinate of the right boundary. Comparisons of predicted pressure distributions are shown in Figure 4.9 for a extraction rate of 10 scfm and an injection rate of 1 scfm. Close agreement between MISER and the quasi analytical solutions is observed, indicating MISER is correctly solving the axisymmetric gas phase flow equation with specified well conditions. Figure 4.9 also shows an extensive radius of influence, indicating that accurate representation of the flow field requires a large grid structure under conditions of confined gas flow in a conductive soil.

The ability to simulate slip flow phenomena based on the Klinkenberg correction factor (2.4) was examined in a second problem involving extraction from a less conductive soil. For this problem the domain was discretized into 1005 nodes (5 vertical by 201 horizontal) and 1600 elements, using a uniform

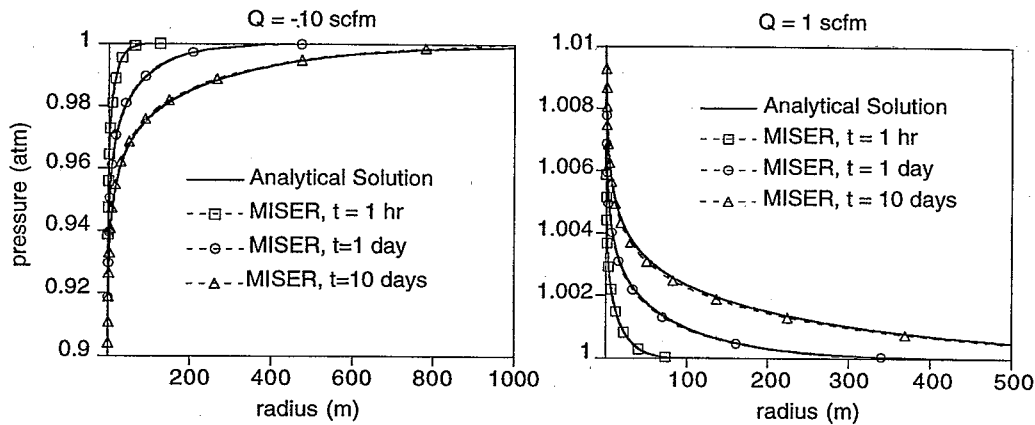


Figure 4.9: Comparison of quasi analytical and numerical solutions for two dimensional radial gas flow in a uniform soil with $k = 1 \times 10^{-11} \text{ m}^2$.

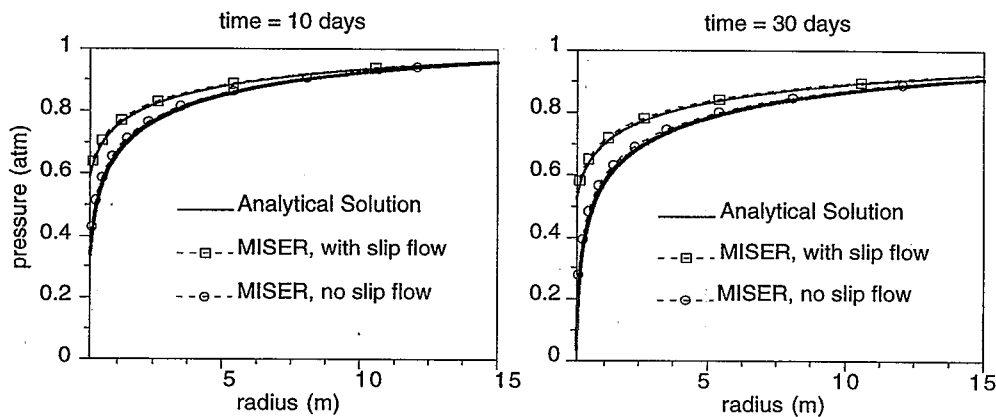


Figure 4.10: Comparison of quasi analytical and numerical solutions for one dimensional radial gas flow in a uniform soil with $k = 1 \times 10^{-14} \text{ m}^2$.

nodal spacing in the vertical direction and (4.12) to compute radial coordinates with $r_{\max} = 200 \text{ m}$. The Klinkenberg parameter (b) was evaluated from (2.5). Comparisons of predicted pressure distributions are shown in Figure 4.10 for a constant extraction rate of 0.08 scfm . Close agreement between MISER and the quasi-analytical solutions is observed, indicating MISER is correctly accounting for slip flow phenomena using the Klinkenberg correction factor.

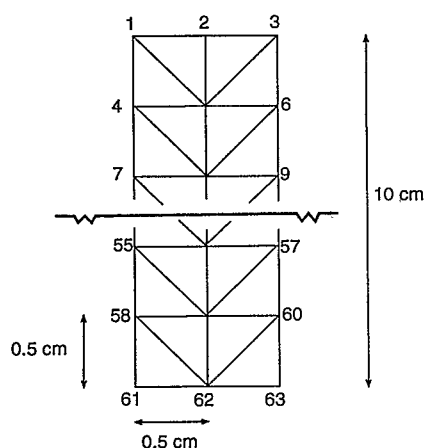


Figure 4.11: Computational grid used for the numerical solution of the one dimensional transport equation with and without advection.

4.3 VERIFICATION OF THE COMPONENT MASS BALANCE SOLUTIONS

4.3.1 Comparison with One Dimensional Analytical Solutions

The first set of model verifications for the transport section of MISER are comparisons with analytical solutions of the one dimensional transport equation with and without advection. In all the simulations discussed in this section, the flow portion of MISER was not operating. Additionally, the component parameters were normalized for the comparisons with analytical solutions and $S_\alpha = 1$. These comparisons were used to verify the ability of MISER to simultaneously simulate transport of multiple components in two mobile phases. The first analytical solution was of nonadvective transport with constant dispersion coefficients. The second analytical solution included constant advection [Ogata and Banks, 1961]. Figure 4.11 shows the "herringbone" grid with 63 nodes and 80 elements used for both of these comparisons. Minor grid effects were observed in that the nodal concentrations were not identical across the axis perpendicular to the direction of flow. Similar simulations performed using the "union jack" grid did not show these effects. Both vertical and horizontal orientations were tested for both mobile phases and for two components in each phase. In all cases, $\Delta x = 0.5$ cm, $\Delta z = 0.5$ cm, $\Delta t = 5$ sec, $D = 0.0003$ cm² sec⁻¹, and the convergence criteria was 10^{-8} . The component boundary conditions for the analytic solutions were first type with a value of $C = 1.0$ at $x = 0$ cm and second type with a zero solute gradient at $x = \infty$. Initially the solute concentration was 0.0 throughout the domain. For the numerical solutions, the domain was large enough so that the second type boundary condition was not violated. For the comparison with the Ogata and Banks solution, $v = 0.0002$ cm sec⁻¹. As Figures 4.12 and 4.13 show, MISER and the analytic solutions closely match at $t = 4000$ sec. The numerical simulations shown are for oxygen in the aqueous phase, however good matches were obtained when an additional component was added or when similar simulations were performed in the aqueous phase.

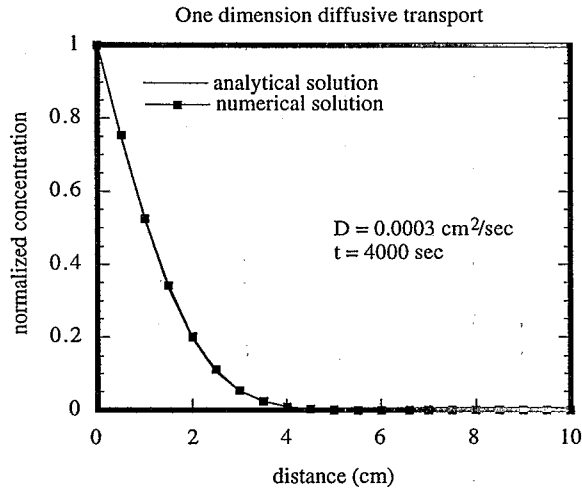


Figure 4.12: Comparison of MISER with the one dimensional analytical solution for diffusion driven transport.

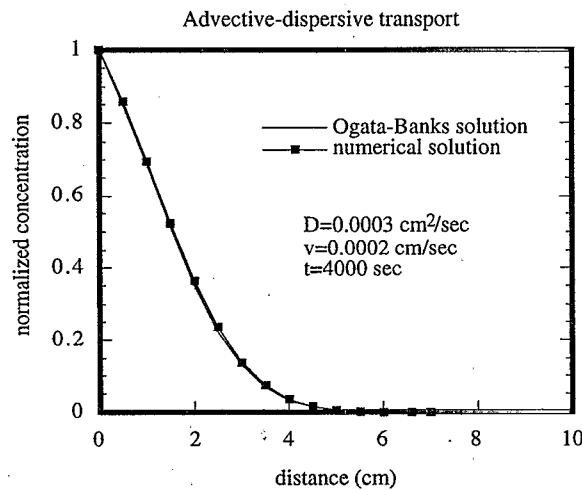


Figure 4.13: Comparison of MISER with the one dimensional Ogata and Banks analytical solution for transport with dispersion and constant advection.

4.3.2 Comparison with Two Dimensional Analytical Solutions

Numerical solutions obtained from MISER were next compared with solutions from a two dimensional analytical groundwater mass transport model [Cleary and Unga, 1978]. The analytical solution used allowed for dispersion and advection in both the x and z directions, as well as first order decay of the solute. These comparisons demonstrate the ability of MISER to correctly solve the transport equation in two dimensions. A "herringbone" grid was used for these simulations with 441 nodes and 800 elements. The

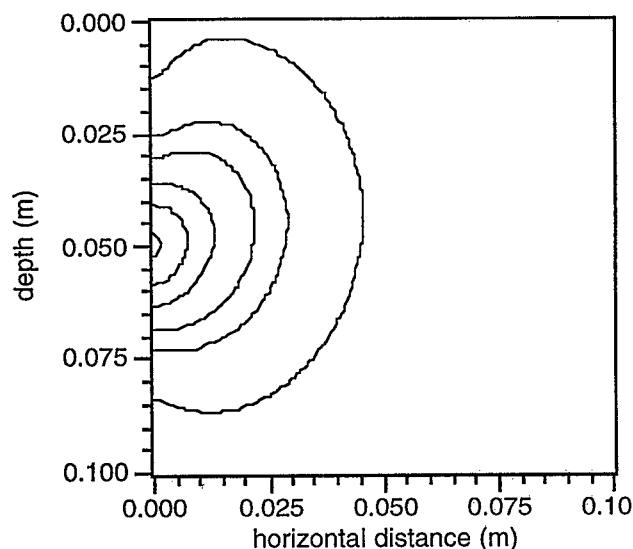


Figure 4.14: Comparison of MISER (solid lines) with a two dimensional analytical transport solution (dashed lines) at 4000 sec. Contours of normalized concentration are from left to right: 0.8, 0.6, 0.4, 0.2, 0.1, and 0.01.

grid was square with 21 nodes on each side. In all cases, $\Delta x = \Delta z = 5.0$ cm, $\Delta t = 5$ sec, $D_x = D_z = 0.0003$ cm² sec⁻¹, $v_x = v_z = 0.0002$ cm sec⁻¹ (where v_z is positive upwards), and the convergence criteria was 10^{-8} . The solute boundary condition for both the analytic and numeric solutions was first type with a Gaussian distribution along $x = 0$ cm, centered at $z = 5$ cm with $\sigma = 1.0$ at $z = 5$ cm and a standard deviation of 1.0. At the other three boundaries the solute concentration gradient was assumed to be zero (second type boundary) at ∞ for the analytic solution. For the numerical solutions, the domain was made large enough so that the second type boundary condition was not violated. The initial solute concentration was 0.0 throughout the domain at $t = 0.0$ for both the analytical and numerical solutions. The close agreement of the analytical and numerical solutions at $t = 4000$ sec in Figure 4.14 verifies the ability of MISER to correctly solve the transport equation in two dimensions.

4.3.3 Verification of Biokinetics

The ability of MISER to simulate Monod-type biological growth and decay was verified in two ways. First comparisons were made with the two dimensional analytical solution discussed in Section 4.3.2. The domain, parameters, boundary conditions, and initial conditions for the analytical solution were the same with the addition of first order solute decay of $k = 0.001$ sec⁻¹. MISER was then used to simulate first order decay by setting the half saturation constant of the degradable solute equal to 10^{32} and the maximum solute use rate equal to 10^{29} resulting in a first order decay rate of 0.001 sec⁻¹ as in the analytic solution. Otherwise, the grid and parameters were the same as for the previous simulation. MISER required that oxygen be present in the aqueous phase in order for the biodegradation routines to function properly, but by setting the oxygen use coefficient equal to zero, any effects of oxygen limitation were eliminated from this simulation. MISER also allows the biomass concentration to remain constant for the course of a simulation eliminating any effect of biomass growth. Both oxygen and biomass were set to an initial concentration of

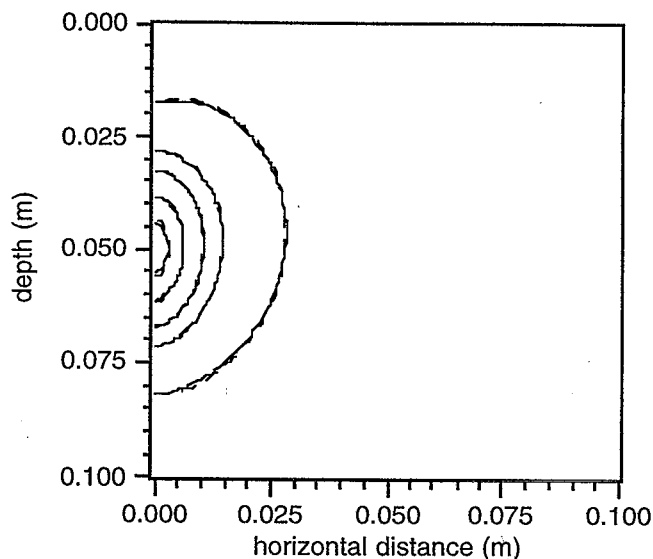


Figure 4.15: Comparison of MISER (solid lines) with a 2D analytical transport solution including first order decay (dashed lines) at 4000 sec. Contours of normalized concentration are from left to right: 0.8, 0.6, 0.4, 0.2, 0.1, and 0.01.

1.0 throughout the domain with first type boundary conditions for oxygen at $x = 0.0$ cm of 1.0. By using these values for oxygen and biomass, any dependency of the solution from MISER on these terms was eliminated. For this simulation a separate biophase was not considered and the reaction terms were directly inserted into the aqueous phase transport equations. The close agreement of the analytical and numerical solutions at $t = 4000$ sec shown in Figure 4.15 indicates that the Monod terms are being calculated and inserted into the appropriate transport equation correctly.

A comparison was also made to the model developed by *Moltz et al.*, [1986] for simulating microbial growth-degradation processes in porous media where the microorganisms are primarily present in microcolonies. Modifications were made to MISER to account for the significant differences in the way *Moltz, et al.*, [1986] handled oxygen usage. Their model also assumed that the transport of substrate and electron acceptor from the pore fluid to the microcolonies may be dominated by an adjacent diffusion layer resistance. This concept can be incorporated in MISER by the inclusion of a separate biophase with rate limited mass transfer. However, for this comparison MISER was operated with the bioreaction terms inserted directly into the aqueous phase transport equation (i.e., no mass transfer resistance) due to the differences in the way mass transfer resistance is handled in the two models. The comparison presented in Figure 4.16 is with Figure 8 from *Moltz et al.*, [1986] and the interested reader is referred to their paper for all pertinent biokinetic and media parameters. A "herringbone" grid similar to Figure 4.11 was used for these simulations with 303 nodes and 400 elements. The grid was quasi one dimensional with 3 nodes in one dimension and 101 nodes in the other dimension. In all cases, $\Delta x = \Delta z = 1.0$ cm, the maximum $\Delta t = 0.1$ day, and the convergence criteria was 10^{-8} . Given the differences between the two models the comparison at 4 days presented in Figure 4.16 is reasonable and indicates that MISER is incorporating nonlinear Monod-type biokinetics into the bioreaction terms.

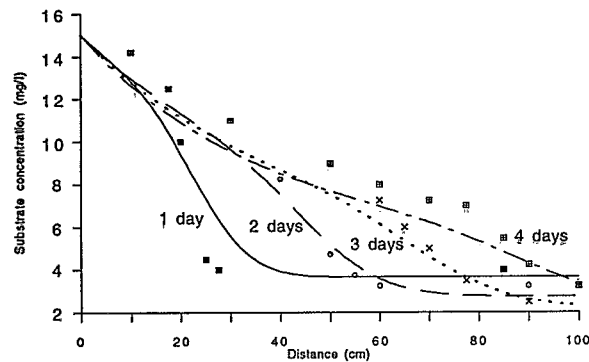


Figure 4.16: Comparison of substrate profiles along a one dimensional column by MISER (lines) and a one dimensional numerical solution (discrete points) for biodegradation by *Moltz et al.*, [1986].

4.3.4 Verification of Interphase Exchange

Next the ability of MISER to simulate linear equilibrium sorption was tested. Comparisons were made with the two dimensional analytical solution discussed in Section 4.3.2. The coefficients of the analytical solution were modified to incorporate a solute retardation factor of 2.0. This was done by setting $D_x = D_z = 0.00015 \text{ cm}^2 \text{ sec}^{-1}$ and $v_x = v_z = 0.0001 \text{ cm sec}^{-1}$. All the other parameters for the analytical solution remained the same. For MISER, the parameters and domain configuration discussed in Section 4.3.2 were used with the addition of the sorption parameters $K_f = 1.0$ and $m = 1.0$. The mass exchange coefficient was set to 0.01 sec^{-1} which was large enough to simulate equilibrium sorption processes using the nonequilibrium formulation. The other parameters remained the same as in the previous simulations. As can be seen in Figure 4.17, good comparisons were obtained between MISER and the analytical solution. The slight differences between the two solutions at the 0.01 contour is consistent with a slight deviation from equilibrium for MISER. This comparison demonstrates the capability of MISER to simulate equilibrium exchange processes between mobile and immobile phases. When a retardation factor of 2.0 was incorporated directly into MISER, the match with the analytical solution was nearly exact (not shown).

Finally, MISER was compared with a one dimensional multicomponent organic liquid volatilization column experiment by *Bloes et al.*, [1989]. In this experiment benzene, TCE, and toluene were vented from a column of glass beads by passing dry nitrogen gas through the column at a constant flow rate. No aqueous phase was present and the NAPL was at an immobile residual saturation. A analysis of the experimental results revealed that at the experimental flow rate, the mole fractions of the organic compounds in the gas phase were approximately at equilibrium with the organic liquid for the duration of the experiment. For this comparison, MISER was operated with only the gas and organic liquid phases present. All the pertinent parameters are available in *Bloes et al.*, [1989] and *Rathfelder et al.*, [1991]. A "union jack" grid similar to Figure 4.9 was used for these simulations with 271 nodes and 480 elements. The grid was quasi one dimensional with 31 nodes in the x direction and 5 in the z direction. $\Delta x = \Delta z = 1.0 \text{ cm}$, $\Delta t = 600 \text{ sec}$, $v_x = 0.108745 \text{ cm sec}^{-1}$, and the convergence criteria was 10^{-8} . The mass transfer coefficients were set to the arbitrarily large value of 500 sec^{-1} and eq. (3.42) was used to limit the mass transfer coefficients to a value sufficient to approximate equilibrium conditions. Figure 4.18 indicates that MISER can simulate equilibrium multicomponent organic liquid volatilization validating the representation of interphase exchange between a mobile and an immobile phase, this time for three components.

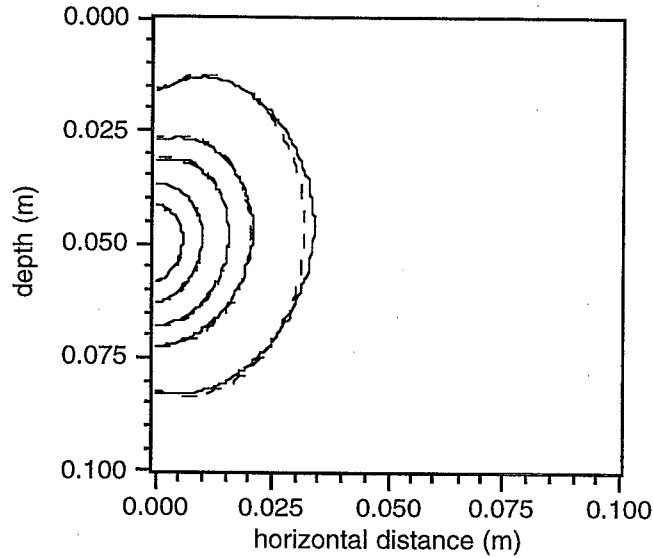


Figure 4.17: Comparison of MISER (solid lines) with a two dimensional analytical solution including linear equilibrium sorption (dashed lines) at 4000 sec. Contours of normalized concentration are from left to right: 0.8, 0.6, 0.4, 0.2, 0.1, and 0.01.

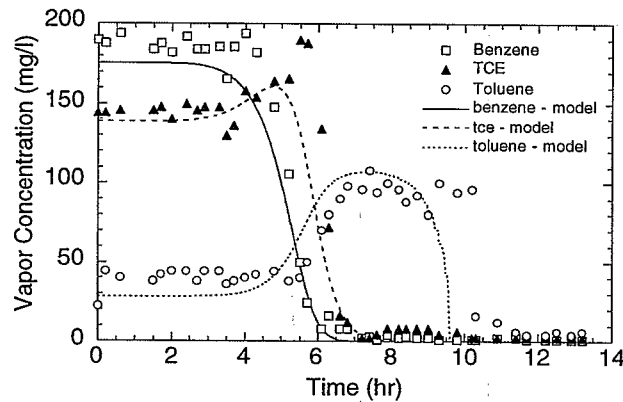


Figure 4.18: Comparison of MISER (lines) with a one dimensional column experiment (discrete points) for multicomponent organic liquid (benzene, TCE, toluene) volatilization under equilibrium conditions.

4.4 VERIFICATION OF THE COUPLED PHASE AND COMPONENT MASS BALANCE SOLUTIONS

In order for the coupled flow and transport portions of MISER to represent transport processes, the phase pressures must be correctly translated into specific fluxes. Verification of this procedure was done by using the domain from Section 4.3.1 and running the coupled flow and transport sections of MISER with first type pressure boundary conditions of 1.001 atm at $x = 0.0$ cm and 1.0 atm at $x = 10.0$ cm. Otherwise the

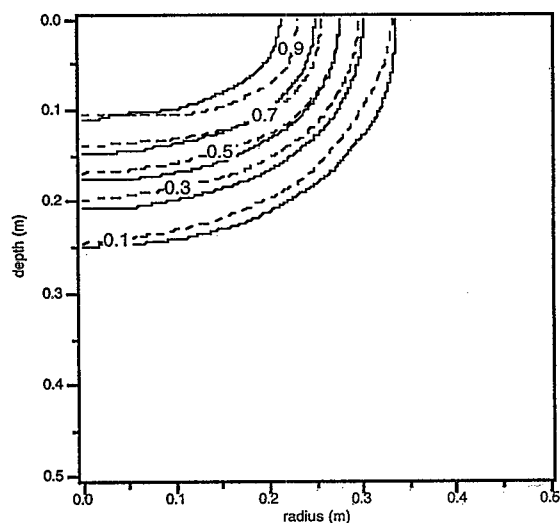


Figure 4.19: Comparison of predicted solute concentrations at time 6 hrs (MISER = solid line; SWMS_2D = dashed line).

parameter set remained the same as in Section 4.3.1. The specific fluxes generated by MISER were nearly identical to specific fluxes calculated using,

$$q_{\alpha_x} = -\lambda_{\alpha_x} \frac{\Delta P_{\alpha}}{\Delta x} \quad (4.13)$$

This was done in both directions.

Tests on the coupled flow equations and transport were also conducted by intermodel comparison with SWMS_2D model [Simunek *et al.*, 1994]. The flow domain was identical to that described in Section 4.2.2. Water introduced through the ring-infiltrometer contains a solute at the solubility limit. This condition was simulated with type I (constant concentration) boundary conditions on the aqueous solute. No sorption was considered in these comparisons. Additional transport parameters are given in Section 7.4 of Simunek *et al.*, [1994].

Figures 4.19 and 4.20 compare predicted solute distributions from MISER and SWMS_2D. Similar results were obtained with MISER using either element average or nodal velocity computations. Results show that MISER predicts a slightly more disperse solute front away from the boundaries. There is, however, relatively good agreement in the location of the center of mass of the solute front ($C=0.5$), and the overall agreement is considered reasonably good, given the differences in the type of elements and material property discretization used by the two models.

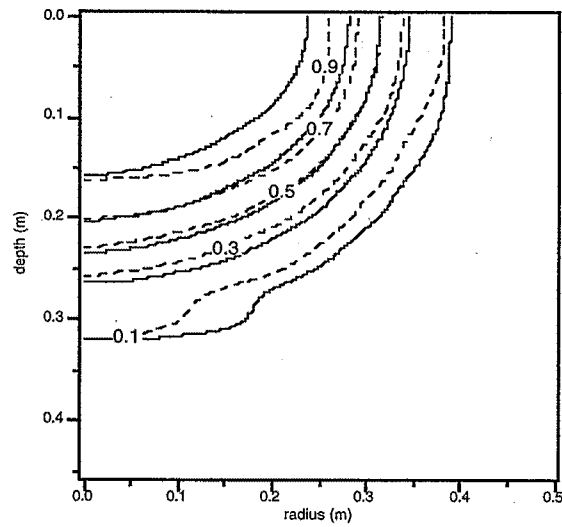


Figure 4.20: Comparison of predicted solute concentrations at time 12 hrs (MISER = solid line; SWMS_2D = dashed line).

Section 5

PROGRAM DESCRIPTION AND SIMULATION SETUP

5.1 CODE DESCRIPTION

MISER is structured in a top down modular format for clarity, to aid in logic tracing, and to simplify code modification. The modular format also enables the code to be easily programmed to run in different modes such as: (1) flow solutions only; (2) transport solutions only using a steady state flow field; (3) transport and biodegradation solutions using a steady state flow field; and (4) full coupling of the transient flow, transport, and biodegradation equations. The code is liberally annotated with comment lines.

MISER is developed in the FORTRAN 77 programming language with a few standard enhancements such as 'include' and 'Do-EndDo' statements. Due to the large number of equations being solved and the complexity of the solution algorithm, the code is intended to be used on work station platforms or main frame computers. The code has been tested with, and should normally be run using double-precision real variables.

5.2 PROGRAM MODULES

MISER is comprised of 25 program modules: 1 main program, 23 subroutine files, and 1 include file. Table 5.1 provides a short description of the program modules.

To implement MISER, all program modules must be compiled and linked into an executable code. A sample make file to compile MISER on an IBM RS6000 workstation is given in Appendix F.

5.3 ARRAY DIMENSIONS AND PROGRAM VARIABLES

Named common blocks are used to dimension all global arrays and to pass information between program modules. The variable dimensions of all global arrays are defined with parameter statements in a single 'include' file (dimen.inc). Most array dimensions are calculated from a combination of only a few parameter variables, such as: the maximum number of nodes; maximum number of elements; maximum number of components; and maximum number of material property blocks. In this way the dimensions of most array variables can be easily adjusted by modifying only a few parameter variables in a single program unit. These parameters should be set greater than or equal to the conditions of the problem to be simulated. The major user defined parameter variables in 'dimen.inc' are listed in Table 5.2. Other parameters in 'dimen.inc' include physical constants, constants related to numerical performance, and control variables. The entire code must be compiled each time changes are made in 'dimen.inc'.

A description of major program variables is given in Appendix E

5.4 DESCRIPTION OF INPUT FILES AND INPUT DATA

Table 5.3 describes all input and output files that can potentially be used in a single simulation run. A given simulation may not generate every listed output file depending upon user specifications. Only those files that are required on the basis of user specifications are opened.

The user supplied input data is contained in the first two input files. The first input file is named 'miser.d1'. 'Miser.d1' must be located in the same directory as the executable code. The name and path of the second input file (designated as D2 in this documentation) is defined in data block A of 'miser.d1'. Similarly, a prefix name and path for all output files is defined in data block A of 'miser.d1'.

The input data in 'miser.d1' and D2 are organized into blocks identified by letters A-T. File 'miser.d1' contains data blocks A-L, and file D2 contains data blocks M-T. Depending on the problem conditions, all data blocks may not be required for each simulation. The data blocks are further subdivided into fields that are separated by comment lines designated by a '#' in column 1. An indefinite number of comment lines can be added between fields, however, there must be at least one. The input data are read using list directed formatting (free format).

The following sections provide a description of all input data and the organization of the data blocks. Examples of input data files are shown in Appendix G.

5.4.1 Data Block A – Input/Output Files and Control Options

Input data in block A defines the names and paths of all input and output files used in a particular simulation. Several optional output files may be generated (Table 5.3) with specified control switches defined in block A. Table 5.4 describes all input data required in block A.

5.4.2 Data Block B – General Model Control Options

Data block B contains a number of control switches used to specify general model options, such as: the type of computational domain; the inclusion or exclusion of flow, transport, and biotransformation processes; and numerical solution control parameters. Table 5.5 describes all data contained in block B.

5.4.3 Data Block C – Time Step and Iteration Control Parameters

Data block C contains parameters that affect time step control, including parameters used for empirical time step adjustment described in Section 3.10. Table 5.6 describes all data contained in block C.

5.4.4 Data Block D – Grid Information and Control Options

Data block D contains the grid and element information. Input data in this block are described in Table 5.7.

MISER employs triangular elements to discretize the solution domain. The nodal incidence list for each element begins at an arbitrary node and proceeds counterclockwise if the vertical coordinate is positive downwards, or clockwise if the vertical coordinate is positive upwards. The minimum material property block size is a quadrilateral containing two elements in the case of a herringbone grid or four elements in the case of a union jack grid (i.e. a single triangular element cannot be designated with unique material property information).

User options are provided to either input all nodal coordinates and elements indices, or generate a regular grid on a rectangular solution domain. Two types of grids can be generated: a symmetric union jack grid (e.g. Figure 4.1), or a herring bone grid (e.g. Figure 4.11). Material properties in a generated grid are restricted to horizontal alignments (soil layers).

5.4.5 Data Block E – Component Chemical Properties

Data block E defines the chemical property information for all organic constituents, water, oxygen, nitrogen, and an optional single limiting nutrient. The ordering of components is important. The organic components are input first. Subsequently the water, oxygen, nitrogen are defined followed by the nutrient if present. Each component is identified with a component number starting with 1 for the first organic component and proceeding in the order that they are input. The organic components, oxygen and nutrient can be restricted from partitioning into the gas or aqueous phases by specifying negative values for the vapor pressure and solubility, respectively. Table 5.8 describes all data in block E. Input values required in this data block are widely available [c.f. *Dean* (Ed.), 1985].

5.4.6 Data Block F – Mass Transfer Coefficients

Data block F defines the lumped mass transfer coefficients for all the components. The organic liquid mass transfer coefficients are input first on a line followed by the minimum allowed deviation (See equation (3.42) from equilibrium for the organic liquid components on a second line. These two lines are repeated for water, oxygen, nitrogen, and nutrient if present. Note that exchange coefficients for nitrogen are entered even though nitrogen does not exchange between phases. Table 5.9 describes all data in block F. The user is referred to several recent studies which have developed correlations for mass transfer coefficients as an initial point for setting values appropriate for the desired scenarios [c.f. *Wilkins et al.*, 1995; *Powers et al.*, 1991, 1992, 1994; *Cho and Jaffe*, 1990]

5.4.7 Data Block G – Material Property Block Information

Soil property information and dispersion parameters are defined in data block G, and described in Table 5.10. Input values required in this data block are widely available [c.f. *Freeze and Cherry*, 1979].

5.4.8 Data Block H – Sorption Parameter Data

Sorption parameter data are specified in block H and are described in Table 5.11.

Sorption can be modeled as either a rate limited or equilibrium process. If rate limited sorption is modeled, then the solid phase transport equations must be solved as designated by the control switch `lctrl(25)` defined in block B, field 2. Under rate limited conditions, sorption can be modeled with either a single compartment or two compartment model. The two compartment model, however, is restricted to the case of a homogeneous soil domain and a single component organic liquid. Additionally, the aqueous-solid exchange coefficients must be nonzero for all organic liquid components. A positive value indicates sorption is modeled for that organic component. A negative value indicates no sorption is considered for that organic component and therefore the solid phase transport equation is not solved for the corresponding component.

Equilibrium sorption processes are modeled by the inclusion of retardation factors. Equilibrium sorption can only be considered in the absence of nonequilibrium sorptive processes. However, any component of the aqueous phase can be modeled with retardation. To implement the use of retardation factors the control switch `lctrl(25)` must be set to `.false.`, indicating that the solid phase transport equations are not solved. Sorption isotherm data is widely available in the literature [c.f. *Weber et al.*, 1988, 1991, 1992].

5.4.9 Data Block I – Biological Parameter Data

Biological parameter data are specified in block I and are described in Table 5.12. This data block is read only if the biotransformation equations are solved, as indicated by the control switch `lctrl(3)` defined in block B, field 2. Otherwise the entire block is omitted.

Five options are available to model biokinetics. Option 1 is standard Monod kinetics. Option 2 is Monod kinetics with substrate inhibition. Inhibition is modeled with hyperbolic functions that impede microbial metabolism when substrate or nutrient concentrations are greater than specified threshold limits. Inhibition can also be applied to the electron acceptor, in which case inhibition occurs when the oxygen concentration is below the specified threshold limit. Option 3 is Monod kinetics with lumped substrate inhibition. Under this option lumping only applies to the degradable substrates; nutrient and oxygen are handled as in option 2. Option 4 is Monod kinetics with saturation dependency. Option 5 is Monod kinetics with saturation dependency and substrate inhibition. Inhibition is handled as in option 2. The user is referred to several recent studies as a aid to determining the appropriate parameters required by this data block [c.f. *Sleep and Sykes*, 1991; *Chen et al.*, 1992; *Fuller et al.*, 1995; *Chen*, 1996].

5.4.10 Data Block J – Phase Parameter Data

Phase parameter data are specified in block J and are described in Table 5.13. On output, the composition of each phase is identified. Phase parameter data can be found in standard reference texts [c.f. *Dean* (Ed.), 1985].

5.4.11 Data Block K – Temperature Parameter Data

Temperature parameter data are specified in block K and are described in Table 5.14.

Steady state temperature distributions can be specified as either uniform or nonuniform with depth. The

latter option can be used only for rectangular domains, and requires input not only for the depth dependent temperature distribution, but additionally requires the temperature dependencies of the following 6 parameters: vapor pressure, vapor viscosity, Henry's Law constant, aqueous solubility, maximum substrate utilization rate, and biomass decay rate. Values for the first 5 temperature dependencies are required for all components present in a given simulation (Note: values may be required for components to which a given parameter does not apply, i.e. maximum substrate utilization rate for nitrogen. These values are ignored subsequent to the input section). Values for the temperature dependent parameters are input for each node along the vertical boundary. Intermediate values are linearly interpolated for the nodes at the center of "union jack" grids. The user is referred to *Dean* (Ed.) [1985] for a description of the temperature dependencies for the first 4 parameters input in this section. Biological parameter temperature dependencies are typically estimated with the van't Hoff- Arrhenius equation [c.f. *Atlas and Bartha*, 1987; *Chen et al.*, 1992].

5.4.12 Data Block L – Output Control Parameters

This data block contains control parameters for the three major output files: 'outpre.out', 'outpre.con', and 'outpre.plt', where 'outpre' is the user specified path and file name from data block A, field 2. The user can specify output variables for printing in the main output file, for printing in a contour plot format, or for printing as a time series output. A complete description of the input required in this data block is described in Table 5.15.

5.4.13 Data Block M – Restart Identifier

This data block contains two logical variables; (lctrl(26)) which is set to .true. if initial conditions (pressure, saturation, and phase composition) are to be read from the restart file and (lctrl(32)) which is set to .true. if the run is a continuation of a previous run and .false. if the run is a new run using the previous run as initial conditions. If (lctrl(26)) is .true. data block M also contains the path and name of the file containing the restart information.

File D2 is read regardless of whether the restart option is used. However, when the restart option is specified, initial conditions read from D2 are ignored and superseded by those read from the restart file. Boundary conditions for either case are read from D2.

5.4.14 Data Block N – Initial Pressure Conditions

Initial pressure conditions are specified in block N as described in Table 5.17. Initial pressure conditions at all nodes can either be input or computed. Computed pressures are assumed to be in hydrostatic equilibrium, referenced to atmospheric pressure at ground surface for the gas phase, and referenced to the gas phase pressure at the water table for the aqueous phase. The presence of organic components in the gas and aqueous phases is ignored in computing the hydrostatic pressure distributions.

5.4.15 Data Block O – Velocity Computation

This data block provides information indicating the method used to compute aqueous and gas phase velocity. Table 5.18 describes all data inputs in block O.

User options are provided to compute nodal velocities from the pressure distribution using a finite element solution of Darcy's Law. Alternatively element velocities may be computed directly from Darcy's Law using element averages of the nodal mobilities and densities.

Velocity distributions can be unsteady (flow equations are solved) or assumed to be at steady state. This is controlled by variable `lctrl(1)` in block B, field 1 (Table 5.5), which indicates if the flow equations are solved. The velocity field is assumed to be at steady state if flow equations are not solved. A steady state velocity field can either be input directly, or computed from an input pressure field.

5.4.16 Data Block P – Organic Liquid Saturation and Composition

Data block P contains the initial quantities for the immobile organic liquid saturation distribution and composition (component mole fractions). All organic liquid saturations and component mole fractions are input on an element basis and converted to nodal quantities by averaging adjacent elemental values. Averaging is not performed across boundaries between different material property blocks. This results in multiple values of NAPL saturation at boundary nodes between different material property blocks. Components of the organic liquid are partitioned at equilibrium into the gas, aqueous, solid, and biophases when present. The mass of the organic liquid is not conserved during this process (i.e. organic mass is generated). This is only done at nodes where organic liquid is present and can lead to sharp discontinuities in the composition of the contacting phases. The use of smooth initial conditions such as those resulting from a restart file generated by a diffusion driven problem is recommended. Table 5.19 describes all data inputs in block P.

5.4.17 Data Block Q – Oxygen and Nutrient Initial Conditions

Initial conditions for oxygen and nutrient are defined in data block Q and described in Table 5.20. Fields 1-3 are for the gas phase, and fields 4-6 are for the aqueous phase. The initial conditions of the biophase components are identical to those of the aqueous phase. Initial conditions are required only when oxygen is present in the phase; i.e. if oxygen is omitted from the gas phase and/or the aqueous phase, then no data is read for the corresponding phase (oxygen can be omitted from the gas and aqueous phases by assigning negative inputs to the oxygen vapor pressure and solubility, respectively, in block E). No data is read for nutrient if it is absent (`lctrl(9) = .false.`). The biodegradation equations must be solved (`lctrl(3) = .true.`) in order for nutrient to be present. When water is present in the gas phase the initial conditions are specified with a relative humidity of 100%.

5.4.18 Data Block R – Boundary Conditions

Boundary conditions for the flow and transport equations are defined in data block R and described in Table 5.21.

Boundary conditions on the flow field can be either constant pressure or constant flux (Section 3.11.1). Boundary conditions at nodes not explicitly specified in data block R are treated as second type with no flow across the boundary. Boundary conditions arising at an injection/extraction well discussed below are also implemented through flux source/sink terms (Section 3.11.3).

Boundary conditions for the gas and aqueous phase component transport equations include specified concentration, specified diffusive flux, or mixed third type conditions. Boundary conditions at nodes not explicitly specified in data block R are treated as second type with no concentration gradient across the boundary. No boundary information is required for the immobile phases. Sections 3.11.2 and 3.11.3 discuss the transport equation boundary and associated specifications with extraction/injection wells.

5.4.19 Data Block S – Extraction/Injection Well Conditions

An extraction/injection can be defined in an r - z rectangular domain. Data block S contains input data defining the well conditions (see Table 5.22). The well is positioned along the left vertical boundary; the nodal coordinates along the left boundary must be equivalent to the specified well radius. A constant injection/extraction rate is defined for the duration of the simulation (variable pumping rates require the use of the restart option). The well screen is defined by specifying the minimum and maximum node numbers along the well screen.

5.4.20 Data Block T – Velocity Boundary Conditions

Data block T contains input data defining the velocity boundary conditions (see Table 5.23). Boundary conditions for the gas and aqueous velocities must be specified when the velocities are calculated using the finite element method (`lctrl(18) = .true.`). When a boundary is specified as impervious, the velocities normal to that boundary are zero. The boundary specification is the same for both the gas and aqueous phases. The domain boundary is divided into 4 sections; top, bottom, left side, and right side. Each section may be entirely impervious. The top boundary may also be partially impervious (i.e. a partial cap may exist). The left boundary is adjusted for the presence of a well.

5.5 DESCRIPTION OF OUTPUT FILES

Table 5.3 describes all the output files that can potentially be used in a single simulation run. A given simulation may not generate every listed output file depending upon user specifications. Only those files that are required on the basis of user specifications are opened. The following sections provide a description of the output files.

5.5.1 Main Output File - 'Outpre.out'

The main output file, 'Outpre.out', is always generated and is written to device 21. 'Outpre' contains both the file name and path, and is specified in data block A, *Field 1*. 'Outpre.out' contains the input parameters and selected output variables. The user may elect not to print out the entire set of grid information (see Data

Block D, *Field 1*) and the initial conditions (see Data Block L, *Field 1*) in order to reduce output file size. The selected output parameters are printed at user specified intervals (see Data Block A, *Field 8*). The selected output variables are specified in Data Block L, *Field 2*. Selection of the individual components in a phase is done with the global component numbers used to specify the component properties in Data Block E.

5.5.2 Convergence History and Runtime Information Output File - 'Outpre.cnv'

The convergence history and runtime information output file is optional. This output can be directed to the screen by setting the $\text{ipt}(28) = 6$ (Data Block A, *Field 3*), to 'outpre.out' by setting $\text{ipt}(28) = 21$, to 'outpre.cnv' by setting $\text{ipt}(28) = 23$, or not generated by setting $\text{ipt}(28) = 0$. Convergence history and runtime information output consists of iteration information from the various routines, maximum element Peclet and Courant numbers for both the gas and aqueous phases ($\text{set lctcl}(4) = \text{.true.}$; Data Block B, *Field 7*), time step information, and error messages from the solver.

5.5.3 Error Message Output File - 'Outpre.err'

Error message output consists of messages from the input error message file, (specified in Data block A, *Field 2*) and relates primarily to error checking of input values. The error message output file is optional. This output can be directed to the screen by setting $\text{ipt}(29) = 6$ (Data Block A, *Field 3*), to 'outpre.out' by setting $\text{ipt}(29) = 21$, to 'outpre.err' by setting $\text{ipt}(29) = 22$, or not generated by setting $\text{ipt}(29) = 0$.

5.5.4 Mass Balance Output File - 'Outpre.mb'

The mass balance output file is optional and is generated when $\text{lprnt}(6) = \text{.true.}$ (Data Block A, *Field 5*). When generated, mass balance output is always written to the file, 'outpre.mb' (device 25). The mass balance output is printed at user specified intervals (see Data Block A, *Field 5*). Mass balance output is available in two forms, referred to as report form ($\text{lprnt}(27) = \text{.true.}$; Data Block A, *Field 5*), and in multiple files as time series form ($\text{lprnt}(27) = \text{.false.}$; Data Block A, *Field 5*).

The report form mass balance contains self explanatory headings and contains both phase and component mass balances. The boundary, reaction, and source fluxes are also reported, along with separate values for the boundary fluxes at the surface and at the extraction well. Three types of mass balance errors are also reported (see Section 4.1). A maximum of nine components are allowed when generating report form mass balance output.

The time series form mass balance also contains self explanatory headings at the beginning of the generated output file(s). 'Outpre.mb' only contains the phase mass balance information. Only the first type mass balance error is reported (see Section 4.1) and surface flux is not reported in 'outpre.mb'. An additional output file is generated for each of the components present. These additional files are named 'outpre.mb#' (device 28+#) where # is the global component number as defined in data block E. A maximum of ten components are allowed when generating time series form mass balance output. In the files, 'outpre.mb#', the surface flux is reported in place of the first type mass balance error. All time series mass balance output files are formatted: e11.5,11e11.4.

5.5.5 Contour Plot Output File - 'Outpre.con'

The contour plot output file is optional and is generated when `lprnt(23) = .true.` (Data Block A, *Field 4*). When generated, contour plot output is written to the file 'outpre.con' (device 26). The contour plot output is printed at the same intervals specified for the main output file (see Data Block A, *Field 8*). The selected contour variables are specified in Data Block L, *Field 2* and may be different than the output variables selected for the main output file, 'outpre.out'. Selection of the individual components in a phase is done with the global component numbers used to specify the component properties in Data Block E. The contour plot file contains self explanatory headings at the beginning of each group of contour variables. Each group of nodal contour variables has the nodal x (or r) location in the first column and the nodal z location in the second column. For element contour variables the corresponding locations are for the element centroids. All contour plot output files are formatted: 8e15.8.

5.5.6 Time Series Plot Output File - 'Outpre.plt'

The time series plot output file is optional and is generated when `lprnt(15) = .true.` (Data Block A, *Field 6*). When generated, time series plot output is always written to the file, 'outpre.plt' (device 27). The time series plot output is printed at user specified intervals (see Data Block A, *Field 6*). The selected time series plot variables are specified in Data Block L, *Field 3 (gas phase) and 4 (aqueous phase)*. Time series plotting is only available for components of the gas and aqueous phases. Specification of time series plot output requires both the global component numbers used to specify the component properties in Data Block E and a node number. A given component may be specified at several locations in a phase. A maximum of six components can be specified for the combined gas and aqueous phases. The time series plot file does not contain headings. All time series plot output files are formatted: 7e11.8. The first column contains the current simulation time in seconds, subsequent columns contain the component mole fractions at the specified locations in order of their appearance in Data Block L, *Field 3 and 4*.

5.5.7 Restart Output File - 'Outpre.rst'

The restart output file is optional and is generated when `lprnt(5) = .true.` (Data Block A, *Field 7*). When generated, the restart file is always written to the file, 'outpre.rst' (device 28). The restart file is printed at the same intervals specified for the main output file (see Data Block A, *Field 8*). At each print time the restart file is rewound and restart information is printed over restart output from the previous print time. Thus 'outpre.rst' contains only restart information corresponding to the latest output time. To use a restart file, rename 'outpre.rst' to the name specified in Data Block M, *Field 2*.

Table 5.1: MISER program modules.

Routine	Type	Description
atri.f	subroutine	Evaluates the area and radial centroid of all elements. Performs minor error checking on grid geometry.
bcflux.f	subroutine	Computes gas and aqueous phase fluxes at prescribed pressure and source nodes.
bio.f	subroutine	Computes the biological reaction terms using Monod kinetics. Solves the biophase transport equations when a separate biophase is considered.
cbal.f	subroutine	Computes global and time step mass balance errors for the aqueous and gas phases, and for all components in all phases.
commnt.f	subroutine	Determines comment lines in the input data files and positions the file pointer to the next input data field.
dimen.inc	include file	Included in most MISER routines, this program unit is used to define parameter variables for array dimensions.
disper.f	subroutine	Computes the phase dependent portion of the tortuosity coefficient and the dispersion tensor.
error.f	subroutine	Reads and writes error message from the error message file. Terminates execution if the error is designated as fatal.
flow.f	subroutine	Solves the mobile aqueous and gas phase mass balance equations using the simultaneously solution method for a single time step.
grid.f	subroutine	Generates a union jack or herring bone grid for a rectangular solution domain.
har.f	subroutine	Contains the subroutines comprising the Harwell sparse matrix package for linear system solutions.
input1.f	subroutine	Reads the input and output file names and opens appropriate file units. Reads input data for: model control options; time step and iteration control information; grid information; component chemical properties; mass exchange information; material property block data; sorption parameters; biological parameters; temperature data; and output control parameters. Creates pointers and performs basic error checking on input data.
input2.f	subroutine	Reads the initial and boundary conditions and performs basic error checking of input data. Reads restart information.
miser.f	main program	Performs primary controls of simulation: initiates the read of all input data; loops over all time steps; controls cycling between calls to appropriate routines for solution of the flow and transport equations; controls time step size; and controls calls to output routines.
mobil.f	subroutine	Evaluates capacity coefficients, and aqueous and gas phase mobility terms in stacked storage.
molewt.f	subroutine	Updates the gas, aqueous, and organic liquid phase molecular weight, phase molar density, and phase mass density, based on composition, temperature and pressure.
mpex.f	subroutine	Computes the mole and mass exchange terms for the flow and transport routines.
napls.f	subroutine	Updates the immobile organic liquid saturation using the finite element solution of the organic liquid phase mass balance equation.

Table 5.1 (continued).

Routine	Type	Description
naplx.f	subroutine	Updates the immobile organic liquid component mole fractions using the finite element solution of the component molar balance equation for the organic liquid phase.
prnt.f	subroutine	Writes current values of selected variables to the main output, contour plot, or time series plot files.
satw.f	subroutine	Computes water and gas saturation at all nodes based on current values of nodal capillary pressure.
solid.f	subroutine	Updates the solid phase mass loadings using the finite element solution of the component mass balance equation for the solid phase.
tlhs.f	subroutine	Assembles the finite element matrices for the two dimensional component mole balance equations.
trans.f	subroutine	Controls the sequential solution of the component molar balance equations for all components in all phases. The solution order is: the organic components, water, oxygen and nutrient. For each component the solution order is: biophase; gas phase, aqueous phase, organic liquid phase; and solid phase.
vel.f	subroutine	Computes the mobile phase specific discharge. This routine is not called when constant velocity simulations are being run.

Table 5.2: Selected parameter variables defined in the include file 'dimen.inc.'

Type	Variable	Description
integer	nmx	Maximum number of nodes.
integer	nelmx	Maximum number of elements.
integer	nmbk	Maximum number of material property blocks.
integer	nxmax	Maximum number of horizontal blocks in a generated grid.
integer	nzmax	Maximum number of vertical blocks in a generated grid.
integer	ncmpb	Maximum number of biomass populations. Currently restricted to one.
integer	ncmpo	Maximum number of organic components.
integer	ncmp	Maximum number of components; 3(always) + ncmpb + ncmpo + 1(if nutrient is present).
integer	nnstk	Maximum number of nodal variables in stacked storage. Currently computed as $1.05 * nmx$.
integer	nsolve	Maximum number of unknowns in the linear system. Currently set to $2 * nmx$.
integer	icnl	Maximum number of nonzero entries in the coefficient matrix.
integer	irnl	Maximum number of nonzero entries in the coefficient matrix.
real	srwmin	Minimum difference between the specified residual aqueous phase saturation and the computed aqueous phase saturation. Currently set to 10^{-16} .
real	sgtest	Minimum value of gas phase saturation for which a transport equation is written. Currently set to 0.05.
real	u	Solver parameter. Currently set to 0.1.
integer	mtype	Solver parameter. Currently set to 1.
real	xmino	Minimum mole fraction for convergence testing in the routine naplx.f. Currently set to 10^{-3} .
real	xround	Minimum mole fraction for convergence testing. Currently set to 10^{-16} .
real	smino	Minimum sustainable organic liquid saturation. Currently set to 10^{-16} .

Table 5.3: Description of input and output files.

File name	input/output	Unit	Description
miser.d1	input	11	Contains user supplied input data for the model control options, grid information, and the physical, chemical, and biological parameters.
D2*	input	13	Contains user supplied input data defining the initial and boundary conditions.
miser.error	input	14	Data file containing error and warning messages.
restart.file†	input	28	A restart input file. Contains restart information necessary to either continue a terminated run or to use results from a previous simulation as the initial conditions for a new run. This file is a renamed copy of 'outpre.rst' generated as output
outpre‡.out	output	21	Main output file containing: a listing of most input variables; a description of boundary and initial conditions; simulation results for specified variables at selected times.
outpre.cnv	output	23	Listing of convergence history and runtime performance.
outpre.err	output	22	Listing of runtime generated error and warning messages.
outpre.mb	output	25	Listing of runtime generated global mass balance calculations.
outpre.con	output	24	A contour plot data file. Lists values of selected variables at specified times together with their nodal coordinates in a column format.
outpre.plt	output	26	A time series plot data file. Lists values of selected variables for each time step.
outpre.rst	output	27	A restart output file generated at the end of the simulation. Contains restart information necessary to either continue a terminated run or to use results from a previous simulation as the initial conditions for a new run.

* input file name is a user defined input variable and can contain path information

† 'restart.file' for the input restart file is a user defined input variable and can contain path information

‡ prefix 'outpre' for all output file names is a user defined input variable and can contain path information

Table 5.4: Input Data in Block A – Input/Output Files and Control Options.

Record	Type	Variable	Description
<i>Field 1 - Input files:</i>			
1	char*20	infile(2)	Path and name of file D2 containing the initial and boundary conditions (must be in single quotes).
2	char*20	infile(3)	Path and name of error message file (must be in single quotes). Normally the file is named 'miser.error' and is located in the directory with the executable code.
<i>Field 2 - Prefix name of all output files:</i>			
1	char*16	outpre	Path and prefix name for all output files (must be in single quotes); e.g. if outpre is defined as 'vent', then the main output file is named 'vent.out'.
<i>Field 3 - Output unit numbers for error and performance information:</i>			
1	integer	ipt(29)	The unit number of the output file to which error messages should be directed: 0 = do not print error messages; 6 = screen; 21 = main output file; or 22 = error message file. The file 'outpre.err' is opened when unit number 22 is specified.
2	integer	ipt(28)	The unit number of the output file to which runtime performance information should be directed: 0 = do not print performance information; 6 = screen; 21 = main output file; or 23 = convergence history file. The file 'outpre.cnv' is opened when unit number 23 is specified.
<i>Field 4 - Contour plot file:</i>			
1	logical	lctrl(23)	Set the switch to .true. to open the contour plot output data file 'outpre.con'.
<i>Field 5 - Mass Balance Output:</i>			
1	logical	lprnt(6)	Set the switch to .true. if material balance information should be computed and printed in the output file 'outpre.mb'.
2	logical	lprnt(25)	This record begins on a new line and is required only if lprnt(6) is .true. Set this switch to .true. if the print interval for material balance output is set by the number of iterations. Otherwise set the switch to .false. if the print interval is a constant time interval.
3	logical	lprnt(27)	This record is required only if lprnt(6) is .true. Set this switch to .true. if the material balance output is in report form, otherwise the output will be in multiple files in time series form.
4	integer real	ipt(83) t(27)	This record begins on a new line and is required only if lprnt(6) is .true. Enter the number of iterations if lprnt(25) = .true., or the time interval if lprnt(25) = .false. (s).
<i>Field 6 - Time series output file:</i>			
1	logical	lctrl(15)	Switch is set to .true. to open the time series plot output data file 'outpre.plt'.

Table 5.4 (continued).

Record	Type	Variable	Description
2	logical	lprnt(26)	This record begins on a new line and is required only if lctrl(15) is .true. Set this switch to .true. if the print interval for time series output is set by the number of iterations. Otherwise set the switch to .false. if the print interval is a constant time interval.
3	logical	lprnt(28)	This record is required only if lctrl(15) is .true. A logical switch indicating the concentration units in the time series output: set to .true. for mole fraction; otherwise set to .false. for mass concentration.
4	integer real	ipt(84) t(28)	This record begins on a new line and is required only if lctrl(15) is .true. Enter the number of iterations if lprnt(26) = .true., or the time interval if lprnt(26) = .false. (s).
<i>Field 7 - Restart file:</i>			
1	logical	lctrl(5)	Switch is set to .true. to open and print restart data to the file 'outpre.rst'.
<i>Field 8 - Uniform print interval to the main output file:</i>			
1	logical	lprnt(0)	Set this switch to .true. if the print interval for MISER output to the main output file 'outpre.out' is set by the number of iterations. Otherwise set the switch to .false. indicating the print interval is a constant time interval.
2	integer real	ipt(25) t(12)	This record begins on a new line. Enter the number of iterations if lprnt(0) = .true., or the time interval if lprnt(25) = .false. (s).

Table 5.5: Input Data in Block B – General Model Control Options.

Record	Type	Variable	Description
<i>Field 1 - Coordinate system:</i>			
1	integer	ipt(27)	Variable designating the coordinate system: 0 = cross-sectional ($x-z$); 1 = axisymmetric ($r-z$).
2	real	t(21)	Horizontal component of the gravity vector (m / s^2).
3	real	t(22)	Vertical component of the gravity vector (m / s^2).
<i>Field 2 - Equation solution options:</i>			
1	logical	lctrl(1)	Switch is set to .true. if the aqueous and gas phase mass balance equations are to be solved.
2	logical	lctrl(2)	Switch is set to .true. if the component mass balance equations are to be solved.
3	logical	lctrl(24)	Switch is set to .true. if the NAPL phase mass balance equations are to be solved. Set to .false. if NAPL is absent.
4	logical	lctrl(25)	Switch is set to .true. if the solid phase mass balance equations are to be solved. Set to .false. if equilibrium sorption or no sorption is considered .
5	logical	lctrl(3)	Switch is set to .true. if the biotransformation equations are to be solved. Set to .false. if biotransformations are not considered.
<i>Field 3 - Mass lumping options:</i>			
1	logical	lctrl(7)	Switch is set to .true. for mass matrix lumping in the solution of the phase mass balance equations.
2	logical	lctrl(8)	Switch is set to .true. for mass matrix lumping in the solution of the component mass balance equations.
<i>Field 4 - Flow solution skipping:</i>			
1	integer	ipt(85)	Number of time steps to be skipped between solutions of the phase phase mass balance equations. For example if the flow equations are to be solved every other time step, then a skipping factor of 1 is specified.
<i>Field 5 - Not currently used:</i>			
<i>Field 6 - Coupling between flow and transport:</i>			
1	logical	lctrl(14)	Switch is set to .true. if mass exchange terms should be included in the solution of the flow equations.
<i>Field 7 - Element dimensionless numbers:</i>			
1	logical	lctrl(4)	Switch is set to .true. if element dimensionless numbers should be calculated for the transport solution.

Table 5.6: Input Data Block C – Time Step and Iteration Control Information.

Record	Type	Variable	Description
<i>Field 1 - Simulation time frame:</i>			
1	real	t(1)	Initial simulation time (s).
2	real	t(2)	Final simulation time (s).
<i>Field 2 - Time weighting:</i>			
1	real	t(10)	Time weighting parameter: 0 = explicit; 1 = implicit; 0.5 = Crank-Nicolson.
<i>Field 3 - Number of time steps:</i>			
1	integer	ipt(30)	Maximum number of time steps.
<i>Field 4 - Convergence tolerance:</i>			
1	real	t(13)	Convergence tolerance in the solution of the phase mass balance equations.
2	real	t(14)	Convergence tolerance in the solution of the component mass balance equations for the mobile phases (aqueous and gas).
3	real	t(15)	Convergence tolerance in the solution of the organic phase saturation equations.
4	real	t(16)	Convergence tolerance in the solution of the component mass balance equations for the immobile phases.
<i>Field 5 - Time step range:</i>			
1	real	t(3)	Initial time step (s).
2	real	t(4)	Minimum time step (s).
3	real	t(5)	Maximum time step (s).
<i>Field 6 - Iterations for convergence:</i>			
1	integer	ipt(31)	Maximum number of iterations for convergence of the phase mass balance equations.
2	integer	ipt(32)	Maximum number of iterations for convergence of the component balance equations.
3	integer	ipt(33)	Maximum number of iterations for convergence of the organic phase saturation equations.
<i>Field 7 - Iterations for time step amplification:</i>			
1	integer	ipt(34)	Number of iterations in the solution of the phase mass balance equations below which time step amplification is permissible. Must be less than the maximum number of iterations - ipt(31).
2	integer	ipt(35)	Number of iterations in the solution of the component mass balance equations below which time step amplification is permissible. Must be less than the maximum number of iterations - ipt(32).
<i>Field 8 - Time step multiplications factors:</i>			
1	real	t(6)	Empirical time step amplification factor. Must be greater than or equal to one.
2	integer	t(7)	Empirical time step reduction factor. Must be less than or equal to one.

Table 5.7: Input Data Block D – Grid Information and Control Options Information.

Record	Type	Variable	Description
<i>Field 1 - Output grid geometry:</i>			
1	logical	lprnt(1)	Set switch to .true. to output all grid geometry to the main output file.
<i>Field 2 - Grid specification options:</i>			
1	integer	igrd	Integer variable indicating if a grid should be generated: 0 = input all element numbers and nodal coordinates; 1 = generate a union jack grid; 2 = generate herring bone grid.
<i>Field 3 - Number of blocks in the generated grid (required only if igrd > 0):</i>			
1	integer	nx	Number of blocks in the horizontal direction of the generated grid.
2	integer	ny	Number of blocks in the vertical direction of the generated grid.
<i>Field 4 - Horizontal Block spacing in the generated grid (required only if igrd > 0):</i>			
1	logical	ldel	Set to .true. if the horizontal spacing is uniform.
2	real	xzero	Horizontal coordinate of the left boundary. This should be equal to the well radius in an axial symmetric domain.
3	real	delx	Starting on a new line, enter the horizontal spacings (m) from left to right. Provide a single value if ldel = .true., otherwise provide nx values. If a single negative value is provided then the nodal spacing is calculated from eq. (4.12).
<i>Field 5 - Vertical Block spacing in generated grid (required only if igrd > 0):</i>			
1	logical	ldel	Set to .true. if the vertical spacing is uniform.
2	real	zzero	Vertical coordinate of the top boundary.
3	real	delz	Starting on a new line, enter of the vertical spacings (m) from top to bottom. Provide a single value if ldel = .true., otherwise provide nz values.
<i>Field 6 - Material property blocks in the generated grid (required only if igrd > 0):</i>			
1	integer	ipt(26)	Number of horizontally aligned material property blocks in the generated grid.
2	integer	imblk	Required if the ipt(26) > 1. Starting on a new line enter nz integer values corresponding to the material block number of each vertical spacing from top to bottom.
<i>Field 7 - Grid dimensions (required only if igrd = 0):</i>			
1	integer	ipt(0)	Number of nodes in the grid.
2	integer	ipt(1)	Number of elements in the grid.
3	integer	ipt(26)	Number of material property blocks in the grid.
<i>Field 8 - Nodal incidence list (required only if igrd = 0):</i>			
1	integer	iel	element number.
2	integer	nodal(3*iel-2)	Global node number of node 1.
3	integer	nodal(3*iel-1)	Global node number of node 2.
4	integer	nodal(3*iel)	Global node number of node 3.
5	integer	matel(iel)	Material block number of element iel. Only required if ipt(26) ≠ 1. Field 8 is repeated for all elements, with data for each element beginning on a new line. The elements do not need to be listed in sequential order.
<i>Field 9 - Nodal coordinates (required only if igrd = 0):</i>			
1	integer	ind	Node number.
2	real	xnode(ind)	x-coordinate (horizontal) of node ind (m).
3	real	znode(ind)	z-coordinate (vertical) of node ind (m). Field 9 is repeated for all nodes, with data for each node beginning on a new line. The nodes do not need to be listed in sequential order.

Table 5.8: Input Data Block E – Component Chemical Properties.

Record	Type	Variable	Description
<i>Field 1 - Number of NAPL components:</i>			
1	integer	ipt(15)	Enter the number of organic liquid phase components.
<i>Field 2 - NAPL component chemical properties (required only if ipt(15) > 0):</i>			
1	integer	ic	Component number - must range between 1 and ipt(15).
2	char*10	cname(ic)	Component name - must be enter in single quotes.
3	real	cmw(ic)	Component molecular weight (g / mole).
4	real	cvp(ic)	Component vapor pressure (atm). A negative value indicates this component in involatile and is excluded from the gas phase composition.
5	real	cvvis(ic)	Component vapor viscosity (cPoise).
6	real	cden(ic)	Component liquid density (g / l).
7	real	cmdif(2*ic-1)	Component gas diffusivity (cm ² / s).
8	real	cmdif(2*ic)	Component aqueous diffusivity (cm ² / s).
9	real	chen(ic)	Component Henry's Law constant (atm l / g). Not currently used.
10	real	casol(ic)	Component aqueous solubility (g / l). A negative value indicates this component in insoluble and is excluded from the aqueous phase composition. Field 2 is repeated for all components in the organic liquid phase. Data for each component must begin on a new line.
<i>Field 3 - Chemical property data for water, oxygen, and nitrogen:</i>			
1-10	-	-	Provide the same 10 data inputs described in field 2 above for water, oxygen, and nitrogen. The input order and component numbers are fixed: water = ipt(15)+1; oxygen = ipt(15)+2; and nitrogen = ipt(15)+3.
<i>Field 4 - Nutrient inclusion:</i>			
1	logical	lctrl(9)	Set to .true. if a nutrient is to be modeled.
<i>Field 5 - Nutrient chemical properties (required only if lctrl(9) = .true.):</i>			
1-10	-	-	Provide the same 10 data inputs described in field 2 for the nutrient component. The component number for nutrient must equal ipt(15)+4.

Table 5.9: Input Data Block F – Mass Transfer Coefficients.

Record	Type	Variable	Description
<i>Field 1 - Interphase mass exchange coefficients and minimum deviations from equilibrium:</i>			
1	integer	ic	Component number as defined in data block E.
2	real	kex(5*ic-4)	Aqueous/gas mass exchange coefficient (sec^{-1}).
3	real	kex(5*ic-3)	Aqueous/NAPL mass exchange coefficient (sec^{-1}).
4	real	kex(5*ic-2)	Gas/NAPL mass exchange coefficient (sec^{-1}).
5	real	kex(5*ic-1)	Aqueous/biophase mass exchange coefficient (sec^{-1}).
6	real	kex(5*ic)	Aqueous/solid mass exchange coefficient (sec^{-1}).
7	integer	ic	Component number as defined in data block E. This record must start on a new line.
8	real	kmax(5*ic-4)	Aqueous/gas minimum deviation from equilibrium.
9	real	kmax(5*ic-3)	Aqueous/NAPL minimum deviation from equilibrium.
10	real	kmax(5*ic-2)	Gas/NAPL minimum deviation from equilibrium.
11	real	kmax(5*ic-1)	Aqueous/biophase minimum deviation from equilibrium.
12	real	kmax(5*ic)	Aqueous/solid minimum deviation from equilibrium.
Field 1 is repeated for all components. Data for each component must be separated by at least one comment line. When a component aqueous-solid mass transfer coefficient is defined to be zero, that component is not present in the solid phase. A negative value of the aqueous-solid mass exchange coefficient indicates that the Freundlich $K_f = f_{oc} * K_{input}$ where f_{oc} is defined in bfoc and K_{input} is defined in bok.			

Table 5.10: Input Data Block G – Material Property Block Information.

Record	Type	Variable	Description
<i>Field 1 - Soil physical properties:</i>			
1	integer	iblk	Material property block number - must range between 1 and ipt(26).
2	real	bphi(iblk)	Porosity (-).
3	real	bpermh(iblk)	Horizontal component of intrinsic permeability (m ²).
4	real	bpermv(iblk)	Vertical component of intrinsic permeability (m ²).
5	real	bsden(iblk)	Bulk soil density (g / cm ³).
6	real	bfoc(iblk)	Solid phase organic carbon fraction. Field 1 is repeated for all material property blocks. Data for each block must start on a new line.
<i>Field 2 - Water retention parameters:</i>			
1	integer	iblk	Material property block number - must range between 1 and ipt(26).
2	real	bsrw(iblk)	Residual water saturation (-).
3	real	bvgn(iblk)	'n' parameter of the van Genuchten fitting function for air/water retention data.
4	real	bvga(iblk)	'α' parameter of the van Genuchten fitting function for air/water retention data (Pa ⁻¹). Field 2 is repeated for all material property blocks. Data for each block must start on a new line.
<i>Field 3 - Dispersion parameters:</i>			
1	integer	iblk	Material property block number - must range between 1 and ipt(26).
2	real	bdisl(iblk)	Longitudinal dispersivity (m).
3	real	bdist(iblk)	Transverse dispersivity (m). Field 3 is repeated for all material property blocks. Data for each block must start on a new line.
<i>Field 4 - Dispersion tensor computation:</i>			
1	logical	lctrl(21)	Set to .true. if the dispersion tensor should be calculated as a function of the dispersivities and velocity distribution. Enter .false. if a constant dispersion tensor is to be input.
<i>Field 4 - Dispersion tensor (required only if lctrl(21) = .false.):</i>			
1	integer	ic	Component number as defined in data block E.
2	real	d(8*ic-7)	D_{xx}^h - dispersion coefficient of component ic in the gas phase (m ² /s).
3	real	d(8*ic-6)	D_{xy}^h - dispersion coefficient of component ic in the gas phase (m ² /s).
4	real	d(8*ic-5)	D_{yx}^h - dispersion coefficient of component ic in the gas phase (m ² /s).
5	real	d(8*ic-4)	D_{yy}^h - dispersion coefficient of component ic in the gas phase (m ² /s).
6	integer	ic	Component number as defined in data block E. This record must start on a new line.
7	real	d(8*ic-7)	D_{xx}^h - dispersion coefficient of component ic in the aqueous phase (m ² /s).
8	real	d(8*ic-6)	D_{xy}^h - dispersion coefficient of component ic in the aqueous phase (m ² /s).
9	real	d(8*ic-5)	D_{yx}^h - dispersion coefficient of component ic in the aqueous phase (m ² /s).
10	real	d(8*ic-4)	D_{yy}^h - dispersion coefficient of component ic in the aqueous phase (m ² /s). Field 4 is repeated for all components. Data for each component must be separated by at least one comment line.

Table 5.11: Input Data Block H – Sorption Parameter Data.

Record	Type	Variable	Description
<i>Field 1 - Sorption Model (required if lctrl(25) = .true.):</i>			
1	logical	lctrl(19)	Set to .true. if a two compartment sorption model is used (can only be used under conditions of a homogeneous soil domain and a single component organic liquid). Set to .false. if sorption is modeled with a single compartment model.
<i>Field 2 - Two compartment sorption parameters (required if lctrl(25) = .true. and lctrl(19) = .true.):</i>			
1	real	xbok	Multiplier to convert the slow compartment value of k_f defined in bok to the fast compartment value.
2	real	xbom	Multiplier to convert the slow compartment value of n defined in bom to the fast compartment value.
3	real	xkex	Multiplier to convert the slow compartment value of the exchange coefficient to the fast compartment value.
4	real	xden	Mass fraction of solid phase in the fast compartment.
<i>Field 3 - Sorption parameters (required if lctrl(25) = .true.):</i>			
1	integer	iblk	Material property block number - must range between 1 and ipt(26).
2	real	bok(see text)	Freundlich isotherm k_f parameter for each organic component in order from 1 to the number of components (enter ipt(15) values). Units are $\mu\text{g} / \text{g}$ solid, with aqueous concentration in mg / l . Index number is $(\text{iblk}-1)*\text{ipt}(15)+\text{ic}$.
3	integer	iblk	Material property block number - must range between 1 and ipt(26). This record must start on a new line.
4	real	bom(see text)	Freundlich isotherm n parameter for each organic component in order from 1 to the number of components (enter ipt(15) values). Index number is $(\text{iblk}-1)*\text{ipt}(15)+\text{ic}$. Field 3 is repeated for all material property blocks. Data for each block must start on a new line.
<i>Field 4 - Include retardation factors (required if lctrl(25) = .false.):</i>			
1	logical	lretrd	Set to .true. if retardation factors should be used.
<i>Field 5 - Retardation factors (required if lctrl(25) = .false. and lretrd = .true.):</i>			
1	integer	ic	Component number - must range between 1 and ipt(15)+3 if no nutrient is present, or between 1 and ipt(15)+4 if nutrient is present.
2	real	krtd(ic)	Retardation factor. Field 5 is repeated for all components with data for each component beginning on a line.

Table 5.12: Input Data Block I – Biological Parameter Data.

Record	Type	Variable	Description
<i>Field 1 - Number of biodegradable substrates:</i>			
1	integer	ipt(17)	Specify the number of biodegradable substrates. Must be less than or equal to the number of components in the organic liquid (ipt(15)).
<i>Field 2 - Biodegradation control switches:</i>			
1	logical	lctrl(17)	Set to .true. if a steady state biomass is to be modeled, otherwise set to .false. if the biomass is time dependent.
2	logical	lctrl(16)	Set to .true. if biotransformations are modeled as a sink term in the aqueous transport equations; otherwise set to .false. if a separate rate limited biophase is modeled.
<i>Field 3 - Growth kinetics options:</i>			
1	integer	ipt(39)	Variable indicating the type of growth kinetics: 1 = standard Monod kinetics; 2 = Monod kinetics with substrate inhibition; 3 = Monod kinetics with lumped substrate inhibition; 4 = Monod kinetics with saturation dependency; 5 = Monod kinetics with saturation dependency and substrate inhibition.
<i>Field 4 - Monod parameters:</i>			
1	integer	ic	Component number as defined in block E.
2	real	fuse(ic,1)	Electron acceptor use coefficient (mole O ₂ / mole substrate).
3	real	fuse(ic,2)	Nutrient use coefficient (mole nutrient / mole substrate).
4	real	umax(ic)	Maximum substrate use rate (g substrate / g biomass / sec).
5	real	khalf(ic)	Half saturation constant (g substrate / l).
6	real	xyield(ic)	Yield coefficient (g biomass / g substrate).
7	real	kinhib(ic)	Inhibition constant (unitless) expressed as a fraction of the aqueous solubility. Constant multiplies the sum of all substrate aqueous solubilities for type 3 growth kinetics. Field 4 is repeated for all degradable substrates, for oxygen, and additionally for nutrient, if present (i.e. field 4 is repeated ipt(17)+1 times if no nutrient is present, and ipt(17)+2 times if nutrient is present). Data for each component must start on a new line.
<i>Field 5 - Decay and biomass range coefficients:</i>			
1	real	kd	Decay coefficient (sec ⁻¹).
2	real	xbmin	Minimum biomass (g biomass / l media).
3	real	xbmax	Maximum biomass (g biomass / l media).
4	real	xinit	Initial uniform biomass (g biomass / l media).
5	real	t(11)	Delay period for initiation of bioreactions (sec).

Table 5.13: Input Data Block J – Phase Parameter Data.

Record	Type	Variable	Description
<i>Field 1 - Water phase viscosity:</i>			
1	real	wvis	Water phase viscosity (cPoise).
<i>Field 2 - Gas phase slip flow parameters:</i>			
1	logical	lctrl(20)	Set to .true. if gas phase slip flow is simulated with the Klinkenberg model.
2	real	b	Klinkenberg parameter (atm). Set the value to zero if lctrl(20) = .false.

Table 5.14: Input Data Block K – Temperature Parameter Data.

Record	Type	Variable	Description
<i>Field 1 - Temperature distribution:</i>			
1	logical	lctrl(10)	Set to .true. if the steady state temperature distribution is uniform, or set to .false. if the temperature distribution is depth dependent.
<i>Field 2 - Uniform temperature (required only if lctrl(10) = .true.):</i>			
1	real	ctemp	Specify the uniform temperature (°C).
<i>Field 3 - Nonuniform temperature distribution (required if lctrl(10) = .false.):</i>			
1	real	depthnd	Depth (m).
2	real	tnode	Temperature at the depth = depthnd (°C). Field 3 is repeated for all vertical nodes along the boundary starting at the surface, downward (ny+1 values). Data for each node begins a new line.
<i>Field 4 - Temperature dependent vapor pressure (required if lctrl(10) = .false.):</i>			
1	real	dtemp	Temperature dependent vapor pressure (atm). Provide ny+1 values, one for each node in the vertical direction.
<i>Field 5 - Temperature dependent vapor viscosity (required if lctrl(10) = .false.):</i>			
1	real	dtemp	Temperature dependent vapor viscosity (cPoise). Provide ny+1 values, one for each node in the vertical direction.
<i>Field 6 - Temperature dependent Henry's Law constant (required if lctrl(10) = .false.):</i>			
1	real	dtemp	Temperature dependent Henry's Law constant (atm l/g). Provide ny+1 values, one for each node in the vertical direction.
<i>Field 7 - Temperature dependent aqueous solubility (required if lctrl(10) = .false.):</i>			
1	real	dtemp	Temperature dependent aqueous solubility (g / l). Provide ny+1 values, one for each node in the vertical direction.
<i>Field 8 - Temperature dependent maximum substrate use rate (required if lctrl(10) = .false.):</i>			
1	real	dtemp	Temperature dependent maximum substrate use rate (g substrate / g biomass / sec). Provide ny+1 values, one for each node in the vertical direction. Fields 4-8 are repeated for all components in the order established in block E: organic liquid components, water, oxygen, nitrogen, nutrient. Data for each component property begins on a new line, separated from the previous information by at least one comment line.
<i>Field 9 - Temperature dependent biomass decay rate (required if lctrl(10) = .false.):</i>			
1	real	dtemp	Temperature dependent biomass decay rate (sec ⁻¹). Provide ny+1 values, one for each node in the vertical direction.

Table 5.15: Input Data Block L – Output Control parameters.

Record	Type	Variable	Description
<i>Field 1 - Print initial conditions:</i>			
1	logical	lprnt(3)	Set to .true. if the initial conditions should be printed to the main output file.
<i>Field 2 - Print switches for selected variables:</i>			
1a	logical	lprnt(8)	Phase concentrations in the main output file are reported in mole fractions (.true.) or mass concentration (.false.).
1b	logical	lcon(1)	Phase concentrations in the contour plot file are reported in mole fractions (.true.) or mass concentration (.false.).
2a	logical	lprnt(9)	Print nodal gas phase pressure to the main output file. This record starts on a new line.
2b	logical	lcon(2)	Print nodal gas phase pressure to the contour plot file.
3a	logical	lprnt(10)	Print nodal aqueous phase pressure to the main output file. This record starts on a new line.
3b	logical	lcon(3)	Print nodal aqueous phase pressure to the contour plot file.
4a	logical	lprnt(11)	Print nodal gas/aqueous capillary pressure to the main output file. This record starts on a new line.
4b	logical	lcon(4)	Print nodal gas/aqueous capillary pressure to the contour plot file.
5a	logical	lprnt(12)	Print nodal gas phase density to the main output file. This record starts on a new line.
5b	logical	lcon(5)	Print nodal gas phase density to the contour plot file.
6a	logical	lprnt(13)	Print nodal aqueous phase density to the main output file. This record starts on a new line.
6b	logical	lcon(6)	Print nodal aqueous phase density to the contour plot file.
7a	logical	lprnt(14)	Print nodal NAPL phase density to the main output file. This record starts on a new line.
7b	logical	lcon(7)	Print nodal NAPL phase density to the contour plot file.
8a	logical	lprnt(15)	Print nodal gas phase component concentrations to the main output file. This record starts on a new line.
8b	logical	lcon(8)	Print nodal gas phase component concentrations to the contour plot file.
8c	integer	ipt(69)	The number of gas phase components to be outputted. This sub-record is required only if lprnt or lcon is .true. in 8a or 8b above. This sub-record starts on a new line.
8d	integer	icp	Enter ipt(69) component numbers of the corresponding components to be outputted. This sub-record is required only if lprnt or lcon is .true. in 8a or 8b above.
9a	logical	lprnt(16)	Print nodal aqueous phase component concentrations to the main output file. This record starts on a new line.
9b	logical	lcon(9)	Print nodal aqueous phase component concentrations to the contour plot file.
9c	integer	ipt(70)	The number of aqueous phase components to be outputted. This sub-record is required only if lprnt or lcon is .true. in 9a or 9b above. This sub-record starts on a new line.
9d	integer	icp	Enter ipt(70) component numbers of the corresponding components to be outputted. This sub-record is required only if lprnt or lcon is .true. in 9a or 9b above.

Table 5.15 (continued).

Record	Type	Variable	Description
10a	logical	lprnt(17)	Print nodal organic liquid component concentrations to the main output file. This record starts on a new line.
10b	logical	lcon(10)	Print nodal organic liquid component concentrations to the contour plot file.
10c	integer	ipt(71)	The number of organic liquid components to be outputted. This sub-record is required only if lprnt or lcon is .true. in 10a or 10b above. This sub-record starts on a new line.
10d	integer	icp	Enter ipt(71) component numbers of the corresponding components to be outputted. This sub-record is required only if lprnt or lcon is .true. in 10a or 10b above.
11a	logical	lprnt(18)	Print nodal solid phase component loadings to the main output file. This record starts on a new line.
11b	logical	lcon(11)	Print nodal solid phase component loadings to the contour plot file.
11c	integer	ipt(72)	The number of solid phase component loadings to be outputted. This sub-record is required only if lprnt or lcon is .true. in 11a or 11b above. This sub-record starts on a new line.
11d	integer	icp	Enter ipt(72) component numbers of the corresponding components to be outputted. This sub-record is required only if lprnt or lcon is .true. in 11a or 11b above.
12a	logical	lprnt(19)	Print nodal biophase component concentrations to the main output file. This record starts on a new line.
12b	logical	lcon(12)	Print nodal biophase component concentrations to the contour plot file.
12c	integer	ipt(73)	The number of biophase component concentrations to be outputted. This sub-record is required only if lprnt or lcon is .true. in 12a or 12b above. This sub-record starts on a new line.
12d	integer	icp	Enter ipt(73) component numbers of the corresponding components to be outputted. This sub-record is required only if lprnt or lcon is .true. in 12a or 12b above.
13a	logical	lprnt(29)	Print total organic soil concentration to the main output file. This record starts on a new line.
13b	logical	lcon(18)	Print total organic soil concentration to the contour plot file.
14a	logical	lprnt(20)	Print nodal gas phase saturation to the main output file. This record starts on a new line.
14b	logical	lcon(13)	Print nodal gas phase saturation to the contour plot file.
15a	logical	lprnt(21)	Print nodal aqueous phase saturation to the main output file. This record starts on a new line.
15b	logical	lcon(14)	Print nodal aqueous phase saturation to the contour plot file.
16a	logical	lprnt(22)	Print nodal NAPL saturation to the main output file. This record starts on a new line.
16b	logical	lcon(15)	Print nodal NAPL saturation to the contour plot file.
17a	logical	lprnt(23)	Print gas phase velocity to the main output file. This record starts on a new line.
17b	logical	lcon(16)	Print gas phase velocity to the contour plot file.
18a	logical	lprnt(24)	Print aqueous phase velocity to the main output file. This record starts on a new line.
18b	logical	lcon(17)	Print aqueous phase velocity to the contour plot file.

Table 5.15 (continued).

Record	Type	Variable	Description
<i>Field 3 - Gas Phase time series plot switches:</i>			
1	logical	lplt(1)	Set to .true. if the time series plot files should be generated for gas phase components.
2	integer	ipt(81)	If lplt = .true. then starting a new line, enter the number of gas phase components to be reported in the time series plot file.
3a	integer	icp	If lplt = .true. enter the global component number and the nodal location for which time series data should be outputted.
3b	integer	icp	If lplt = .true. enter the nodal location for which time series data should be outputted. Field 3a and 3b is repeated ipt(81) times.
<i>Field 4 - Aqueous Phase time series plot switches:</i>			
1	logical	lplt(2)	Set to .true. if the time series plot files should be generated for aqueous phase components.
2	integer	ipt(82)	If lplt = .true. then starting a new line, enter the number of aqueous phase components to be reported in the time series plot file.
3a	integer	icp	If lplt = .true. enter the global component number. and the nodal location for which time series data should be outputted.
3b	integer	icp	If lplt = .true. enter the nodal location for which time series data should be outputted. Field 3a and 3b is repeated ipt(82) times. A maximum of 6 components for the combined gas and aqueous phases can be defined for time series output.

Table 5.16: Input Data Block M – Restart Identifier.

Record	Type	Variable	Description
<i>Field 1 - Restart control switches:</i>			
1	logical	lctrl(26)	Specify .true. if initial saturation and component information should be read from the restart file 'restart.file', where the file path and name is defined in field 2 of this data block.
2	logical	lctrl(32)	Specify .true. if the run is a continuation of the previous run and .false. if the run is a new run using the previous run as initial conditions.
<i>Field 2 - Restart file identifier: (required only if lctrl(26)=.true.</i>			
1	char*20	infile(4)	Path and name of restart input file (must be in single quotes). This file is a renamed copy of 'outpre.rst' generated as output.

Table 5.17: Input Data Block N – Initial Pressure Conditions.

Record	Type	Variable	Description
<i>Field 1 - Initial pressure distribution:</i>			
1	integer	ipt(75)	Variable indicating how initial conditions are specified: 1 = compute hydrostatic gas and aqueous phase distributions; 2 = input gas and aqueous pressures for all nodes.
<i>Field 2 - Water table depth (required only if ipt(75) = 1):</i>			
1	real	wtdpth	Water table depth (m).
<i>Field 3 - Initial pressure:</i>			
1	integer	nd	Node number.
2	real	p(2*nd-1)	Aqueous phase pressure at node nd (Pa gauge).
3	real	p(2*nd)	Gas phase pressure at node nd (Pa gauge).
Field 3 is repeated for all nodes with data for each node beginning on a new line. The nodes do not need to be in sequential order. A uniform pressure distribution can be specified by specifying a single line of data containing a negative node number and the uniform aqueous and gas pressures.			

Table 5.18: Input Data Block O – Velocity Computation.

Record	Type	Variable	Description
<i>Field 1 - Velocity computation method:</i>			
1	logical	lctrl(18)	Set to .true. if nodal velocities should be computed by solution of finite element equations. Set to .false. if velocities should be computed as element averages, or if a steady state velocity field is assumed (i.e. flow equations are not solved - lctrl(1) = .false.).
<i>Field 2 - Steady state velocity distribution (required only if lctrl(1) = .false.):</i>			
1	logical	lcssv	Set to .true. if the steady state velocity distribution should be calculated from the pressure field by the method defined in field 1. Set to .false. to input velocity values at all nodes.
<i>Field 3 - Input velocity distribution (required only if lcssv = .false.):</i>			
1	logical	lcv	Set to .true. if the user defined steady state velocity distribution has uniform components.
<i>Field 4 - Uniform velocity components (required only if lcv = .true.):</i>			
1	real	qgx	Horizontal component of the uniform steady state gas phase Darcy velocity (m / s).
2	real	qgz	Vertical component of the uniform steady state gas phase Darcy velocity (m / s).
3	real	qax	Horizontal component of the uniform steady state aqueous phase Darcy velocity (m / s).
4	real	qaz	Vertical component of the uniform steady state aqueous phase Darcy velocity (m / s).
<i>Field 5 - Nonuniform velocity components (required only if lcv = .false.):</i>			
1	integer	i	Node or element number.
2	real	q(i)	Horizontal component of the steady state gas phase Darcy velocity (m / s) at node or element i.
3	real	q(ipt(1)+i)	Vertical component of the steady state gas phase Darcy velocity (m / s) at node or element i.
4	real	q(2*ipt(1)+i)	Horizontal component of the steady state aqueous phase Darcy velocity (m / s) at node or element i.
5	real	q(3*ipt(1)+i)	Vertical component of the steady state aqueous phase Darcy velocity (m / s) at node or element i. Field 5 is repeated for all nodes if lctrl(18) = .true., or for all elements if lctrl(18) = .false. Data for each node or element must start on a new line.

Table 5.19: Input Data Block P – Organic Liquid Saturation and Composition.

Record	Type	Variable	Description
<i>Field 1 - Elements containing organic liquid saturation:</i>			
1	integer	inoel	Number of elements containing nonzero organic liquid saturation. A number less than zero indicates that the organic liquid saturation is uniform and contained in all elements. A value of zero must be entered if the organic liquid mass balance equations are not solved (lctrl(24)=.false.).
<i>Field 2 - Uniform organic liquid saturation and composition (required only if inoel < 0):</i>			
1	real	soel(1)	Uniform organic liquid saturation for all elements.
2	real	omfel(ic)	Enter the organic liquid mole fraction of each organic component. There must be ipt(15) mole fractions specified and they must sum to 1. Mole fractions are entered in sequential order as defined in block E, field 2 (i.e. component number 1 to ipt(15)).
<i>Field 3 - Nonuniform organic liquid saturation and composition (required only if inoel > 0):</i>			
1	integer	iel	Element number.
2	real	soel(iel)	Organic liquid saturation in element iel.
3	real	omfel(ic,iel)	Enter the organic liquid mole fraction of each organic component in element iel. There must be ipt(15) mole fractions specified and they must sum to 1. Mole fractions are entered in sequential order as defined in block E, field 2 (i.e. component number 1 to ipt(15)). Field 3 is repeated for all elements with nonzero organic liquid saturation (inoel elements). Data for each element must start on a new line.

Table 5.20: Input Data Block Q – Oxygen and Nutrient Initial Conditions.

Record	Type	Variable	Description
<i>Field 1 - Uniform gas phase conditions (required if oxygen is present in the gas phase):</i>			
1	logical	lunfx	Set to .true. if the initial oxygen and nutrient (if present) partial pressure in the gas phase is uniform. Otherwise set to .false. for nonuniform initial conditions. Skip this input if oxygen is absent from the gas phase (i.e. the oxygen partial pressure (cvp(ipt(15)+2)) is assigned a negative value).
<i>Field 2 - Uniform oxygen and nutrient conditions in the gas phase (lunfx = .true.):</i>			
1	real	xog	Initial uniform oxygen partial pressure (i.e. mole fraction) in the gas phase.
2	real	xng	Initial uniform nutrient partial pressure (i.e. mole fraction) in the gas phase. Not required if nutrient is absent (i.e. lctrl(9) = .false.).
<i>Field 3 - Nonuniform oxygen and nutrient conditions in the gas phase (lunfx = .false.):</i>			
1	integer	nd	Node number.
2	real	xmf(nd+go)	Initial oxygen partial pressure (i.e. mole fraction) in the gas phase at node nd (go is an internally defined pointer).
3	real	xmf(nd+gn)	Initial nutrient partial pressure (i.e. mole fraction) in the gas phase at node nd (gn is an internally defined pointer). Not required if nutrient is absent (i.e. lctrl(9) = .false.). Field 3 is repeated for all nodes. Data for each node must start on a new line.
<i>Field 4 - Uniform aqueous phase conditions (required if O₂ is present in the aqueous phase):</i>			
1	logical	lunfx	Set to .true. if the initial oxygen and nutrient (if present) concentrations in the aqueous phase are uniform. Otherwise set to .false. for nonuniform initial conditions. Skip this input if oxygen is absent from the aqueous phase (i.e. the oxygen solubility (casol(ipt(15)+2)) is assigned a negative value). Note that this variable is read twice in this data block.
<i>Field 5 - Uniform oxygen and nutrient conditions in the aqueous phase (lunfx = .true.):</i>			
1	real	xog	Initial uniform oxygen concentration (g / l) in the aqueous phase.
2	real	xng	Initial uniform nutrient concentration (g/l) in the aqueous phase. Not required if nutrient is absent (i.e. lctrl(9) = .false.).
<i>Field 6 - Nonuniform oxygen and nutrient conditions in the aqueous phase (lunfx = .false.):</i>			
1	integer	nd	Node number.
2	real	xmf(nd+ao)	Initial oxygen concentration (g/l) in the aqueous phase at node nd (ao is an internally defined pointer).
3	real	xmf(nd+an)	Initial nutrient concentration (g/l) in the aqueous phase at node nd (ao is an internally defined pointer). Not required if nutrient is absent (i.e. lctrl(9) = .false.). Field 6 is repeated for all nodes. Data for each node must start on a new line.

Table 5.21: Input Data Block R – Boundary Conditions.

Record	Type	Variable	Description
<i>Field 1 - Constant gas pressure nodes equivalent to the initial pressure:</i>			
1	integer	itype1	Specify the number of nodes with a constant gas pressure equal to the initial gas pressure.
2	integer	ibc	Beginning on a new line specify the node number of all nodes with a constant gas pressure equal to the initial gas pressure. There must be a total of itype1 values.
<i>Field 2 - Constant gas pressure nodes different from the initial pressure:</i>			
1	integer	ipt(18)	Specify the number of nodes with a constant gas pressure that is different from the initial gas pressure.
2	integer	ibc	Node number of a constant gas pressure node different from the initial gas pressure at that node (required if ipt(18) > 0).
3	real	p(2*ibc)	Constant gas pressure at node ibc (Pa gauge). Records 2 and 3 are repeated for all nodes, with data for each node starting on a new line. There must be a total of ipt(18) lines.
<i>Field 3 - Constant aqueous pressure nodes equivalent to the initial pressure:</i>			
1	integer	itype1	Specify the number of nodes with a constant aqueous pressure equal to the initial aqueous pressure.
2	integer	ibc	Beginning on a new line specify the node number of all nodes with a constant aqueous pressure equal to the initial aqueous pressure. There must be a total of itype1 values.
<i>Field 4 - Constant aqueous pressure nodes different from the initial pressure:</i>			
1	integer	ipt(19)	Specify the number of nodes with a constant aqueous pressure that is different from the initial aqueous pressure.
2	integer	ibc	Node number of a constant aqueous pressure node different from the initial gas pressure at that node (required if ipt(19) > 0).
3	real	p(2*ibc-1)	Constant aqueous pressure at node ibc (Pa gauge). Records 2 and 3 are repeated for all nodes, with data for each node starting on a new line. There must be a total of ipt(18) lines.
<i>Field 5 - Gas phase component boundary conditions:</i>			
1	integer	ipt(20)	Specify the number of nodes for which gas phase component boundary conditions are specified.
2	integer	ibc	Node number (required if ipt(20) > 0).
3	integer	ibcxmf	An integer variable indicating the boundary condition type (1, 2, or 3) for all gas phase components at the node ibc: 1 = constant mole fraction; 2 = constant diffusive flux; 3 = mixed type (contact with a known fluid).
4 & 5	real	bcxmf dfxmf	For each component in the gas phase provide two values: 1) the partial pressure (i.e. mole fraction) of the component in the contacting fluid at node ibc, and 2) the molecular diffusivity divided by a characteristic length, D_m/L (m/s). Each pair of values must be listed in sequential order corresponding to the component numbers. Only components that are present in the gas phase are listed; component boundary conditions are not read for components which are excluded from the gas phase (i.e. negative vapor pressure).

Table 5.21 (continued).

Record	Type	Variable	Description
			Records 2-5 are repeated for all nodes for which gas phase component boundary conditions are provided (ipt(20) nodes). Data for each node must start on a new line.
<i>Field 6 - Aqueous phase component boundary conditions:</i>			
1	integer	ipt(21)	Specify the number of nodes for which aqueous phase component boundary conditions are specified.
2	integer	ibc	Node number (required if ipt(21) > 0).
3	integer	ibcxmf	An integer variable indicating the boundary condition type (1, 2, or 3) for all aqueous phase components at the node ibc: 1 = constant mole fraction; 2 = constant diffusive flux; 3 = mixed type (contact with a known fluid).
4 & 5	real	bcxmf dfxmf	For each component in the aqueous phase provide two values: 1) the concentration of the component concentration in the contacting fluid at node ibc (g/l), and 2) the molecular diffusivity divided by a characteristic length, D_m/L (m/s). Each pair of values must be listed in sequential order corresponding to the component numbers. Only components that are present in the aqueous phase are listed; component boundary conditions are not read for components which are excluded from the aqueous phase (i.e. negative solubility). Records 2-5 are repeated for all nodes for which aqueous phase component boundary conditions are provided (ipt(21) nodes). Data for each node must start on a new line.
<i>Field 7 - Gas phase boundary fluxes:</i>			
1	integer	ipt(22)	Specify the number of nodes for which a constant gas phase flux is to specified.
2	integer	ibc	Node number (required if ipt(22) > 0).
3	real	source	The constant gas phase flux referenced to atmospheric pressure and the steady temperature at the node (m^3/s). Records 2-3 are repeated for all nodes for which gas phase fluxes are provided (ipt(22) nodes). Data for each node must start on a new line.
<i>Field 8 - Aqueous phase boundary fluxes:</i>			
1	integer	ipt(23)	Specify the number of nodes for which a constant aqueous phase flux is to specified.
2	integer	ibc	Node number (required if ipt(23) > 0).
3	real	source	The constant aqueous phase flux at the node (m^3 / s). Records 2-3 are repeated for all nodes for which aqueous phase fluxes are provided (ipt(23) nodes). Data for each node must start on a new line.

Table 5.22: Input Data Block S – Extraction/Injection Well Conditions.

Record	Type	Variable	Description
<i>Field 1 - Include extraction/injection well:</i>			
1	logical	lctrl(12)	Specify .true. if a extraction/injection well should be simulated in an axisymmetric domain.
<i>Field 2 - Extraction/injection rate:</i>			
1	real	qwell	Enter the volumetric extraction (negative) or injection (positive) rate (scfm).
<i>Field 3 - Well coordinates:</i>			
1	real	rwell	Enter the well radius (m). This must equal the nodal coordinate along the left vertical boundary.
2	integer	ii	Minimum node number along the well screen.
3	integer	jj	Maximum node number along the well screen.

Table 5.23: Input Data Block T – Velocity Boundary Conditions.

Record	Type	Variable	Description
<i>Field 1 - Specify bottom boundary (required only if lctrl(18) = .true.):</i>			
1	logical	lctrl(28)	Specify .true. if the bottom boundary is impervious.
<i>Field 2 - Specify right boundary (required only if lctrl(18) = .true.):</i>			
1	logical	lctrl(29)	Specify .true. if the right boundary is impervious.
<i>Field 3 - Specify left boundary (required only if lctrl(18) = .true.):</i>			
1	logical	lctrl(30)	Specify .true. if the left boundary is impervious (note: this boundary is adjusted for the presence of a well).
<i>Field 4 - Specify top boundary (required only if lctrl(18) = .true.):</i>			
1	logical	lctrl(31)	Specify .true. if the top boundary is impervious.
<i>Field 5 - Specify the cap length (required only if lctrl(31) = .true.):</i>			
1	real	caplen	Length of the impervious segment along the top boundary starting at the left edge (m).

Section 6

DEMONSTRATION OF MISER

Three example simulations are presented to illustrate usage of the MISER model. The first two examples present simulations of hypothetical SVE and BV scenarios in domains wherein the water table is at a large depth and is therefore excluded from the simulation. The third example problem describes the simulation of bioventing under more realistic field conditions. In this simulation the contaminants are positioned close to the water table such that the capillary fringe and water table are included in the simulation. Simulation results illustrate the influence of contaminant migration and partitioning into the capillary fringe region on the simulated bioventing performance.

Hypothetical SVE and BV scenarios are simulated for a layered soil domain shown in Fig. 6.1. Input files for the SVE simulation are included in Appendix G. The soil is a medium uniform sand, intersected with a layer of slightly less permeable fine sand. The soil water is at residual levels throughout the domain. All soil properties are listed in Table 6.1.

The organic liquid is pure toluene which is assumed to be present as an immobile residual. The initial toluene mass is 239 kg with saturations ranging up to 3.5%. This initial distribution was generated with a two dimensional multiphase flow simulator, M-VALOR [Abriola *et al.*, 1992]. Initial toluene concentrations in the aqueous and gas phases are assumed to be at equilibrium with the organic liquid when present, and zero elsewhere. The constituent and transport properties used in the simulations may be found in the input file listing given in Appendix G.

Numerical simulations of SVE and BV remediation were obtained on a radially symmetric r - z rectangular solution domain (41.67 m by 4.5 m). No flow boundary conditions were specified along the top boundary over the radius of the impermeable cap, along the left boundary above and below the well screen, and along the right boundary.

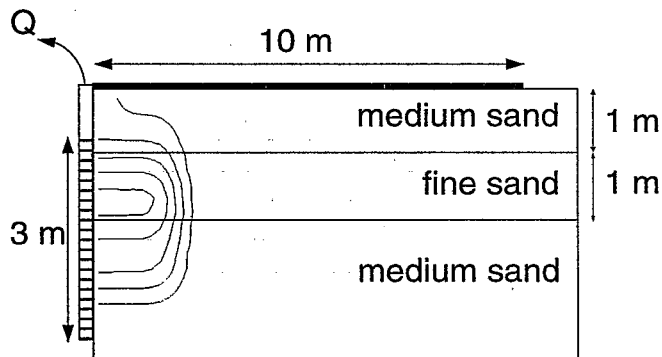


Figure 6.1: Problem depiction used in example simulations. Contours show the initial organic liquid distribution. The contour interval is 0.005 with levels increasing inward.

Parameter	medium sand	fine sand
ϕ	0.35	0.33
k_x (m ²)	1×10^{-11}	6×10^{-12}
k_z (m ²)	8×10^{-12}	4×10^{-12}
S_{rw}	0.12	0.16
n	7.0	5.0
α (1/Pa)	0.002	0.0008

Table 6.1: Soil properties used in example SVE and BV simulations.

Phase Pair	Mass Transfer Coefficient (1/sec)	Minimum Element Deviation from Equilibrium
organic-gas	5.0×10^{-4}	0.1
aqueous-gas	5.0×10^{-5}	0.1
aqueous-organic	5.0×10^{-4}	0.1
aqueous-solid	5.0×10^{-5}	0.1

Table 6.2: Mass transfer coefficients used to simulate an SVE system.

and along the bottom boundary. Atmospheric pressure conditions were specified along the right boundary and the portion of the ground surface open to the atmosphere. Both aqueous and gas phases are assumed to be in hydrostatic equilibrium initially. Other problem dependent conditions are described below.

6.1 SOIL VAPOR EXTRACTION

Remediation of the contaminated soil by SVE was simulated by the numerical application of a constant extraction rate of 100 cubic feet per minute (scfm). Biodegradation is not included in this simulation. The solution domain is developed with a generated "herring bone" grid using uniform vertical spacings of 0.25 m and horizontal spacing ranging from 1 cm near the well to 5 m at the right boundary. The solution domain is divided into 1116 elements and 608 nodes.

Mass transfer coefficients were selected to represent relatively fast organic liquid volatilization and slower rates for desorption and aqueous/gas partitioning. These later processes are considered primary factors controlling long term tailing processes frequently observed in SVE systems. Table 6.2 summarizes the values of mass transfer coefficients used in this simulation.

Fig. 6.2 shows a progression of predicted organic liquid saturation profiles. The greatest rate of organic liquid is in the surface and bottom layers where the initial organic liquid saturations are smallest and gas phase velocities are comparatively large. The organic liquid persists in the middle layer due to the larger initial organic liquid mass present and due to effects from flow bypassing of the lower permeability zone. The pattern of organic liquid removal is generally radially inward since the greatest mass transfer occurs at the outward edge of the organic liquid zone.

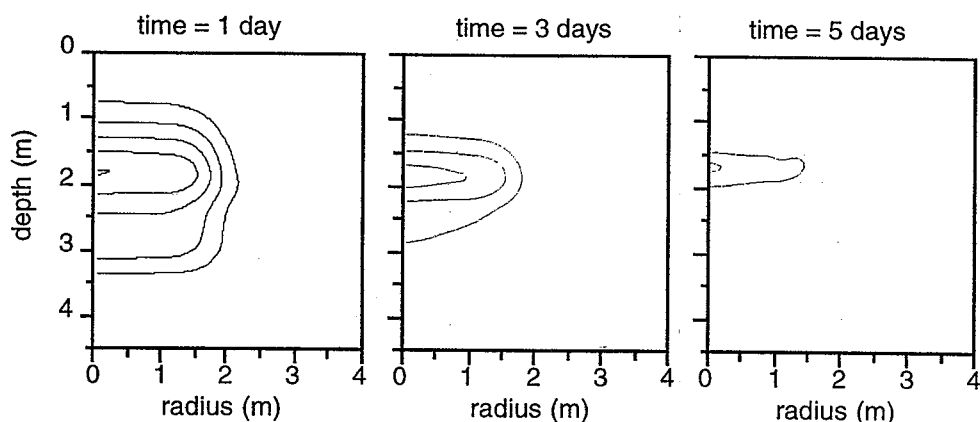


Figure 6.2: Predicted organic liquid saturation distribution in SVE simulations with intermediate mass transfer rates. The contour interval is 0.005 with levels increasing inward.

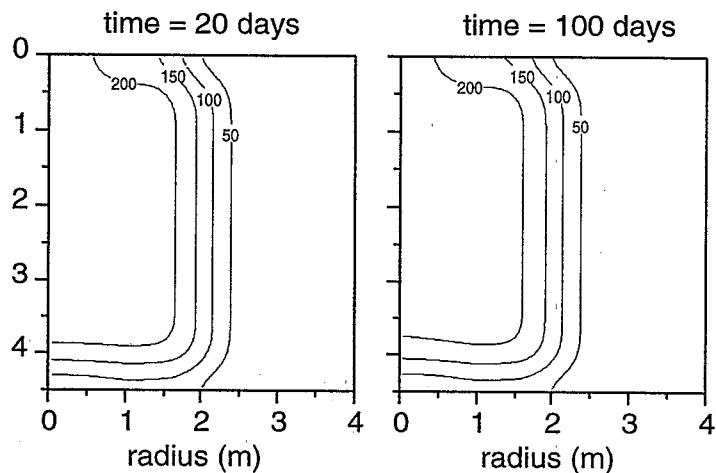


Figure 6.3: Toluene sorbed (ppm) at 20 and 100 days.

Fig. 6.2 shows that organic liquid is removed relatively quickly from the domain, in about 7 days. However, toluene is retained on the solid phase and in the pore water throughout the simulation, due to low solid-aqueous and aqueous-gas mass transfer rates. Comparison of the sorbed toluene mass at 20 and 100 days (Fig. 6.3) indicates that very little sorbed mass is removed during this period. Moreover, toluene mass distributions plotted in Fig. 6.4 indicate that in this simulation the overall SVE efficiency is controlled by the aqueous-solid desorption rate after the period of organic liquid removal.

6.2 BIOVENTING

Remediation of the contaminated soil by BV was simulated by the numerical application of a constant injection rate of 1 cubic feet per minute (scfm) to supply oxygen and enhance biotransformation. Air

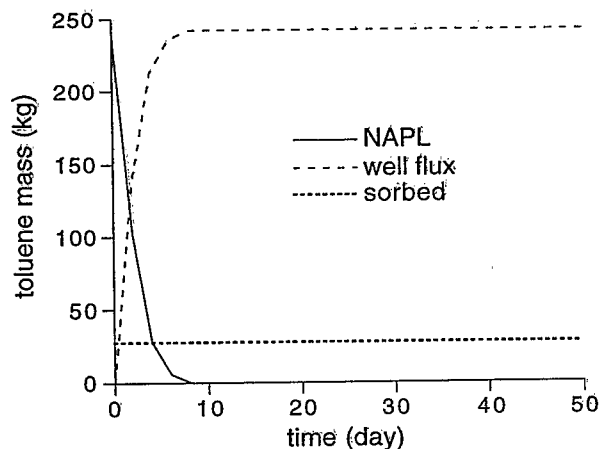


Figure 6.4: Toluene removal versus time.

injection produces outward radial movement of oxygen (electron acceptor), as well as for toluene (substrate) due to volatilization from the organic liquid. Sorption is not included in this simulation. The solution domain is developed with a generated "herring bone" grid using uniform vertical spacings of 0.25 m and horizontal spacing ranging from 1 cm near the well to 2 m at the right boundary. The solution domain is divided into 1404 elements and 660 nodes.

Because gas phase velocities are smaller in the BV system, the fluid-fluid mass transfer coefficients used in this simulation were reduced one order of magnitude from those listed in Table 6.2. The resulting progression of predicted organic liquid saturation profiles is shown in Fig. 6.5. Since the gas flow is radially outward, organic liquid removal occurs from left to right in BV (injection) in contrast with the right to left progression observed in the SVE (extraction) results. Also note that the organic liquid persists substantially longer in the BV system than in the SVE system due to diminished flow rate. Similar to the SVE results, organic liquid persists in the lower portion of the lower permeability layer, where bypassing effects are most pronounced.

Biotransformation processes in the BV scenario were simulated using the assumption of equilibrium partitioning between the aqueous and biophases (i.e. bioreaction was modeled as a sink). The effect of substrate inhibition was examined by setting the inhibitory threshold to 25% of the toluene aqueous solubility. The presence of a limiting nutrient was not considered. Other biodegradation parameters employed in this simulation are listed in Table 6.3.

Fig. 6.6 shows the predicted growth of biomass over the course of BV simulation. Due to substrate inhibition, biomass growth is concentrated away from the NAPL contaminated core, developing a so-called 'biofence.' Once the 'biofence' has developed, the toluene is removed from the gas phase over a relatively short distance, and there is little or no growth to the right of the 'biofence' due to an absence of substrate. Biomass growth is also observed to fill in regions close to the well after organic liquid has been removed and aqueous concentrations fall below the inhibitory threshold. The maximum oxygen depletion was on the order of 10%.

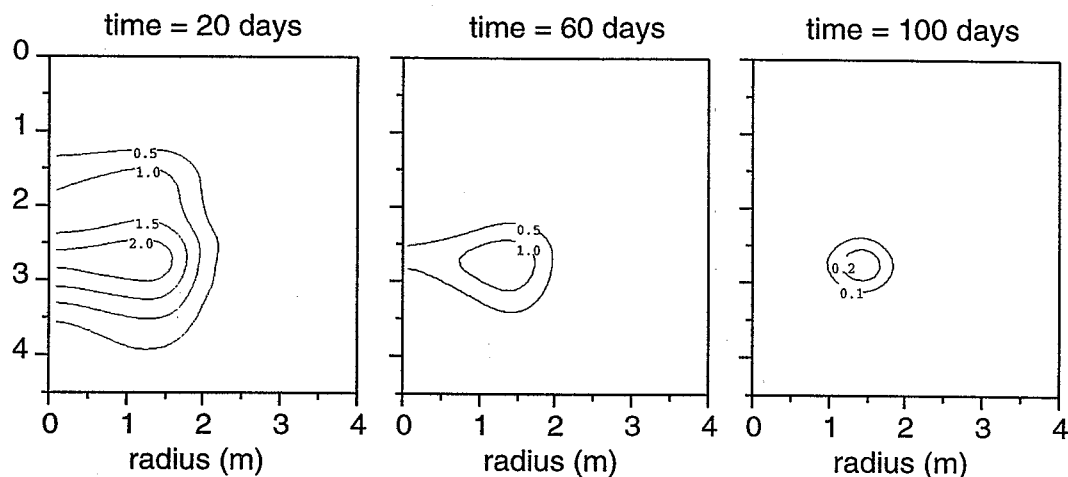


Figure 6.5: Predicted organic liquid saturation (%) at selected times in the example BV simulation.

Parameter	Value
Initial uniform biomass (g/l)	0.00162
Minimum biomass (g/l)	0.001
Maximum biomass (g/l)	0.1
Decay coefficient (1/sec)	1.157×10^{-6}
Oxygen Use coefficient (gm O ₂ /gm toluene)	2.19
Maximum substrate use rate (gm/(gm cell sec))	1.157×10^{-6}
Half saturation constant (gm toluene/l)	
toluene	0.0174
oxygen	0.0001
Yield coefficient (gm cell/gm toluene)	0.50

Table 6.3: Biotransformation parameters used in an example BV simulation.

6.3 FIELD SCALE BIOVENTING

A final example problem highlights a number of capabilities of MISER. In this problem a realistic initial condition is set up in which contaminants from a organic liquid spill are allowed to migrate by diffusion

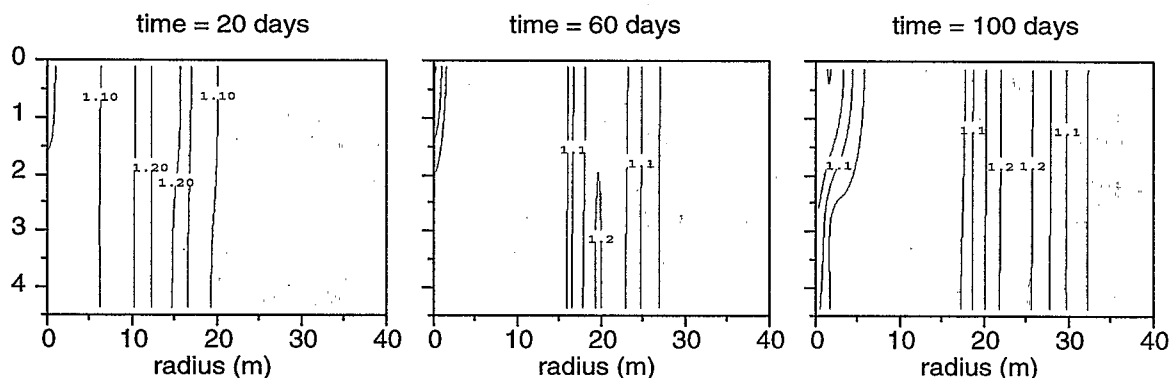


Figure 6.6: Predicted biomass distribution ($\text{g/l} \times 10^{-3}$) at selected times in the example BV simulation.

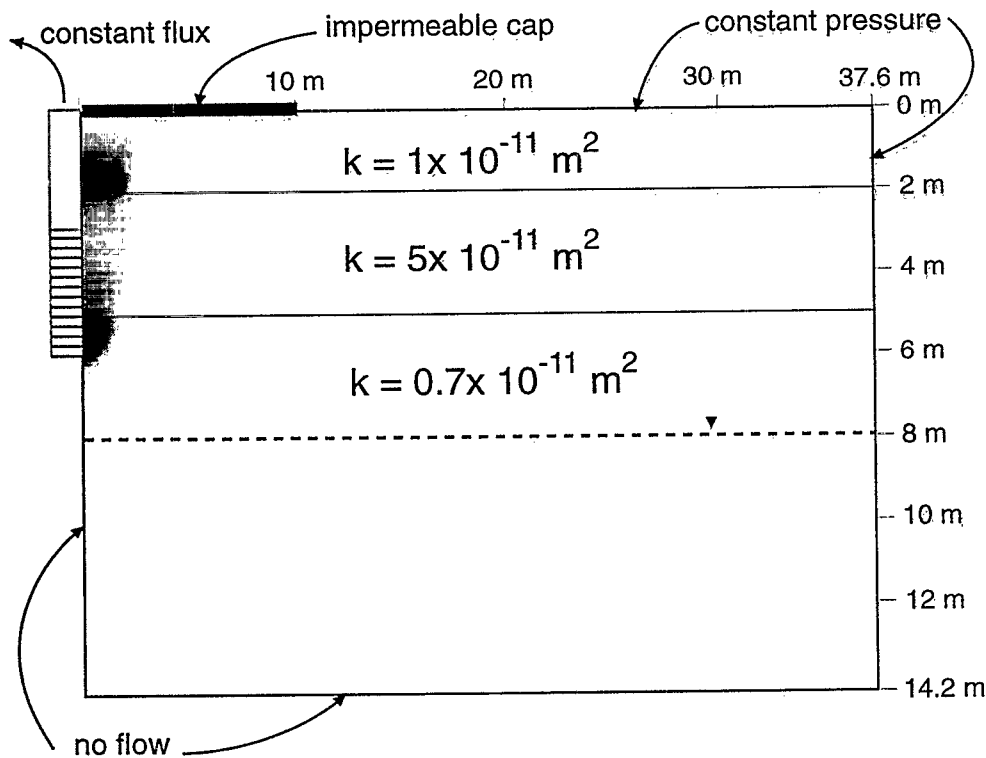


Figure 6.7: Simulation domain used in the field scale bioventing demonstration simulation.

and density driven flow and then partition into the capillary fringe region. MISER is applied to a single well, axisymmetric domain with three horizontally stratified soil layers as shown in Figure 6.7. Soil properties are listed in Table 6.4 and are roughly based on those of the Borden aquifer. An immobile residual organic liquid distribution is present above the water table as shown in Figure 6.7. For the purposes of illustration this organic liquid distribution was developed by simulating the migration from a organic liquid spill event using an immiscible flow simulator, M-VALOR [Abriola *et al.*, 1992]. The initial organic liquid distribution shown in Figure 6.7 is composed of a binary mixture of benzene and xylene. All chemical properties are listed in Table 6.5.

To develop realistic initial conditions for the simulation of BV remediation, the MISER code was first used to simulate the partitioning and migration of the organic liquid components over a 54 day redistribution period during which time no stresses are applied at the well. To simulate this process the domain was discretized into 2337 nodes and 4480 elements, with vertical discretization between nodes ranging between 0.1 and 0.5 m, and horizontal discretization ranging between 0.05 - 2.0 m. Other transport parameters used in the simulation are listed in Table 6.6 and boundary conditions are shown in Figure 6.7. The resulting contaminant and biomass distributions are shown in Figure 6.8. These distributions illustrate the outward migration of contaminants from the core organic liquid distribution. The migration pathway is primarily volatilization of organic liquid constituents into the soil gas wherein they can readily migrate by diffusion and density driven flow. As the organic constituents in the gas phase migrate radially outward they simultaneously partition into the soil water and subsequently sorb to the soil particles. The increase in organic substrate in the soil water results in buildup of the biomass and corresponding depletion of oxygen as shown in the bottom two plots in Figure 6.8. There is no biomass growth within the NAPL contaminated

	Soil 1	Soil 2	Soil 3
ϕ^*	0.33	0.33	0.33
k (m^2)	1.0×10^{-11}	5.0×10^{-11}	0.7×10^{-11}
S_{rw}^*	0.073	0.073	0.073
n^*	3.97	3.97	3.97
α_{aw}^* (Pa^{-1})	4.34×10^{-4}	7.47×10^{-4}	2.79×10^{-4}
benzene K_f (mg/g)/(mg/l) ⁿ	1.16×10^{-3}	1.16×10^{-3}	1.16×10^{-3}
benzene Freundlich n	0.86	0.86	0.86
o-xylene K_f (mg/g)/(mg/l) ⁿ	0.87×10^{-3}	0.87×10^{-3}	0.87×10^{-3}
o-xylene Freundlich n	1.07	1.07	1.07

* Demond and Roberts [1991]

* computed by Leverett scaling

Table 6.4: Soil parameters used in the field scale bioventing demonstration simulation.

	Benzene	o-Xylene	Water	Nitrogen	Oxygen
Molecular weight (g/mole)	78.1	106.2	18.0	28.0	32.0
Density (g/l)*	879.0	880.1	998.2	-	-
Vapor pressure (atm)**	0.102	0.0092	0.0231	-	-
Aqueous solubility (g/l)**	1.78	0.175	-	-	0.009
Vapor viscosity (cPoise)**	0.0075	0.007	0.0095	0.0174	0.02
Liquid viscosity (cPoise)*	0.649	0.809	1.002	-	-
Vapor binary diffusion coeff. (cm^2/s)**	0.088	0.062	0.245	-	0.2
Aqueous binary diffusion coeff. (cm^2/s)†	9.0(-6)	7.2(-6)	-	-	2.7(-5)

* Riddick *et al.*, [1986]

** Perry and Chilton [1973]

† estimated using eq. 17-24 in Lyman *et al.*, [1982]

Table 6.5: Fluid properties used in the field scale bioventing demonstration simulation. All values are for 20 °C.

region due to inhibition.

For comparison purposes bioventing processes were simulated using air injection rates of 1 and 10 cfm and with the low and high growth parameters listed in table 6.6. Figures 6.9-6.12 show results with the low injection rate, high growth conditions. The predicted organic liquid distributions at specified times are presented in Figure 6.9. Results here show behavior similar to the previous example problem in that organic liquid is removed radially outward from the injection well and the fastest rate of removal occurs in the most permeable layer. The predicted aqueous phase substrate concentrations correspondingly decrease radially outward over time as shown in Figure 6.10. Notice, however, that substrate located in the capillary fringe region shows a persistence due to reduced access to the gas stream.

Predicted biomass distribution and corresponding aqueous phase oxygen concentrations are shown in Figures 6.11 and 6.12, respectively. Due to the effects of substrate inhibition, the biomass distribution at

Mass transfer coefficients	diffusion problem	BV simulation
gas-NAPL (1/day)	50	50
gas-aqueous (1/day)	1	1
NAPL-aqueous (1/day)	20	20
solid-aqueous (1/day)	10	1
Monod parameters	high set	low set
maximum substrate utilization (1/d)	1	0.1
half saturation constant (mg/l)	0.5	0.5
decay coefficient (1/d)	0.1	0.01
minimum biomass (mg/l)	0.001	0.001
maximum biomass (mg/l)	20	20

Table 6.6: Mass exchange and biokinetic parameters used in the field scale bioventing demonstration simulation.

156 days is observed to be densest along the outside fringe of the organic liquid zone and near the well screen where organic liquid has been completely removed. High oxygen depletion is observed in the corresponding regions of high biomass growth. The predicted biomass concentrations increase substantially between 156 and 256 days, maintaining high concentrations until the end of the simulation at 356 days. This is explained by the increased availability of oxygen as shown in Figure 6.12 resulting from the decreased oxygen demand due to the reduction of substrate concentrations as shown in Figure 6.10. Notice that oxygen remains depleted in the capillary fringe region due to the poor accessibility to the air stream. Consequently there is limited biomass growth in this region resulting in the persistence of the organic substrate.

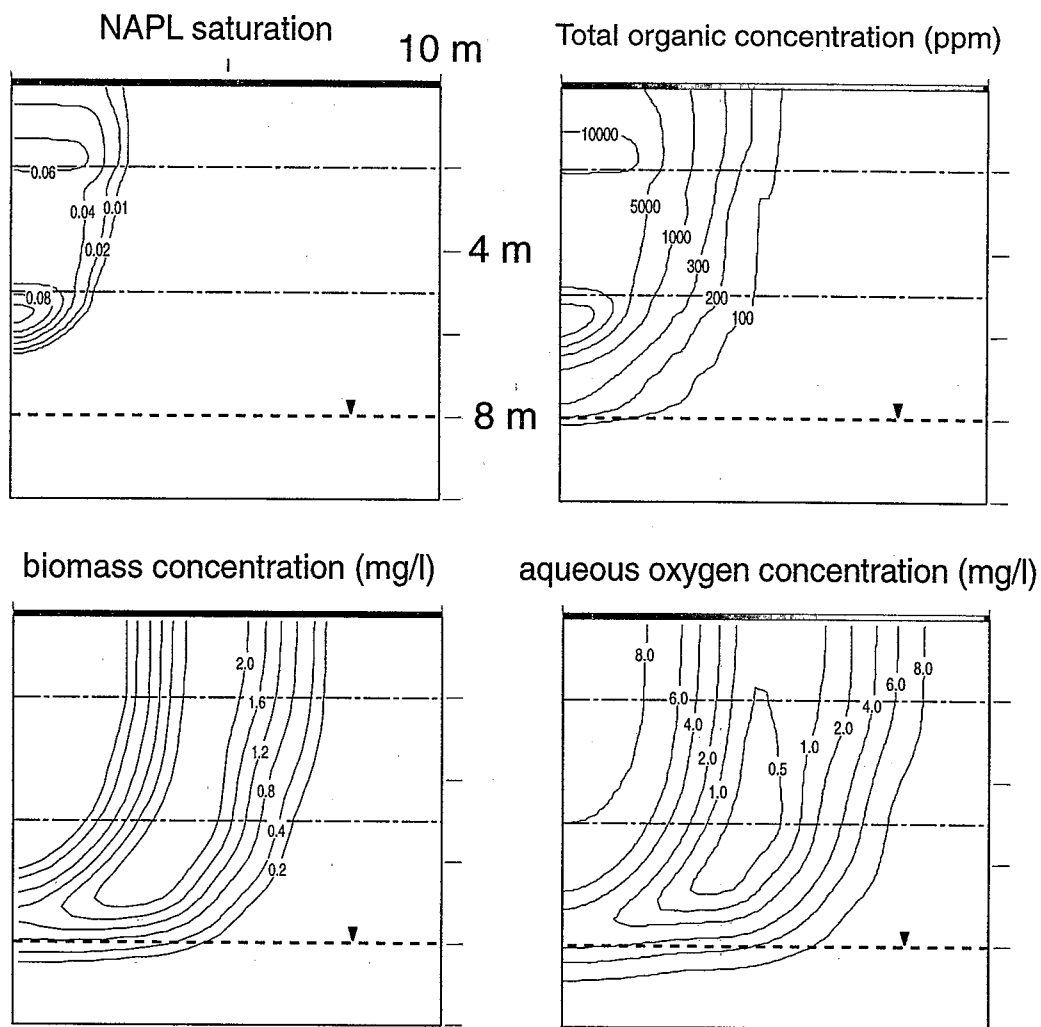


Figure 6.8: Initial conditions used for the field scale bioventing demonstration simulation.

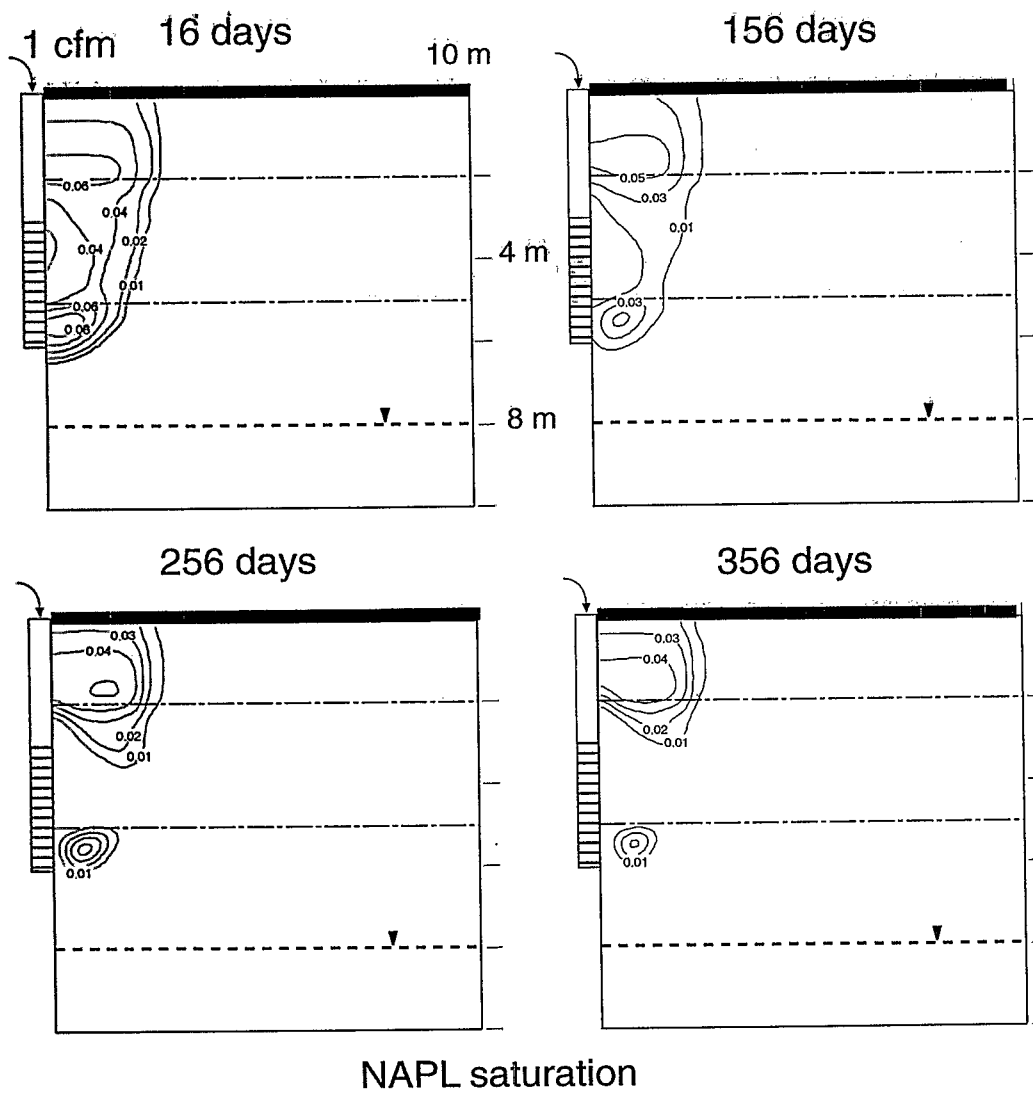


Figure 6.9: Predicted organic liquid distributions at specified times for the field scale bioventing demonstration simulation.

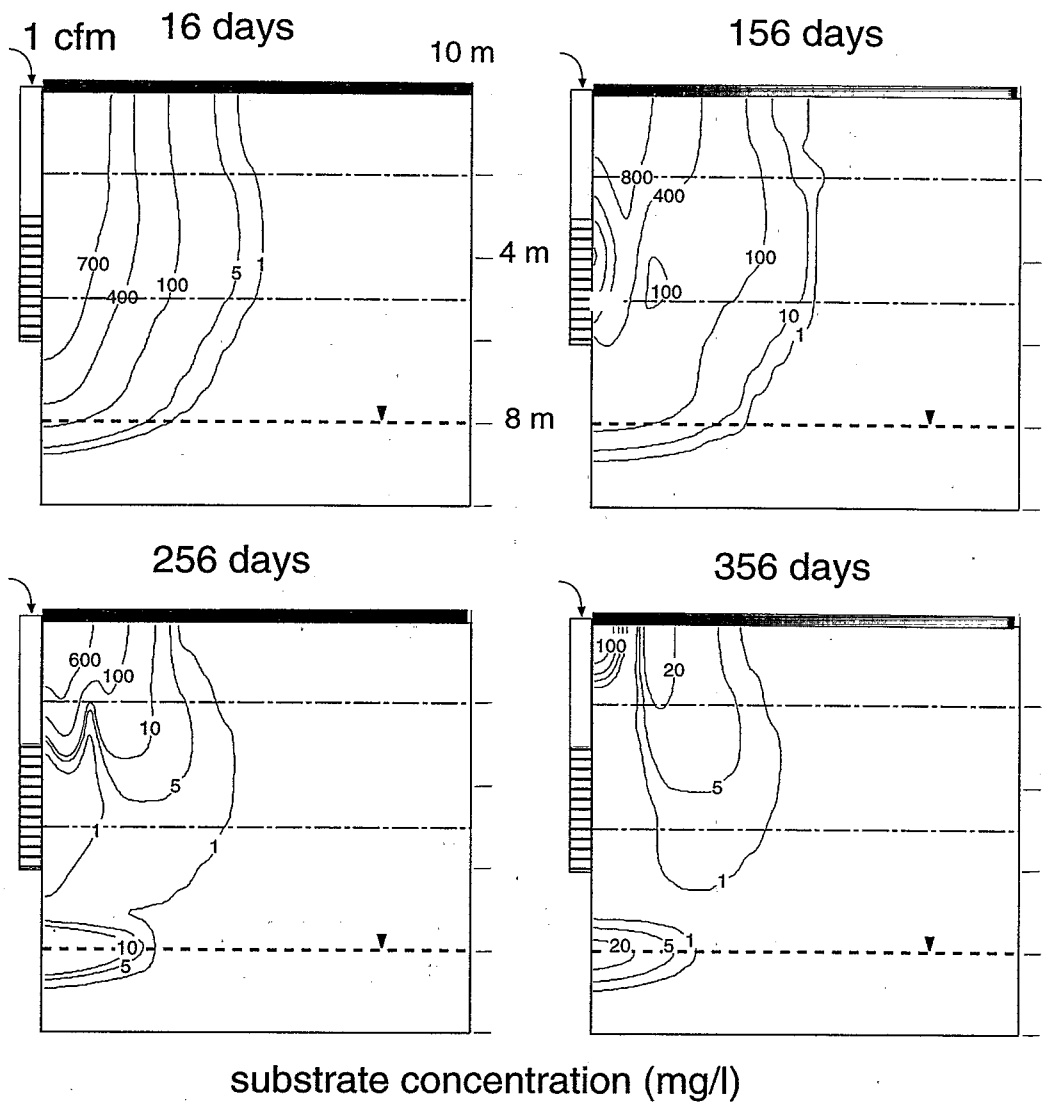


Figure 6.10: Predicted benzene substrate distributions at specified times for the field scale bioventing demonstration simulation.

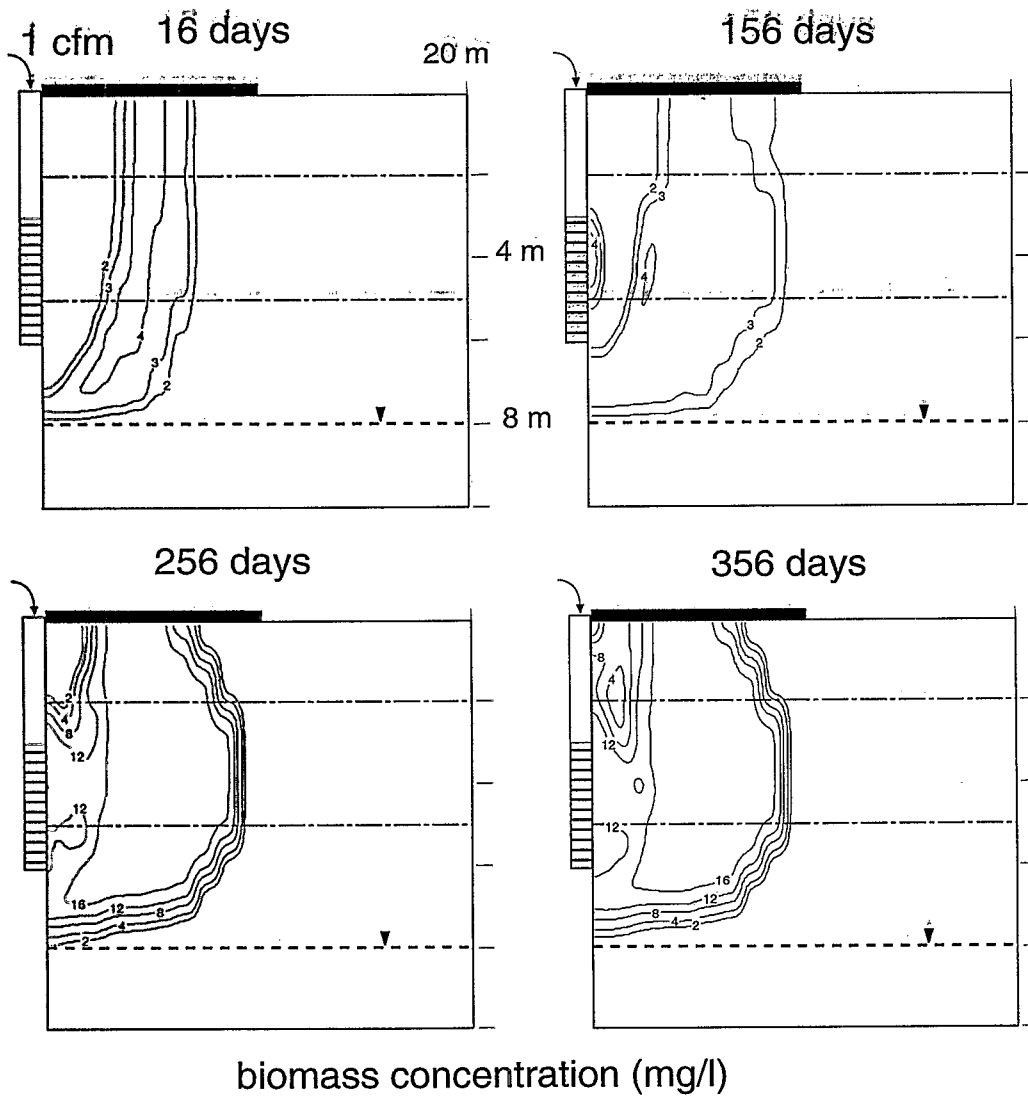


Figure 6.11: Predicted biomass distributions at specified times for the field scale bioventing demonstration simulation.

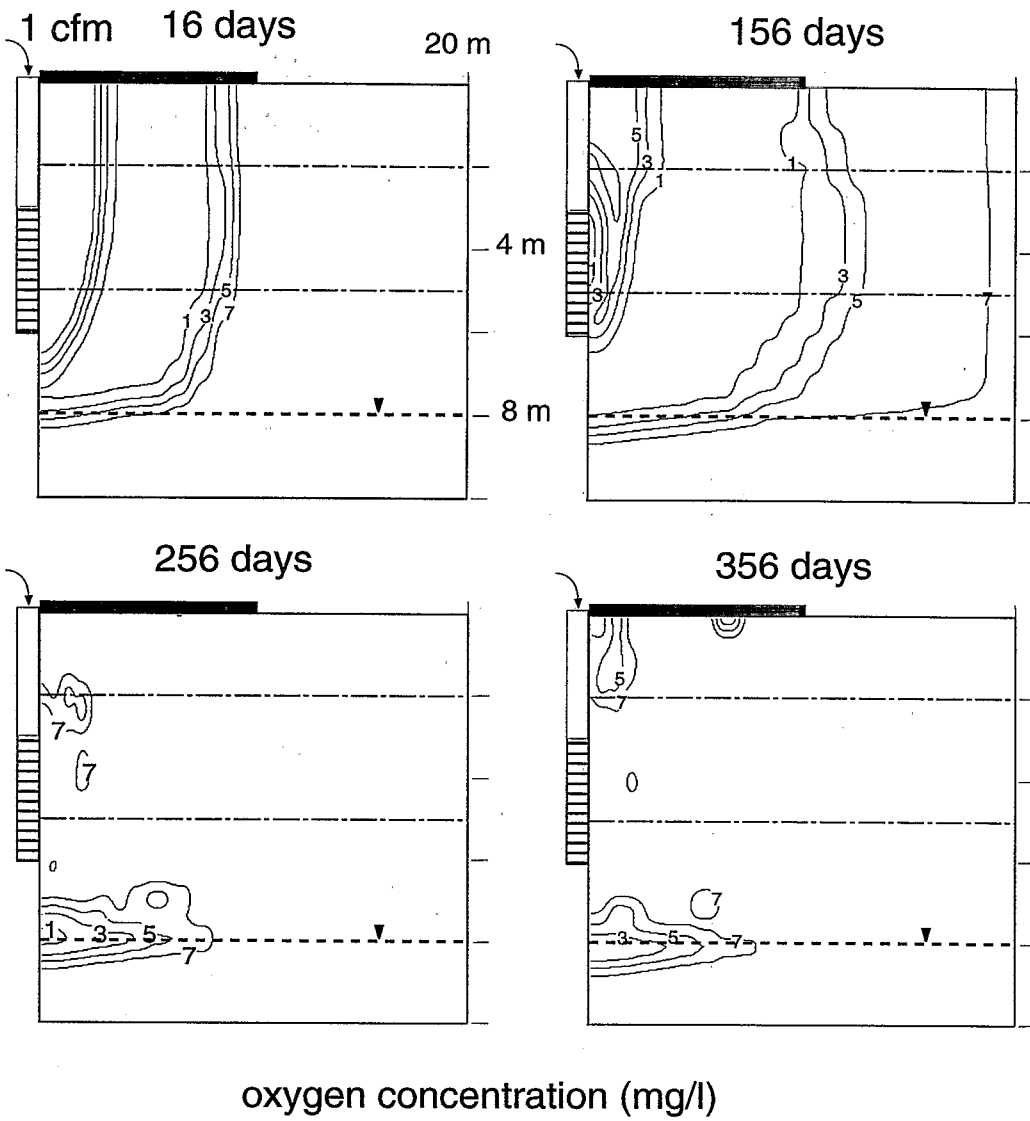


Figure 6.12: Predicted oxygen distributions at specified times for the field scale bioventing demonstration simulation.

Appendix A

ELEMENT MATRICES FOR THE SIMULTANEOUS SOLUTION OF THE PHASE MASS BALANCE EQUATIONS

Integration of (3.17) gives to the element matrix equation,

$$[A]^e \frac{\partial \{P\}^e}{\partial t} + [B]^e \{P\}^e = \{F\}^e + \{E\}^e + \{Q\}^e \quad (\text{A.1})$$

where $\{P\}^e$ is the vector of alternating aqueous and gas pressures at the three nodes of the element,

$$\{P\}^e = \left\{ P_{a_1}^{k+1}, P_{g_1}^{k+1}, P_{a_2}^{k+1}, P_{g_2}^{k+1}, P_{a_3}^{k+1}, P_{g_3}^{k+1} \right\} \quad (\text{A.2})$$

where k is an iteration counter.

The consistent form of the mass matrix is developed from eqs. (3.17a) and (3.17b). It may be expressed as,

$$A_{ij} = \begin{bmatrix} A_{aa} & A_{ag} \\ A_{ga} & A_{gg} \end{bmatrix} \quad (\text{A.3})$$

where,

$$A_{aa} = \sum_{e=1}^{N_e} \phi_e \int_{\Omega_e} -C_{p_j} N_i N_j d\Omega_e$$

$$A_{ag} = A_{ga} = \sum_{e=1}^{N_e} \phi_e \int_{\Omega_e} C_{p_j} N_i N_j d\Omega_e$$

$$A_{gg} = \sum_{e=1}^{N_e} \phi_e \int_{\Omega_e} \left\{ \frac{S_{g_j}}{RT_j \rho_{g_j}} N_j - C_{p_j} \right\} N_i N_j d\Omega_e$$

Evaluating the integrals, the element mass matrix $[A]^e$ in consistent form is,

$$[A]^e = \frac{\phi_e A_e \bar{r}_e}{12} \begin{bmatrix} 2d_{a_1} & -2d_{a_1} & d_{a_2} & -d_{a_2} & d_{a_3} & -d_{a_3} \\ d_{a_1} & -d_{a_1} & 2d_{a_2} & -2d_{a_2} & d_{a_3} & -d_{a_3} \\ d_{a_1} & -d_{a_1} & d_{a_2} & -d_{a_2} & 2d_{a_3} & -2d_{a_3} \\ -2d_{a_1} & 2d_{g_1} & -d_{a_2} & d_{g_2} & -d_{a_3} & d_{g_3} \\ -d_{a_1} & d_{g_1} & -2d_{a_2} & 2d_{g_2} & -d_{a_3} & d_{g_3} \\ -d_{a_1} & d_{g_1} & -d_{a_2} & d_{g_2} & -2d_{a_3} & 2d_{g_3} \end{bmatrix} \quad (\text{A.4})$$

where,

$$d_{a_j} = -C_{p_j}^k$$

$$d_{g_j} = \frac{S_{g_j}^k}{RT_j \rho_{g_j}} - C_{p_j}^k$$

where A_e is the area of element e .

It is often advocated to diagonalize the mass matrix $[A]$ by the procedure of "mass lumping" in order to reduce oscillatory behavior and improve computational stability [Celia *et al.*, 1990; Abriola and Rathfelder, 1993]. Therefore an option is included in MISER for lumping of the mass matrix. The lumped mass matrix is developed by,

$$A_{ij}^L = \delta_{ij} \begin{bmatrix} A_{aa}^L & A_{ag}^L \\ A_{ga}^L & A_{gg}^L \end{bmatrix} \quad (\text{A.5})$$

$$A_{aa}^L = \sum_{e=1}^{N_e} \phi_e \int_{\Omega_e} -C_{pj} N_i d\Omega_e$$

$$A_{ag}^L = A_{ga}^L = \sum_{e=1}^{N_e} \phi_e \int_{\Omega_e} C_{pj} N_i d\Omega_e$$

$$A_{gg}^L = \sum_{e=1}^{N_e} \phi_e \int_{\Omega_e} \left\{ \frac{S_{gj}}{RT_j \rho_{gj}} - C_{pj} \right\} N_i d\Omega_e$$

where δ_{ij} is the Kronecker delta. Evaluating the integrals, the lumped mass matrix is,

$$[A^L]^e = \frac{\phi_e A_e \bar{r}_e}{12} \begin{bmatrix} 4d_{a1} & -4d_{a1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 4d_{a2} & -4d_{a2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 4d_{a3} & -4d_{a3} \\ -4d_{a1} & 4d_{g1} & 0 & 0 & 0 & 0 \\ 0 & 0 & -4d_{a2} & 4d_{g2} & 0 & 0 \\ 0 & 0 & 0 & 0 & -4d_{a3} & 4d_{g3} \end{bmatrix} \quad (\text{A.6})$$

The stiffness matrix developed from eqs. (3.17a) and (3.17b) may be represented by,

$$B_{ij} = \begin{bmatrix} B_{aa} & B_{ag} \\ B_{ga} & B_{gg} \end{bmatrix} \quad (\text{A.7})$$

where,

$$B_{aa} = \sum_{e=1}^{N_e} \int_{\Omega_e} \left\{ \left(\lambda_{axx_j} N_j \right) \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \left(\lambda_{azz_j} N_j \right) \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} - \left(\frac{\lambda_{axx_j} N_j}{\rho_{a_j}^*} \right) \left(\rho_{a_j}^* \frac{\partial N_j}{\partial x} \right) N_i \frac{\partial N_j}{\partial x} - \left(\frac{\lambda_{azz_j} N_j}{\rho_{a_j}^*} \right) \left(\rho_{a_j}^* \frac{\partial N_j}{\partial z} \right) N_i \frac{\partial N_j}{\partial z} \right\} d\Omega_e$$

$$B_{ag} = B_{ga} = 0$$

$$B_{gg} = \sum_{e=1}^{N_e} \int_{\Omega_e} \left\{ \left(\lambda_{gxx_j} N_j \right) \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \left(\lambda_{gzz_j} N_j \right) \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} - \left(\frac{\lambda_{gxx_j} N_j}{\rho_{g_j}^*} \right) \left(\rho_{g_j}^* \frac{\partial N_j}{\partial x} \right) N_i \frac{\partial N_j}{\partial x} - \left(\frac{\lambda_{gzz_j} N_j}{\rho_{g_j}^*} \right) \left(\rho_{g_j}^* \frac{\partial N_j}{\partial z} \right) N_i \frac{\partial N_j}{\partial z} \right\} d\Omega_e$$

Evaluating the integrals in (A.7) yields,

$$[B]^e = \frac{\bar{r}_e}{12A_e} \begin{bmatrix} b_{a11} & 0 & b_{a12} & 0 & b_{a13} & 0 \\ b_{a21} & 0 & b_{a22} & 0 & b_{a23} & 0 \\ b_{a31} & 0 & b_{a32} & 0 & b_{a33} & 0 \\ 0 & b_{g11} & 0 & b_{g12} & 0 & b_{g13} \\ 0 & b_{g21} & 0 & b_{g22} & 0 & b_{g23} \\ 0 & b_{g31} & 0 & b_{g32} & 0 & b_{g33} \end{bmatrix} \quad (\text{A.8})$$

where,

$$\begin{aligned} b_{\alpha_{ij}} &= u_{\alpha_{ij}} - w_{\alpha_{ij}} \\ u_{\alpha_{ij}} &= (\lambda_{\alpha_{x1}}^k + \lambda_{\alpha_{x2}}^k + \lambda_{\alpha_{x3}}^k) \beta_i \beta_j + (\lambda_{\alpha_{z1}}^k + \lambda_{\alpha_{z2}}^k + \lambda_{\alpha_{z3}}^k) \gamma_i \gamma_j \\ w_{\alpha_{ij}} &= \frac{\beta_j}{4} (\rho_{\alpha_1}^{*k} \beta_1 + \rho_{\alpha_2}^{*k} \beta_2 + \rho_{\alpha_3}^{*k} \beta_3) \left(\frac{\lambda_{\alpha_{xi}}^k}{\rho_{\alpha_i}^{*k}} + \sum_{j=1}^3 \frac{\lambda_{\alpha_{xj}}^k}{\rho_{\alpha_j}^{*k}} \right) + \\ &\quad \frac{\gamma_j}{4} (\rho_{\alpha_1}^{*k} \gamma_1 + \rho_{\alpha_2}^{*k} \gamma_2 + \rho_{\alpha_3}^{*k} \gamma_3) \left(\frac{\lambda_{\alpha_{zi}}^k}{\rho_{\alpha_i}^{*k}} + \sum_{j=1}^3 \frac{\lambda_{\alpha_{zj}}^k}{\rho_{\alpha_j}^{*k}} \right) \end{aligned}$$

and β_i and γ_i are defined in (3.5). Notice that $u_{\alpha_{ij}} = u_{\alpha_{ji}}$.

The RHS matrix incorporating density gradient terms may be developed as,

$$F_i = \begin{Bmatrix} F_a \\ F_g \end{Bmatrix} \quad (\text{A.9})$$

where,

$$\begin{aligned} F_a &= \sum_{e=1}^{N_e} \int_{\Omega_e} \left\{ g_x (\lambda_{a_{xx}j} N_j) (\rho_{a_j}^* N_j) \frac{\partial N_i}{\partial x} + g_z (\lambda_{a_{zz}j} N_j) (\rho_{a_j}^* N_j) \frac{\partial N_i}{\partial z} - \right. \\ &\quad \left. g_x (\lambda_{a_{xx}j} N_j) \left(\rho_{a_j}^* \frac{\partial N_j}{\partial x} \right) N_i - g_z (\lambda_{a_{zz}j} N_j) \left(\rho_{a_j}^* \frac{\partial N_j}{\partial z} \right) N_i \right\} d\Omega_e \\ F_g &= \sum_{e=1}^{N_e} \int_{\Omega_e} \left\{ g_x (\lambda_{g_{xx}j} N_j) (\rho_{g_j}^* N_j) \frac{\partial N_i}{\partial x} + g_z (\lambda_{g_{zz}j} N_j) (\rho_{g_j}^* N_j) \frac{\partial N_i}{\partial z} - \right. \\ &\quad \left. g_x (\lambda_{g_{xx}j} N_j) \left(\rho_{g_j}^* \frac{\partial N_j}{\partial x} \right) N_i - g_z (\lambda_{g_{zz}j} N_j) \left(\rho_{g_j}^* \frac{\partial N_j}{\partial z} \right) N_i \right\} d\Omega_e \end{aligned}$$

Evaluating these integrals yields,

$$\{F\}^e = \frac{\bar{r}_e}{24} \begin{bmatrix} g_x \left(h_{a_x} \beta_1 - t_{a_{x1}} \right) + g_z \left(h_{a_z} \gamma_1 - t_{a_{z1}} \right) \\ g_x \left(h_{a_x} \beta_2 - t_{a_{x2}} \right) + g_z \left(h_{a_z} \gamma_2 - t_{a_{z2}} \right) \\ g_x \left(h_{a_x} \beta_3 - t_{a_{x3}} \right) + g_z \left(h_{a_z} \gamma_3 - t_{a_{z3}} \right) \\ g_x \left(h_{g_x} \beta_1 - t_{g_{x1}} \right) + g_z \left(h_{g_z} \gamma_1 - t_{g_{z1}} \right) \\ g_x \left(h_{g_x} \beta_2 - t_{g_{x2}} \right) + g_z \left(h_{g_z} \gamma_2 - t_{g_{z2}} \right) \\ g_x \left(h_{g_x} \beta_3 - t_{g_{x3}} \right) + g_z \left(h_{g_z} \gamma_3 - t_{g_{z3}} \right) \end{bmatrix} \quad (\text{A.10})$$

wherein,

$$h_{\alpha x} = \rho_{\alpha 1}^{*k} (2\lambda_{\alpha x_1}^k + \lambda_{\alpha x_2}^k + \lambda_{\alpha x_3}^k) + \rho_{\alpha 2}^{*k} (\lambda_{\alpha x_1}^k + 2\lambda_{\alpha x_2}^k + \lambda_{\alpha x_3}^k) + \rho_{\alpha 3}^{*k} (\lambda_{\alpha x_1}^k + \lambda_{\alpha x_2}^k + 2\lambda_{\alpha x_3}^k)$$

$$h_{\alpha z} = \rho_{\alpha 1}^{*k} (2\lambda_{\alpha z_1}^k + \lambda_{\alpha z_2}^k + \lambda_{\alpha z_3}^k) + \rho_{\alpha 2}^{*k} (\lambda_{\alpha z_1}^k + 2\lambda_{\alpha z_2}^k + \lambda_{\alpha z_3}^k) + \rho_{\alpha 3}^{*k} (\lambda_{\alpha z_1}^k + \lambda_{\alpha z_2}^k + 2\lambda_{\alpha z_3}^k)$$

$$t_{\alpha x_i} = (\rho_{\alpha 1}^{*k} \beta_1 + \rho_{\alpha 2}^{*k} \beta_2 + \rho_{\alpha 3}^{*k} \beta_3) \left(\lambda_{\alpha x_i}^k + \sum_{j=1}^3 \lambda_{\alpha x_j}^k \right)$$

$$t_{\alpha z_i} = (\rho_{\alpha 1}^{*k} \gamma_1 + \rho_{\alpha 2}^{*k} \gamma_2 + \rho_{\alpha 3}^{*k} \gamma_3) \left(\lambda_{\alpha z_i}^k + \sum_{j=1}^3 \lambda_{\alpha z_j}^k \right)$$

The RHS matrix incorporating compositional effects on density and interphase mass exchange is represented by:

$$E_i = \begin{Bmatrix} E_a \\ E_g \end{Bmatrix} \quad (\text{A.11})$$

where,

$$E_a = \sum_{e=1}^{N_e} \phi_e \int_{\Omega_e} \left\{ \left(\frac{1}{\rho_{a_j}^*} \sum_{c,\beta} E_{a\beta c_j}^* - \frac{S_{a_j}}{\rho_{a_j}^*} \frac{\partial \rho_{a_j}^*}{\partial t} \right) N_j N_i \right\} d\Omega_e$$

$$E_g = \sum_{e=1}^{N_e} \phi_e \int_{\Omega_e} \left\{ \left(\frac{1}{\rho_{g_j}^*} \sum_{c,\beta} E_{g\beta c_j}^* - \frac{S_{g_j} P_{g_j}}{\rho_{g_j}^* RT_j} \frac{\partial M_{g_j}}{\partial t} \right) N_j N_i \right\} d\Omega_e$$

Here it has been assumed that $\{E\}$ can be evaluated with compositional information from the previous time step. For the known composition, the terms $\partial \rho_{\alpha}^* / \partial x_{gc}$ are evaluated from the appropriate differentiation of Amagat's Law (2.50) or the Ideal Gas Law (2.49). The temporal change in mole fraction, $\partial x_{\alpha c} / \partial t$, and interphase exchange terms, $E_{\alpha\beta c_j}^*$, are also lagged by a single time step. Evaluating the integrals above, yields,

$$\{E\}^e = \frac{A\phi_e \bar{r}_e}{12} \begin{bmatrix} e_{a_1} \\ e_{a_2} \\ e_{a_3} \\ e_{g_1} \\ e_{g_2} \\ e_{g_3} \end{bmatrix} \quad (\text{A.12})$$

and,

$$e_{a_i} = \frac{1}{\rho_{a_i}^{*k}} \left(\sum_{c,\beta} E_{a\beta c_i}^* - S_{a_i}^k \frac{\partial \rho_{a_i}^*}{\partial t} \right) + \sum_{j=1}^3 \frac{1}{\rho_{a_j}^{*k}} \left(\sum_{c,\beta} E_{a\beta c_j}^* - S_{a_j}^k \frac{\partial \rho_{a_j}^*}{\partial t} \right)$$

$$e_{g_i} = \frac{1}{\rho_{g_i}^{*k}} \left(\sum_{c,\beta} E_{g\beta c_i}^* - \frac{S_{g_i} P_{g_i}}{\rho_{g_i}^* RT_i} \frac{\partial M_{g_i}}{\partial t} \right) + \sum_{j=1}^3 \frac{1}{\rho_{g_j}^{*k}} \left(\sum_{c,\beta} E_{g\beta c_j}^* - \frac{S_{g_j} P_{g_j}}{\rho_{g_j}^* RT_j} \frac{\partial M_{g_j}}{\partial t} \right)$$

The RHS vector $\{Q\}^e$ represents the phase sources and sinks,

$$Q_i = \begin{Bmatrix} Q_{a_i} \\ Q_{g_i} \end{Bmatrix} \quad (\text{A.13})$$

where Q_{α_i} is the prescribed extraction or injection rate of phase α at node i [L^3/T] at the aquifer temperature and pressure. Q_{α} is negative for extraction. This vector is used to incorporate fluxes at simulated injection or extraction wells.

Appendix B

ELEMENT MATRICES FOR THE SOLUTION OF DARCY'S LAW EQUATION

The element matrix equation for the solution of the Darcy velocities is derived from (3.23) and (3.24). For purposes of conciseness only the expansion of the aqueous phase equation (3.23) will be shown here,

$$[A]^e \{q_{\alpha_x}\}^e = \{F\}^e \quad (\text{B.1a})$$

$$[A]^e \{q_{\alpha_z}\}^e = \{F_{\alpha}\}^e \quad (\text{B.1b})$$

where $\{x_{\alpha_x}\}^e$ and $\{x_{\alpha_z}\}^e$ are the vectors of x and z direction Darcy velocities at the three nodes of the element e ,

$$\{q_{\alpha_x}\}^e = \{q_{a_1}, q_{a_2}, q_{a_3}\}_{\alpha_x} \quad (\text{B.2a})$$

$$\{q_{\alpha_z}\}^e = \{q_{a_1}, q_{a_2}, q_{a_3}\}_{\alpha_z} \quad (\text{B.2b})$$

The $[A]^e$ matrix developed from eq. (3.23) may be expressed as,

$$A_{ij} = \int_{\Omega^e} N_j N_i d\Omega^e \quad (\text{B.3})$$

where in local coordinates the indices i and j can equal 1, 2, or 3. This matrix is the same for both directions, x , and z . In this development r can be substituted for x when the domain is axisymmetric. After evaluation of the integrals, $[A]^e$ is,

$$[A]^e = \frac{A^e \bar{r}^e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad (\text{B.4})$$

Note that $[A]^e$ is constant for a given domain and need be factorized only once during a simulation.

The RHS vector from (3.23) where l is used to represent the directions, x or z is,

$$F_{a_l}^e = - \int_{\Omega^e} \lambda_{a_{ll}} N_j \left(P_{a_j} \frac{\partial N_j}{\partial l} - \rho_{a_j}^* g_l N_j \right) N_i d\Omega^e \quad (\text{B.5})$$

Evaluating the integrals in B.5 yields,

$$\{F_{a_l}\}^e = \{F_{a_l}\}_{P_a}^e + \{F_{a_l}\}_g^e \quad (\text{B.6})$$

where the pressure gradient term, $\{F_{a_l}\}_{P_a}^e$ is,

$$\{F_{a_l}\}_{P_a}^e = - \frac{\bar{r}^e}{24} \begin{bmatrix} \left(2\lambda_{a_{ll_1}} + \lambda_{a_{ll_2}} + \lambda_{a_{ll_3}} \right) \left(\sum_e \left(P_{a_j} \frac{\partial N_j}{\partial l} \right) \right) \\ \left(\lambda_{a_{ll_1}} + 2\lambda_{a_{ll_2}} + \lambda_{a_{ll_3}} \right) \left(\sum_e \left(P_{a_j} \frac{\partial N_j}{\partial l} \right) \right) \\ \left(\lambda_{a_{ll_1}} + \lambda_{a_{ll_2}} + 2\lambda_{a_{ll_3}} \right) \left(\sum_e \left(P_{a_j} \frac{\partial N_j}{\partial l} \right) \right) \end{bmatrix} \quad (\text{B.7})$$

and,

$$\sum_e \left(P_{aj} \frac{\partial N_j}{\partial x} \right) = P_{a1} \beta_1 + P_{a2} \beta_2 + P_{a3} \beta_3 \quad (\text{B.8a})$$

$$\sum_e \left(P_{aj} \frac{\partial N_j}{\partial z} \right) = P_{a1} \gamma_1 + P_{a2} \gamma_2 + P_{a3} \gamma_3 \quad (\text{B.8b})$$

The variables β_j and γ_j are the local coordinate derivatives of the basis functions at node j in the x and z directions, respectively.

Finally the gravity term, $\{F_{a1}\}_g^e$, is expressed as,

$$\{F_{a1}\}_g^e = \quad (\text{B.9})$$

$$\frac{A^e \bar{\gamma}_e g l}{60} \left[\begin{array}{l} \left(6\lambda_{a11} + 2\lambda_{a12} + 2\lambda_{a13} \right) \rho_{a1} + \left(2\lambda_{a11} + 2\lambda_{a12} + \lambda_{a13} \right) \rho_{a2} + \left(2\lambda_{a11} + 2\lambda_{a12} + \lambda_{a13} \right) \rho_{a3} \\ \left(2\lambda_{a11} + 2\lambda_{a12} + \lambda_{a13} \right) \rho_{a1} + \left(2\lambda_{a11} + 6\lambda_{a12} + 2\lambda_{a13} \right) \rho_{a2} + \left(\lambda_{a11} + 2\lambda_{a12} + 2\lambda_{a13} \right) \rho_{a3} \\ \left(2\lambda_{a11} + \lambda_{a12} + 2\lambda_{a13} \right) \rho_{a1} + \left(\lambda_{a11} + 2\lambda_{a12} + 2\lambda_{a13} \right) \rho_{a2} + \left(2\lambda_{a11} + 2\lambda_{a12} + 6\lambda_{a13} \right) \rho_{a3} \end{array} \right]$$

Appendix C

ELEMENT MATRICES FOR THE SEQUENTIAL SOLUTION OF THE COMPONENT MOLE BALANCE EQUATIONS

Integration of (3.36) gives the element matrix equation,

$$[A]^e \frac{\partial \{x_{\alpha_c}\}^e}{\partial t} + [B]^e \{x_{\alpha_c}\}^e = \{F\}^e \quad (C.1)$$

where $\{x_{\alpha_c}\}^e$ is the vector of component mole fractions at the three nodes of the element e (1, 2, 3),

$$\{x_{\alpha_c}\}^e = \{x_{\alpha_1}, x_{\alpha_2}, x_{\alpha_3}\}_{\alpha_c}^T \quad (C.2)$$

The mass matrix is developed from eq. (3.37). It may be expressed as,

$$A_{ij} = \int_{\Omega^e} \phi^e S_{\alpha_j} \rho_{\alpha_j} r_{\alpha_c}^e N_j \frac{\partial x_{\alpha_c j}}{\partial t} N_j N_i d\Omega^e \quad (C.3)$$

where in local coordinates the indices i and j can equal 1, 2, or 3. Evaluating the integrals, the element mass matrix $[A]^e$ is,

$$[A]^e = \frac{\phi^e r_{\alpha_c} A^e \bar{r}^e}{30\Delta t} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad (C.4)$$

where the individual elements of $[A]^e$ are defined as follows,

$$a_{11} = 3S_{\alpha_1}\rho_{\alpha_1} + S_{\alpha_2}\rho_{\alpha_2} + S_{\alpha_3}\rho_{\alpha_3} \quad (C.5a)$$

$$a_{12} = a_{21} = S_{\alpha_1}\rho_{\alpha_1} + S_{\alpha_2}\rho_{\alpha_2} + 0.5S_{\alpha_3}\rho_{\alpha_3} \quad (C.5b)$$

$$a_{13} = a_{31} = S_{\alpha_1}\rho_{\alpha_1} + 0.5S_{\alpha_2}\rho_{\alpha_2} + S_{\alpha_3}\rho_{\alpha_3} \quad (C.5c)$$

$$a_{22} = S_{\alpha_1}\rho_{\alpha_1} + 3S_{\alpha_2}\rho_{\alpha_2} + S_{\alpha_3}\rho_{\alpha_3} \quad (C.5d)$$

$$a_{23} = a_{32} = 0.5S_{\alpha_1}\rho_{\alpha_1} + S_{\alpha_2}\rho_{\alpha_2} + S_{\alpha_3}\rho_{\alpha_3} \quad (C.5e)$$

$$a_{33} = S_{\alpha_1}\rho_{\alpha_1} + S_{\alpha_2}\rho_{\alpha_2} + 3S_{\alpha_3}\rho_{\alpha_3} \quad (C.5f)$$

An option is included in MISER for mass lumping of $[A]^e$ giving,

$$[A^L]^e = \frac{\phi^e r_{\alpha_c} A^e \bar{r}^e}{30\Delta t} \begin{bmatrix} a_{11} + a_{12} + a_{13} & 0 & 0 \\ 0 & a_{21} + a_{22} + a_{23} & 0 \\ 0 & 0 & a_{31} + a_{32} + a_{33} \end{bmatrix} \quad (C.6)$$

The stiffness matrix developed from eqs. (3.36) may be expressed as,

$$B_{ij}^1 = \int_{\Omega^e} \rho_{\alpha_c} \left\{ q_{\alpha_x}^e x_{\alpha_c j} \frac{\partial N_i}{\partial x} + q_{\alpha_z}^e x_{\alpha_c j} \frac{\partial N_i}{\partial z} \right\} N_i d\Omega^e \quad (C.7)$$

$$B_{ij}^2 = \int_{\Omega^e} \phi^e \left\{ S_{\alpha_j} \rho_{\alpha_j} N_j \left[D_{\alpha_{c_{xx}}}^{h^e} x_{\alpha_c j} \frac{\partial N_j}{\partial x} + D_{\alpha_{c_{xz}}}^{h^e} x_{\alpha_c j} \frac{\partial N_j}{\partial z} \right] \frac{\partial N_i}{\partial x} + \right. \quad (C.8)$$

$$\left. \left[D_{\alpha_{c_{xx}}}^{h^e} x_{\alpha_c j} \frac{\partial N_j}{\partial x} + D_{\alpha_{c_{xz}}}^{h^e} x_{\alpha_c j} \frac{\partial N_j}{\partial z} \right] \frac{\partial N_i}{\partial z} \right\} d\Omega^e \quad (C.9)$$

$$B_{ij}^3 = \int_{\Omega^e} \phi^e \left\{ \bar{K}_{\alpha_c j} N_j x_{\alpha_c j} N_j \right\} N_i d\Omega^e \quad (C.10)$$

Evaluating the integrals, the element stiffness matrix $[B]^e$ is,

$$[B]^e = [B^1]^e + [B^2]^e + [B^3] \quad (C.11)$$

where the terms $[B^1]^e$, $[B^2]^e$, and $[B^3]$ are defined as follows. First the advective portion of the stiffness matrix can be expressed as,

$$[B^1]^e = \frac{\bar{r}^e}{24} \begin{bmatrix} b_{11}^1 & b_{12}^1 & b_{13}^1 \\ b_{21}^1 & b_{22}^1 & b_{23}^1 \\ b_{31}^1 & b_{32}^1 & b_{33}^1 \end{bmatrix} \quad (C.12)$$

with the individual terms defined as,

$$b_{11}^1 = \left\{ 2\rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_1 + \left\{ 2\rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_1 \quad (C.13a)$$

$$b_{12}^1 = \left\{ 2\rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_2 + \left\{ 2\rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_2 \quad (C.13b)$$

$$b_{13}^1 = \left\{ 2\rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_3 + \left\{ 2\rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_3 \quad (C.13c)$$

$$b_{21}^1 = \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + 2\rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_1 + \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + 2\rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_1 \quad (C.13d)$$

$$b_{22}^1 = \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + 2\rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_2 + \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + 2\rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_2 \quad (C.13e)$$

$$b_{23}^1 = \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + 2\rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_3 + \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + 2\rho_{\alpha_2} q_{\alpha_{x_2}} + \rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_3 \quad (C.13f)$$

$$b_{31}^1 = \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + 2\rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_1 + \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + 2\rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_1 \quad (C.13g)$$

$$b_{32}^1 = \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + 2\rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_2 + \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + 2\rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_2 \quad (C.13h)$$

$$b_{33}^1 = \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + 2\rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \beta_3 + \left\{ \rho_{\alpha_1} q_{\alpha_{x_1}} + \rho_{\alpha_2} q_{\alpha_{x_2}} + 2\rho_{\alpha_3} q_{\alpha_{x_3}} \right\} \gamma_3 \quad (C.13i)$$

Next the dispersive portion of the stiffness matrix is,

$$[B^2]^e = \frac{\bar{r}^e \{ S_{\alpha_1} \rho_{\alpha_1} + S_{\alpha_2} \rho_{\alpha_2} + S_{\alpha_3} \rho_{\alpha_3} \}}{12A^e} \begin{bmatrix} b_{11}^2 & b_{12}^2 & b_{13}^2 \\ b_{21}^2 & b_{22}^2 & b_{23}^2 \\ b_{31}^2 & b_{32}^2 & b_{33}^2 \end{bmatrix} \quad (C.14)$$

where the individual terms are defined as,

$$b_{11}^2 = \beta_1 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_1 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_1 \right\} + \gamma_1 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_1 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_1 \right\} \quad (C.15a)$$

$$b_{12}^2 = \beta_2 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_1 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_1 \right\} + \gamma_2 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_1 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_1 \right\} \quad (C.15b)$$

$$b_{13}^2 = \beta_3 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_1 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_1 \right\} + \gamma_3 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_1 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_1 \right\} \quad (C.15c)$$

$$b_{21}^2 = \beta_1 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_2 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_2 \right\} + \gamma_1 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_2 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_2 \right\} \quad (C.15d)$$

$$b_{22}^2 = \beta_2 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_2 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_2 \right\} + \gamma_2 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_2 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_2 \right\} \quad (C.15e)$$

$$b_{23}^2 = \beta_3 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_2 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_2 \right\} + \gamma_3 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_2 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_2 \right\} \quad (C.15f)$$

$$b_{31}^2 = \beta_1 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_3 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_3 \right\} + \gamma_1 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_3 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_3 \right\} \quad (C.15g)$$

$$b_{32}^2 = \beta_2 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_3 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_3 \right\} + \gamma_2 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_3 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_3 \right\} \quad (C.15h)$$

$$b_{33}^2 = \beta_3 \left\{ D_{\alpha_{c_{xx}}}^{h^e} \beta_3 + D_{\alpha_{c_{xz}}}^{h^e} \gamma_3 \right\} + \gamma_3 \left\{ D_{\alpha_{c_{zx}}}^{h^e} \beta_3 + D_{\alpha_{c_{zz}}}^{h^e} \gamma_3 \right\} \quad (C.15i)$$

Finally, the exchange and reaction portion of the stiffness matrix is defined as,

$$[B^3]^e = \frac{\phi^e A^e \bar{r}^e}{30} \begin{bmatrix} b_{11}^3 & b_{12}^3 & b_{13}^3 \\ b_{21}^3 & b_{22}^3 & b_{23}^3 \\ b_{31}^3 & b_{32}^3 & b_{33}^3 \end{bmatrix} \quad (C.16)$$

where the individual terms are defined as,

$$b_{11}^3 = 3\bar{K}_{\alpha_{c_1}} + \bar{K}_{\alpha_{c_2}} + \bar{K}_{\alpha_{c_3}} \quad (C.17a)$$

$$b_{12}^3 = b_{21}^3 = \bar{K}_{\alpha_{c_1}} + \bar{K}_{\alpha_{c_2}} + 0.5\bar{K}_{\alpha_{c_3}} \quad (C.17b)$$

$$b_{13}^3 = b_{31}^3 = \bar{K}_{\alpha_{c_1}} + 0.5\bar{K}_{\alpha_{c_2}} + \bar{K}_{\alpha_{c_3}} \quad (C.17c)$$

$$b_{22}^3 = \bar{K}_{\alpha_{c_1}} + 3\bar{K}_{\alpha_{c_2}} + \bar{K}_{\alpha_{c_3}} \quad (C.17d)$$

$$b_{23}^3 = b_{32}^3 = 0.5\bar{K}_{\alpha_{c_1}} + \bar{K}_{\alpha_{c_2}} + \bar{K}_{\alpha_{c_3}} \quad (C.17e)$$

$$b_{33}^3 = \bar{K}_{\alpha_{c_1}} + \bar{K}_{\alpha_{c_2}} + 3\bar{K}_{\alpha_{c_3}} \quad (C.17f)$$

and the terms $\bar{K}_{\alpha_{c_i}}$ are defined in Table 3.2.

The RHS vector developed from integration of (3.36) is,

$$F_{\alpha_c}^e = \int_{\Omega^e} \phi^e \bar{F}_{\alpha_{c_j}} N_j N_i d\Omega^e + \int_{\Gamma^e} \left\{ \phi^e S_{\alpha_j} \rho_{\alpha_j} N_j \left[D_{\alpha_{c_{xx}}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial x} \right. \right. \\ \left. \left. + D_{\alpha_{c_{xz}}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial z} + D_{\alpha_{c_{zx}}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial x} + D_{\alpha_{c_{zz}}}^{h^e} x_{\alpha_{c_j}} \frac{\partial N_j}{\partial z} \right] \right\} n N_i d\Gamma^e$$

The terms \bar{F}_{α_j} are defined in Table 3.2. Upon assembly of the global RHS vector, the boundary integral terms sum to zero except at the domain boundaries (see Section 3.11.2). Evaluating the remaining portion of the integral which contains the exchange and reaction term, \bar{F}_{α_j} , yields,

$$\{F\}^e = \frac{\phi^e A^e \bar{r}_e}{12} \begin{bmatrix} 2\bar{F}_{\alpha_{c_1}} + \bar{F}_{\alpha_{c_2}} + \bar{F}_{\alpha_{c_3}} \\ \bar{F}_{\alpha_{c_1}} + 2\bar{F}_{\alpha_{c_2}} + \bar{F}_{\alpha_{c_3}} \\ \bar{F}_{\alpha_{c_1}} + \bar{F}_{\alpha_{c_2}} + 2\bar{F}_{\alpha_{c_3}} \end{bmatrix} \quad (\text{C.18})$$

The development above applies to the mobile phases $\alpha = g, a$. A similar development starting with (3.39) for the organic liquid and (3.41) for the biophase leads to the same terms presented above except that (C.12) and (C.14) are both omitted.

Since significant differences exist for the solid phase development, only the mass matrix and RHS will be shown below. Integrating (3.40), and assuming that the solid phase bulk density is constant yields for the mass matrix,

$$[A]^e = \frac{\rho_s^* A^e \bar{r}_e}{12\Delta t} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad (\text{C.19})$$

and for the RHS,

$$\{F\}^e = \frac{A^e \bar{r}_e M_c}{12} \begin{bmatrix} 2\bar{F}_{s_{c_1}} + \bar{F}_{s_{c_2}} + \bar{F}_{s_{c_3}} \\ \bar{F}_{s_{c_1}} + 2\bar{F}_{s_{c_2}} + \bar{F}_{s_{c_3}} \\ \bar{F}_{s_{c_1}} + \bar{F}_{s_{c_2}} + 2\bar{F}_{s_{c_3}} \end{bmatrix} \quad (\text{C.20})$$

Appendix D

ELEMENT MATRICES FOR THE SOLUTION OF THE ORGANIC PHASE MASS BALANCE EQUATION

The governing matrix equation for the solution of the organic phase mass balance is derived from equation (3.47). Integration of (3.47) gives the element matrix equation,

$$[A]^e \frac{\partial \{S_o\}^e}{\partial t} = \{F\}^e \quad (D.1)$$

where $\{S_o\}^e$ is the vector of NAPL saturations at the three nodes of the element,

$$\{S_o\}^e = \{S_{o1}, S_{o2}, S_{o3}\}_{\alpha_c} \quad (D.2)$$

The mass matrix is developed from eq. (3.47). It may be expressed as,

$$A_{ij} = \int_{\Omega^e} \left\{ \phi^e \rho_{o_j}^* N_j \frac{\partial S_{o_j}}{\partial t} N_j \right\} N_i d\Omega^e \quad (D.3)$$

where in local coordinates the indices i and j can equal 1, 2, or 3. Evaluating the integrals, the element mass matrix $[A]^e$ is,

$$[A]^e = \frac{\phi^e A^e \bar{r}^e}{30\Delta t} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad (D.4)$$

where the individual elements of $[A]^e$ are defined as follows,

$$a_{11} = 3\rho_{o1}^* + \rho_{o2}^* + \rho_{o3}^* \quad (D.5a)$$

$$a_{12} = a_{21} = \rho_{o1}^* + \rho_{o2}^* + 0.5\rho_{o3}^* \quad (D.5b)$$

$$a_{13} = a_{31} = \rho_{o1}^* + 0.5\rho_{o2}^* + \rho_{o3}^* \quad (D.5c)$$

$$a_{22} = \rho_{o1}^* + 3\rho_{o2}^* + \rho_{o3}^* \quad (D.5d)$$

$$a_{23} = a_{32} = 0.5\rho_{o1}^* + \rho_{o2}^* + \rho_{o3}^* \quad (D.5e)$$

$$a_{33} = \rho_{o1}^* + \rho_{o2}^* + 3\rho_{o3}^* \quad (D.5f)$$

$[A]$ is then "mass lumped" to facilitate solution of the stacked NAPL saturation giving,

$$[A^L]^e = \frac{\phi^e A^e \bar{r}^e}{30\Delta t} \begin{bmatrix} a_{11} + a_{12} + a_{13} & 0 & 0 \\ 0 & a_{21} + a_{22} + a_{23} & 0 \\ 0 & 0 & a_{31} + a_{32} + a_{33} \end{bmatrix} \quad (D.6)$$

The RHS vector developed from integration of (3.47) is,

$$F_{\alpha_c}^e = \int_{\Omega^e} \phi \left\{ E_{o_j}^* N_j - S_{o_j} \frac{\partial \rho_{o_j}^*}{\partial t} N_j \right\} N_i d\Omega^e \quad (D.7)$$

where $E_{o_j}^*$ is defined in Table 3.1. Evaluating (D.7) gives,

$$\{F\}^e = \frac{\phi^e A^e \bar{r}_e}{12} \begin{bmatrix} 2E_{o_1}^* - S_{o_1}^k \frac{\partial \rho_{o_1}^*}{\partial t} + E_{o_2}^* - S_{o_2}^k \frac{\partial \rho_{o_2}^*}{\partial t} + E_{o_3}^* - S_{o_3}^k \frac{\partial \rho_{o_3}^*}{\partial t} \\ E_{o_1}^* - S_{o_1}^k \frac{\partial \rho_{o_1}^*}{\partial t} + 2E_{o_2}^* - S_{o_2}^k \frac{\partial \rho_{o_2}^*}{\partial t} + E_{o_3}^* - S_{o_3}^k \frac{\partial \rho_{o_3}^*}{\partial t} \\ E_{o_1}^* - S_{o_1}^k \frac{\partial \rho_{o_1}^*}{\partial t} + E_{o_2}^* - S_{o_2}^k \frac{\partial \rho_{o_2}^*}{\partial t} + 2E_{o_3}^* - S_{o_3}^k \frac{\partial \rho_{o_3}^*}{\partial t} \end{bmatrix} \quad (D.8)$$

This allows a direct solution for $\{S_o\}^{k+1}$ after assembly of the global matrices as,

$$S_{o_i}^{k+1} = F_i / A_{ii} \quad (D.9)$$

where $A_{ii} \neq 0$ and i is iterated over the total number of stacked variables.

Appendix E

Description of Major Variables

Integer Scalars

ia	Number of nonzero entries in sparse matrix.
nnhor	Maximum number of nodes in the horizontal direction in a generated grid.
nnver	Maximum number of nodes in the vertical direction in a generated grid.

Real Scalars

b	Klinkenberg parameter; Table 5.13: <i>Field 2</i> .
caplen	Radius of impermeable cap on the ground surface.
kd	Decay coefficient; Table 5.12: <i>Field 5</i> .
qwll	Extraction/injection rate; Table 5.22: <i>Field 2</i> .
rwell	Well radius; Table 5.22: <i>Field 3</i> .
tmass0	Initial total mass in domain.
tmass1	Current time step total mass in domain.
trefqg	Reference temperature.
wtdpth	Depth to water table.
wvis	Water viscosity; Table 5.13: <i>Field 1</i> .
xbmax	Maximum biomass; Table 5.12: <i>Field 5</i> .
xbmin	Minimum biomass; Table 5.12: <i>Field 5</i> .
xbok	Two compartment K_f multiplier; Table 5.11: <i>Field 3</i> .
xbom	Two compartment m multiplier; Table 5.11: <i>Field 3</i> .
xden	Two compartment density multiplier; Table 5.11: <i>Field 3</i> .
xinit	Initial biomass; Table 5.12: <i>Field 5</i> .
xkex	Two compartment exchange coefficient multiplier; Table 5.11: <i>Field 3</i> .

zwell Length of well screen.

Character Scalars

outpre Output file prefix; Table 5.4: *Field 2*.

Integer Arrays

ibc(nnm _x)	Nodal boundary condition locations; Table 5.21: <i>Field 1</i> (gas pressures), 2 (gas pressures), 3 (aqueous pressures), 4 (aqueous pressures), 5 (gas phase components), 6 (aqueous phase components), 7 (gas phase boundary fluxes), and 8 (aqueous phase boundary fluxes).
ibcxmf(nm _b c)	Nodal component boundary condition types; Table 5.21: <i>Field 5</i> (gas phase components) and 6 (aqueous phase components).
idepth(nnm _x)	Nodal depth reference.
icn(icn _l)	Sparse matrix column indices.
icp(0:50)	Vector of control integers for component identification.
ikeep(icn _l ,5)	Solver work space.
ipt(0:90)	Vector of control integers.
ipt(0)	Number of elements; Table 5.7: <i>Field 7</i> .
ipt(1)	Number of nodes; Table 5.7: <i>Field 7</i> .
ipt(2)	Number of stacked variables.
ipt(3)	Number of components in the gas phase.
ipt(4)	Number of components in the aqueous phase.
ipt(5)	Number of components in the NAPL phase.

ipt(6)	Number of components in the solid phase.	ipt(28)	Device designator for performance output; Table 5.4: <i>Field 3</i> .
ipt(7)	Number of components in the bio phase (includes biomass).	ipt(29)	Device designator for error messages; Table 5.4: <i>Field 3</i> .
ipt(8)	Start of gas phase section in xmf ordering.	ipt(30)	Maximum number of time steps; Table 5.6: <i>Field 3</i> .
ipt(9)	Start of aqueous phase section in xmf ordering.	ipt(31)	Maximum phase balance iterations, also used as the criterion for decreasing dt in phase balance; Table 5.6: <i>Field 6</i> .
ipt(10)	Start of NAPL phase section in xmf ordering.	ipt(32)	Maximum component balance iterations, also used as the criterion for decreasing dt in component balance routines; Table 5.6: <i>Field 6</i> .
ipt(11)	Start of solid phase section in xmf ordering.	ipt(33)	Maximum NAPL saturation iterations; Table 5.6: <i>Field 6</i> .
ipt(12)	Start of bio phase section in xmf ordering.	ipt(34)	Maximum number of iterations in phase balance for increasing dt; Table 5.6: <i>Field 7</i> .
ipt(13)	Number of organic components in the gas phase.	ipt(35)	Maximum number of iterations in component balance routines for increasing dt; Table 5.6: <i>Field 7</i> .
ipt(14)	Number of organic components in the aqueous phase.	ipt(36)	Flag from flow.f for time step modification.
ipt(15)	Number of organic components in the NAPL phase; Table 5.8: <i>Field 1</i> .	ipt(37)	Flag from tran.f for time step modification.
ipt(16)	Number of organic components in the solid phase.	ipt(38)	Flag from napl.f for time step modification.
ipt(17)	Number of organic components in the bio phase; Table 5.12: <i>Field 1</i> .	ipt(39)	Integer flag determining the biokinetics type (1 - standard monod kinetics; 2 - monod kinetics with substrate inhibition; 3 - monod kinetics with lumped substrate inhibition; 4 - monod kinetics with saturation dependency; 5 - monod kinetics with saturation dependency and substrate inhibition); Table 5.12: <i>Field 3</i> .
ipt(18)	Number of nodes with constant gas pressure; Table 5.21: <i>Field 2</i> .	ipt(40)	Constant - ipt(1) * 2.
ipt(19)	Number of nodes with constant aqueous pressure; Table 5.21: <i>Field 4</i> .	ipt(41)	Constant - ipt(1) * 3.
ipt(20)	Number of nodes with gas phase component boundary conditions; Table 5.21: <i>Field 5</i> .	ipt(42)	Constant - ipt(1) * 4.
ipt(21)	Number of nodes with aqueous phase component boundary conditions; Table 5.21: <i>Field 6</i> .	ipt(43)	Constant - ipt(1) * 5.
ipt(22)	Number of nodes with constant gas volumetric flux; Table 5.21: <i>Field 7</i> .	ipt(44)	Constant - ipt(1) * 6.
ipt(23)	Number of nodes with constant aqueous volumetric flux; Table 5.21: <i>Field 8</i> .	ipt(45)	Constant - ipt(1) * 7.
ipt(24)	Number of nodes along the well screen.	ipt(46)	Constant - ipt(1) * 8.
ipt(25)	Print results every ipt(25) time steps if lprnt(0) is true; Table 5.4: <i>Field 8</i> .	ipt(47)	Constant - ipt(1) * 9.
ipt(26)	Number of material property blocks; Table 5.7: <i>Field 6</i> .	ipt(48)	Constant - ipt(1) * 10.
ipt(27)	Integer variable denoting the type of domain (ipt(27) = 0 - xz domain; ipt(27) = 1 - rz domain); Table 5.5: <i>Field 1</i> .	ipt(49)	Constant - ipt(2) * 2.
		ipt(50)	Constant - ipt(2) * 3.

ipt(51)	Constant - $\text{ipt}(2) * 4$.	ipt(83)	Print mass balance every ipt(83) time steps if lprnt(25) is true; Table 5.4: <i>Field 5</i> .
ipt(52)	Constant - $\text{ipt}(2) * 5$.	ipt(84)	Print time series every ipt(84) time steps if lprnt(26) is true; Table 5.4: <i>Field 6</i> .
ipt(53)	Constant - $\text{ipt}(2) * 6$.	ipt(85)	Compute flow field every ipt(85) time steps; Table 5.5: <i>Field 4</i> .
ipt(54)	Constant - $\text{ipt}(2) * 7$.	ipt(86)	Number of nodes in the vertical direction.
ipt(55)	Constant - $\text{ipt}(2) * 8$.	ipt(87)	Number of nodes in the horizontal direction.
ipt(56)	Constant - $\text{ipt}(2) * 9$.	ipt(88)	Pointer for temperature dependencies.
ipt(57)	Constant - $\text{ipt}(2) * 10$.	ipt(89)	Constant - $5 * \text{ipt}(88)$.
ipt(58)	Constant - $\text{ipt}(3) + \text{ipt}(4)$.	irn(icnl)	Sparse matrix row indices.
ipt(59)	Constant - $\text{ipt}(3) + \text{ipt}(4) + \text{ipt}(5)$.	iw(icnl,8)	Solver work space.
ipt(60)	Constant - $\text{ipt}(3) + \text{ipt}(4) + \text{ipt}(5) + \text{ipt}(6)$.	matel(nelmx)	Element material blocks; Table 5.7: <i>Field 8</i> .
ipt(61)	Constant - $\text{ipt}(3) + \text{ipt}(4) + \text{ipt}(5) + \text{ipt}(6) + \text{ipt}(7)$.	matpt(nn6)	Nodal material blocks.
ipt(62)	Constant - $\text{ipt}(18) + \text{ipt}(19) + \text{ipt}(20) + \text{ipt}(21)$.	nbdB(nxmax)	Node numbers along the bottom boundary of a generated rectangular domain.
ipt(63)	Constant - $\text{ipt}(62) + \text{ipt}(22)$.	nbdL(nzmax)	Node numbers along the left boundary of a generated rectangular domain.
ipt(64)	Constant - $\text{ipt}(63) + \text{ipt}(23)$.	nbdR(nzmax)	Node numbers along the right boundary of a generated rectangular domain.
ipt(65)	Number of components present.	nbdT(nxmax)	Node numbers along the top boundary of a generated rectangular domain.
ipt(66)	Constant - $\text{ipt}(65) * \text{ipt}(1)$.	nbw(0:2)	Sparse matrix band widths.
ipt(67)	Constant - $\text{ipt}(0) * 2$.	nelpt(nel3)	Element stacking references.
ipt(68)	Constant - $\text{ipt}(0) * 3$.	nodeL(nel3)	Element connectivity vector; Table 5.7: <i>Field 8</i> .
ipt(69)	Number of gas phase components in output; Table 5.15: <i>Field 2</i> .	nodept(nnmX)	Nodal stacking references.
ipt(70)	Number of aqueous phase components in output; Table 5.15: <i>Field 2</i> .	Real Arrays	
ipt(71)	Number of NAPL components in output; Table 5.15: <i>Field 2</i> .	a(icnl)	Global FEM matrix.
ipt(72)	Number of solid phase components in output; Table 5.15: <i>Field 2</i> .	aby12(nelmx)	Element; $\text{rbar} * \text{area} / 12$.
ipt(73)	Number of bio phase components in output; Table 5.15: <i>Field 2</i> .	aby30(nelmx)	Element; $\text{rbar} * \text{area} / 30$.
ipt(74)	Constant - $\text{ipt}(69) + \text{ipt}(70) + \text{ipt}(71) + \text{ipt}(72) + \text{ipt}(73)$.	amb(icnl)	Global FEM matrix for phase mass balance.
ipt(75)	Initial condition type; Table 5.17: <i>Field 1</i> .	area(nelmx)	Element areas.
ipt(76)	Restart input value of current time step number.	bcf(nn2)	Nodal phase boundary fluxes.
ipt(77)	Not Used.		
ipt(78)	Not Used.		
ipt(79)	Not Used.		
ipt(80)	Current number of time steps.		
ipt(81)	Number of time series plot gas phase components; Table 5.15: <i>Field 3</i> .		
ipt(82)	Number of time series plot aqueous phase components; Table 5.15: <i>Field 4</i> .		

bexmf(nmbc)	Nodal component boundary condition in contacting fluid; Table 5.21: <i>Field 5</i> (gas phase components) and <i>6</i> (aqueous phase components).	cmdiff(ncmp2)	Component gas and aqueous phase molecular diffusivities; Table 5.8: <i>Field 2</i> (organic components), <i>3</i> (water, oxygen, and nitrogen), and <i>4</i> (nutrient).
bdisl(nmblk)	Block longitudinal dispersivities; Table 5.10: <i>Field 3</i> .	cmf(ncmpp5)	Phase and component cumulative total boundary fluxes.
bdist(nmblk)	Block transverse dispersivities; Table 5.10: <i>Field 3</i> .	cmw(ncmp)	Component molecular weights; Table 5.8: <i>Field 2</i> (organic components), <i>3</i> (water, oxygen, and nitrogen), and <i>4</i> (nutrient).
beta(nel3)	Nodal basis function x -derivatives.	cphex(ncmpp5)	Phase and component cumulative exchange fluxes.
bfoc(nmblk)	Block organic carbon contents; Table 5.10: <i>Field 1</i> .	crsink(ncmpp5)	Phase and component cumulative reaction sinks.
bok(nbcmp)	Block component Freundlich K_f parameters; Table 5.11: <i>Field 2</i> .	csflux(ncmpp5)	Cumulative surface flux.
bom(nbcmp)	Block component Freundlich m parameters; Table 5.11: <i>Field 2</i> .	csink(ncmpp5)	Phase and component cumulative sinks.
bpermh(nmblk)	Block horizontal intrinsic permeability; Table 5.10: <i>Field 1</i> .	cvp(ncmp)	Component vapor pressures; Table 5.8: <i>Field 2</i> (organic components), <i>3</i> (water, oxygen, and nitrogen), and <i>4</i> (nutrient).
bpermv(nmblk)	Block vertical intrinsic permeability; Table 5.10: <i>Field 1</i> .	cvvis(ncmp)	Component vapor viscosity; Table 5.8: <i>Field 2</i> (organic components), <i>3</i> (water, oxygen, and nitrogen), and <i>4</i> (nutrient).
bphi(nmblk)	Block porosity; Table 5.10: <i>Field 1</i> .	cwsink(ncmpp5)	Phase and component cumulative well sinks.
bsden(nmblk)	Block bulk soil densities; Table 5.10: <i>Field 1</i> .	d(nmd)	Element dispersivities; Table 5.10: <i>Field 3</i> .
bvga(nmblk)	Block van Genuchten α parameters; Table 5.10: <i>Field 2</i> .	dden(nn6)	Nodal gas, aqueous, and NAPL phase mole and mass density derivatives.
bvgm(nmblk)	Block van Genuchten $1 - \frac{1}{n}$.	den(nn6)	Nodal gas, aqueous, and NAPL phase current time step mole and mass densities.
bvgn(nmblk)	Block van Genuchten n parameters; Table 5.10: <i>Field 2</i> .	dent(nn6)	Nodal gas, aqueous, and NAPL phase previous time step mole and mass densities.
bsrw(nmblk)	Block residual aqueous phase saturations; Table 5.10: <i>Field 2</i> .	den0(nnmx)	Initial gas phase mass density.
cc(nnstk)	Nodal capacity coefficients.	dfxmf(nmbc)	Nodal component boundary diffusive flux; Table 5.21: <i>Field 5</i> (gas phase components) and <i>6</i> (aqueous phase components).
casol(ncmp)	Component aqueous solubility; Table 5.8: <i>Field 2</i> (organic components), <i>3</i> (water, oxygen, and nitrogen), and <i>4</i> (nutrient).	dtemp(nzmax6)	Depth temperature corrections; Table 5.14: <i>Field 4</i> (component vapor pressures), <i>5</i> (component vapor viscosity), <i>6</i> (component Henry's Law constants), <i>7</i> (component aqueous solubility), <i>8</i> (component maximum substrate utilization rates), and <i>9</i> (biomass decay coefficients).
cden(ncmp)	Component densities; Table 5.8: <i>Field 2</i> (organic components), <i>3</i> (water, oxygen, and nitrogen), and <i>4</i> (nutrient).	first(ncmp)	Boundary fluxes at nodes with constant concentration conditions.
ccx(nmf)	Nodal gas, aqueous, NAPL, solid, and bio phase mole component left hand side exchange terms.		
chen(ncmp)	Component Henry's Law constants; Table 5.8: <i>Field 2</i> (organic components), <i>3</i> (water, oxygen, and nitrogen), and <i>4</i> (nutrient).		
cmass0(ncmp5)	Component initial storages.		
cmass1(ncmp5)	Component current time step storages.		

flux(ncmpp5)	Phase and component boundary fluxes.	rhsex(nmf)	Nodal gas, aqueous, NAPL, solid, and bio phase mole component right hand side exchange terms.
fmb(icnl)	Global FEM right hand side vector for phase mass balance.	rxn(nmf)	Nodal aqueous and bio phase component reaction terms.
fuse(ncmp2)	Oxygen and nutrient use factors; Table 5.12: <i>Field 4</i> .	rxnp(nn2)	Nodal aqueous and bio phase reaction terms.
gama(nel3)	Nodal basis function z-derivatives.	sat(nnstk3)	Nodal gas phase, aqueous phase, and NAPL current time step saturations.
gamma(ncsqd)	Nodal component coefficients for vapor viscosity expression.	satk(nnstk2)	Nodal gas phase and aqueous phase previous iteration saturations.
kex(ncmp5)	Component aqueous/gas, aqueous/NAPL, gas/NAPL, aqueous/biophase, and aqueous/solid exchange coefficients; Table 5.9: <i>Field 1</i> .	satt(nnstk3)	Nodal gas phase, aqueous phase, and NAPL previous time step saturations.
khalf(ncmp)	Component half saturation constants; Table 5.12: <i>Field 4</i> .	sflux(ncmpp5)	Phase and component boundary fluxes at ground surface.
kinhib(ncmp)	Inhibition constants; Table 5.12: <i>Field 4</i> .	source(nn2)	Nodal phase boundary fluxes; Table 5.21 <i>Field 7</i> (gas phase) and 8 (aqueous phase).
kmax(ncmp5)	Component aqueous/gas, aqueous/NAPL, gas/NAPL, aqueous/biophase, and aqueous/solid minimum deviations from equilibriums; Table 5.9: <i>Field 1</i> .	srw(nelmx)	Nodal aqueous phase residual saturations.
krt(ncmp)	Component retardation factors; Table 5.11: <i>Field 5</i> .	str1(ncmpp5)	Current time step phase and component storages.
p(nn3)	Nodal gas phase, aqueous phase, and capillary current time step pressures; Table 5.17: <i>Field 3</i> .	str0(ncmpp5)	Initial phase and component storages.
pex(nn10)	Nodal gas, aqueous, NAPL, solid, and bio phase mole and mass phase exchange terms.	t(50)	Vector of real variables.
pmob(nnstk4)	Nodal aqueous and gas phase <i>x</i> and <i>z</i> mobilities.	t(1)	Initial time of simulation; Table 5.6: <i>Field 1</i> .
pmw(nn3)	Nodal gas, aqueous, and NAPL current time step phase molecular weights.	t(2)	Maximum or final time of simulation; Table 5.6: <i>Field 1</i> .
pmwt(nn3)	Nodal gas, aqueous, and NAPL previous time step phase molecular weights.	t(3)	Initial time step size; Table 5.6: <i>Field 5</i> .
pmw0(nnmx)	Initial gas phase molecular weights.	t(4)	Minimum time step size; Table 5.6: <i>Field 5</i> .
por(nelmx)	Element porosity.	t(5)	Maximum time step size; Table 5.6: <i>Field 5</i> .
pt(nn3)	Nodal gas phase, aqueous phase, and capillary previous time step pressures.	t(6)	Time step multiplier for increases; Table 5.6: <i>Field 8</i> .
q(nel4)	Gas phase and aqueous phase specific fluxes; Table 5.18 <i>Field 4</i> (uniform) or 5 (nonuniform).	t(7)	Time step multiplier for decreases; Table 5.6: <i>Field 8</i> .
rbar(nelmx)	Element radial coordinates.	t(8)	Current time step size.
rhs(ksolve)	Global FEM right hand side vector.	t(9)	Current simulation time.
		t(10)	Time weighting factor; Table 5.6: <i>Field 2</i> .
		t(11)	Delay period for initiation of bioreaction; Table 5.12: <i>Field 5</i> .
		t(12)	Print results at t(12) intervals if lprnt(0) is false; Table 5.4: <i>Field 8</i> .
		t(13)	Convergence criterion for pressures; Table 5.6: <i>Field 4</i> .

t(14)	Convergence criterion for mole fractions; Table 5.6: <i>Field 4</i> .	Icon(50)	Vector of logical switches to print specified variables to the contour plot file at specified print intervals.
t(15)	Convergence criterion for NAPL saturations; Table 5.6: <i>Field 4</i> .	Icon(1)	Contour in molar form; Table 5.15: <i>Field 2</i> .
t(16)	Convergence criterion for immobile phases; Table 5.6: <i>Field 4</i> .	Icon(2)	Contour gas phase pressures; Table 5.15: <i>Field 2</i> .
t(17)	Not used.	Icon(3)	Contour aqueous phase pressures; Table 5.15: <i>Field 2</i> .
t(18)	Not used.	Icon(4)	Contour gas/aqueous phase pressures; Table 5.15: <i>Field 2</i> .
t(19)	Not used.	Icon(5)	Contour gas phase mass and molar density; Table 5.15: <i>Field 2</i> .
t(20)	Not used.	Icon(6)	Contour aqueous phase mass and molar density; Table 5.15: <i>Field 2</i> .
t(21)	Horizontal component of gravity vector; Table 5.5: <i>Field 1</i> .	Icon(7)	Contour NAPL phase mass and molar density; Table 5.15: <i>Field 2</i> .
t(22)	Vertical component of gravity vector; Table 5.5: <i>Field 1</i> .	Icon(8)	Contour gas phase mole fractions; Table 5.15: <i>Field 2</i> .
t(23)	Maximum gas phase cell Peclet number.	Icon(9)	Contour aqueous phase mole fractions; Table 5.15: <i>Field 2</i> .
t(24)	Maximum aqueous phase cell Peclet number.	Icon(10)	Contour NAPL phase mole fractions; Table 5.15: <i>Field 2</i> .
t(25)	Maximum gas phase cell Courant number.	Icon(11)	Contour solid phase mole fractions; Table 5.15: <i>Field 2</i> .
t(26)	Maximum aqueous phase cell Courant number.	Icon(12)	Contour bio-phase mole fractions; Table 5.15: <i>Field 2</i> .
t(27)	Print mass balance at t(27) intervals if lprnt(25) is false; Table 5.4: <i>Field 5</i> .	Icon(13)	Contour gas phase saturation; Table 5.15: <i>Field 2</i> .
t(28)	Print time series at t(28) intervals if lprnt(26) is false; Table 5.4: <i>Field 6</i> .	Icon(14)	Contour aqueous phase saturation; Table 5.15: <i>Field 2</i> .
temp(nnmx)	Nodal temperatures.	Icon(15)	Contour NAPL phase saturation; Table 5.15: <i>Field 2</i> .
tort(nelmx)	Element tortuosity factors.	Icon(16)	Contour gas phase Darcy velocities; Table 5.15: <i>Field 2</i> .
umax(ncmp)	Maximum substrate use rate; Table 5.12: <i>Field 4</i> .	Icon(17)	Contour aqueous phase Darcy velocities; Table 5.15: <i>Field 2</i> .
vis(nnmx)	Nodal gas phase viscosity.	Icon(18)	Contour total organic soil concentration; Table 5.15: <i>Field 2</i> .
w(icnl)	Solver work space.	lctrl(50)	Vector of logical control switches.
xmf(nmf)	Nodal gas, aqueous, NAPL, solid, and bio phase current time step mole fractions.	lctrl(1)	Solve transient phase balance; Table 5.5: <i>Field 2</i> .
xmft(nmf)	Nodal gas, aqueous, NAPL, solid, and bio phase previous time step mole fractions.	lctrl(2)	Solve transient component balance; Table 5.5: <i>Field 2</i> .
xnode(nnmx)	Nodal x-coordinates; Table 5.7: <i>Field 9</i> .	lctrl(3)	Include bioreactions; Table 5.5: <i>Field 2</i> .
xyield(ncmp)	Biomass yield coefficient; Table 5.12: <i>Field 4</i> .		
znode(nnmx)	Nodal z-coordinates; Table 5.7: <i>Field 9</i> .		

Logical Arrays

lctrl(4)	Compute/print element dimensionless numbers; Table 5.5: <i>Field 7</i> .	lplt(2)	Generate time series output for the aqueous phase; Table 5.15: <i>Field 4</i> .
lctrl(5)	Generate a restart file; Table 5.4: <i>Field 7</i> .	lprnt(0:30)	Vector of logical switches to print specified variables to the output file at specified print intervals.
lctrl(6)	Not used.	lprnt(0)	T=print at specified time steps, f=print by time increment; Table 5.4: <i>Field 8</i> .
lctrl(7)	Mass lump flow equation; Table 5.5: <i>Field 3</i> .	lprnt(1)	Print grid information; Table 5.7: <i>Field 1</i> .
lctrl(8)	Mass lump transport equation; Table 5.5: <i>Field 3</i> .	lprnt(2)	Not used.
lctrl(9)	Switch defining if nutrient is to be modeled; Table 5.8: <i>Field 4</i> .	lprnt(3)	Print initial conditions; Table 5.15: <i>Field 1</i> .
lctrl(10)	Generate uniform temperature distribution; Table 5.14: <i>Field 1</i> .	lprnt(4)	Not used.
lctrl(11)	Not used.	lprnt(5)	Not used.
lctrl(12)	True if an extraction is to be simulated; Table 5.22: <i>Field 1</i> .	lprnt(6)	Compute and print mass balance information; Table 5.4: <i>Field 5</i> .
lctrl(13)	Make gas phase viscosity composition dependent.	lprnt(7)	Not used.
lctrl(14)	Couple flow and transport through exchange; Table 5.5: <i>Field 6</i> .	lprnt(8)	Print in molar form; Table 5.15: <i>Field 2</i> .
lctrl(15)	Print time series output; Table 5.4: <i>Field 6</i> .	lprnt(9)	Print nodal gas phase pressure; Table 5.15: <i>Field 2</i> .
lctrl(16)	Include bioreaction in aqueous transport; Table 5.12: <i>Field 2</i> .	lprnt(10)	Print nodal aqueous phase pressure; Table 5.15: <i>Field 2</i> .
lctrl(17)	Model biomass a steady state; Table 5.12: <i>Field 2</i> .	lprnt(11)	Print nodal gas/aqueous capillary pressure; Table 5.15: <i>Field 2</i> .
lctrl(18)	Use nodal Darcy velocities for transport; Table 5.18: <i>Field 1</i> .	lprnt(12)	Print nodal gas phase mole and mass density; Table 5.15: <i>Field 2</i> .
lctrl(19)	Use two compartment sorption model; Table 5.11: <i>Field 1</i> .	lprnt(13)	Print nodal aqueous phase mole and mass density; Table 5.15: <i>Field 2</i> .
lctrl(20)	Include Klinkenberg effect; Table 5.13: <i>Field 2</i> .	lprnt(14)	Print nodal NAPL mole and mass density; Table 5.15: <i>Field 2</i> .
lctrl(21)	Calculate hydrodynamic dispersion; Table 5.10: <i>Field 4</i> .	lprnt(15)	Print gas phase mole fractions; Table 5.15: <i>Field 2</i> .
lctrl(22)	Calculate density derivative terms.	lprnt(16)	Print aqueous phase mole fractions; Table 5.15: <i>Field 2</i> .
lctrl(23)	Print contour plot data; Table 5.4: <i>Field 4</i> .	lprnt(17)	Print NAPL phase mole fractions; Table 5.15: <i>Field 2</i> .
lctrl(24)	NAPL is present in the domain; Table 5.5: <i>Field 2</i> .	lprnt(18)	Print solid phase mole fractions; Table 5.15: <i>Field 2</i> .
lctrl(25)	Consider sorption; Table 5.5: <i>Field 2</i> .	lprnt(19)	Print bio-phase mole fractions; Table 5.15: <i>Field 2</i> .
lctrl(26)	Run is a restart; Table 5.16: <i>Field 1</i> .	lprnt(20)	Print nodal gas phase saturation; Table 5.15: <i>Field 2</i> .
lplt(20)	Vector of logical switches to print specified variables to the time series file.	lprnt(21)	Print nodal aqueous phase saturation; Table 5.15: <i>Field 2</i> .
lplt(1)	Generate time series output for the gas phase; Table 5.15: <i>Field 3</i> .	lprnt(22)	Print nodal NAPL phase saturation; Table 5.15: <i>Field 2</i> .

lprnt(23) Print gas phase Darcy velocities; Table 5.15: *Field 2*.

lprnt(24) Print aqueous phase Darcy velocities; Table 5.15: *Field 2*.

lprnt(25) T=print mass balance at specified time steps, f=print by time increment; Table 5.4: *Field 5*.

lprnt(26) T=print time series at specified time steps, f=print by time increment; Table 5.4: *Field 6*.

lprnt(27) T=print material balance in report form, f=print material balance in time series form with multiple output files; Table 5.4: *Field 5*.

lprnt(28) T=print time series file in mole fractions, f=print time series file in concentrations; Table 5.4: *Field 6*.

lprnt(29) T=print total organic soil concentration; Table 5.4: *Field 6*.

Character Arrays

cname(ncmp) Component labels; Table 5.8: *Field 2* (organic components), 3 (water, oxygen, and nitrogen), and 4 (nutrient).

infile(4) Input file labels; Table 5.4: *Field 1*.

outfile(8+ncmp) Output file labels.

Appendix F

EXAMPLE MAKE FILE

Below is a sample make file for compiling MISER on an IBM6000 or Sun Sparc workstation platform. In this example the source and object files are located in subdirectories called 'src' and 'obj', respectively. The executable file is named 'miser' and is generated in the directory containing the makefile.

```
# makemiser - to compile and link MISER code on Sun Sparc workstations.
#
# Define aliases for compilation and linking.
xl= f77 -O5 -o
xc = f77 -O5 -c -o
#
# Define paths to source and object files.
ps = ./src/
po = ./obj/
#
# Define an include file alias.
includes = $(ps)dimen.inc
#
# Define an object files alias.
objects = $(po)atri's.o $(po)bcflux's.o $(po)bio's.o \
$(po)cbal's.o $(po)commnt's.o $(po)disper's.o \
$(po)error's.o $(po)flow's.o $(po)grid's.o \
$(po)har's.o $(po)input1's.o $(po)input2's.o \
$(po)miser's.o $(po)mobil's.o $(po)molewt's.o \
$(po)mpex's.o $(po)napls's.o $(po)naplx's.o \
$(po)prnt's.o $(po)satw's.o $(po)solid's.o \
$(po)tlhs's.o $(po)trans's.o $(po)vel's.o \
#
# Link object files into an executable named 'miser'.
miser: $(objects) $(includes)
$(xl) $@ $(objects)
#
# Define object file dependencies.
$(po)atri's.o: $(ps)atri.f $(includes)
$(xc) $@ $(ps)atri.f
$(po)bcflux's.o: $(ps)bcflux.f $(includes)
$(xc) $@ $(ps)bcflux.f
$(po)bio's.o: $(ps)bio.f $(includes)
$(xc) $@ $(ps)bio.f
$(po)cbal's.o: $(ps)cbal.f $(includes)
$(xc) $@ $(ps)cbal.f
$(po)commnt's.o: $(ps)commnt.f $(includes)
$(xc) $@ $(ps)commnt.f
$(po)disper's.o: $(ps)disper.f $(includes)
$(xc) $@ $(ps)disper.f
$(po)error's.o: $(ps)error.f $(includes)
$(xc) $@ $(ps)error.f
$(po)flow's.o: $(ps)flow.f $(includes)
$(xc) $@ $(ps)flow.f
$(po)grid's.o: $(ps)grid.f $(includes)
$(xc) $@ $(ps)grid.f
$(po)har's.o: $(ps)har.f
$(xc) $@ $(ps)har.f
$(po)input1's.o: $(ps)input1.f $(includes)
$(xc) $@ $(ps)input1.f
$(po)input2's.o: $(ps)input2.f $(includes)
$(xc) $@ $(ps)input2.f
$(po)miser's.o: $(ps)miser.f $(includes)
$(xc) $@ $(ps)miser.f
$(po)mobil's.o: $(ps)mobil.f $(includes)
$(xc) $@ $(ps)mobil.f
$(po)molewt's.o: $(ps)molewt.f $(includes)
$(xc) $@ $(ps)molewt.f
$(po)mpex's.o: $(ps)mpex.f $(includes)
$(xc) $@ $(ps)mpex.f
$(po)napls's.o: $(ps)napls.f $(includes)
$(xc) $@ $(ps)napls.f
$(po)naplx's.o: $(ps)naplx.f $(includes)
$(xc) $@ $(ps)naplx.f
$(po)prnt's.o: $(ps)prnt.f $(includes)
$(xc) $@ $(ps)prnt.f
$(po)satw's.o: $(ps)satw.f $(includes)
$(xc) $@ $(ps)satw.f
$(po)solid's.o: $(ps)solid.f $(includes)
$(xc) $@ $(ps)solid.f
$(po)tlhs's.o: $(ps)tlhs.f $(includes)
$(xc) $@ $(ps)tlhs.f
$(po)trans's.o: $(ps)trans.f $(includes)
$(xc) $@ $(ps)trans.f
$(po)vel's.o: $(ps)vel.f $(includes)
$(xc) $@ $(ps)vel.f
```

Appendix G

EXAMPLE DATA FILES

Below are sample input files for the three example problems described in Section 6: example 1 is the SVE problem; example 2 is the first BV problem; and example 3 is the field scale BV problem. Note that data for example problems 2 and 3 are included as comment lines using the # in column 1.

The first input file should be named 'miser.d1' and located in the directory with the executable code.

```
# This is the data file for input1 for example 1 (SVE)
# Data for examples 2 (BV) and 3 (field scale BV) are included.
# Data fields can be separated by comment lines beginning with the '#'
# All data are input in free format.
#
===== BLOCK A: INPUT/OUTPUT FILES AND OP-
TIONS =====
#
# Field 1 - Input files.
# Specify the name of D2 and the error message file.
# The file name must be in single quotes.
#
# for example 1, the datafile for input2 is located in the subdirectory
# data
# 'data/d2.example1' 'miser.error'
# for example 2, the datafile for input2 is located in the subdirectory
# data
# 'data/d2.example2' 'miser.error'
# for example 3, the datafile for input2 is located in the subdirectory
# data
# 'data/d2.example3' 'miser.error'
#
# Field 2 - Prefix name for all output files.
# c.g. 'outpre.out', 'outpre.err', etc.
# The prefix name must also be entered in single quotes.
#
# for example 1, output is written to the tmp directory
# '/tmp/example1'
# for example 2, output is written to the tmp directory
# '/tmp/example2'
# for example 3, output is written to the tmp directory
# '/tmp/example3'
#
# Field 3 - Output unit numbers for error and performance information.
# Device numbers for:
# (1) printing error and warning messages, and
# (2) printing performance information.
# 0 = do not print information;
# 6 = screen;
# 21 = main output file (prefix.out);
# 22 = error message file (prefix.err) (only for error messages);
# 23 = convergence history file (prefix.cnv) (only for performance
# output).
#
# for examples 1, 2 and 3, print information to the screen
6 6
#
# Field 4 - Contour plot file.
# Logical switch to open and print contour plot data to the file
# 'outpre.con' (lctrl(23) = t or f).
#
# for examples 1, 2 and 3
t
#
# Field 5 - Mass balance output file.
# Enter either 1 or 3 lines of data.
# line 1: A logical switch to open and print mass balance results to
# the file 'outpre.mb' (lprnt(6) = t or f).
# line 2: Only needed if lprnt(6) above is true, otherwise disregard.
# Enter a logical variable lprnt(25) indicating if the print
# interval is set by the number of time steps (true) or by a
# constant time interval (false). Enter a logical variable
# lprnt(27) indicating if the material balance is in report
# form (t) or in multiple files in time series form (f).
# line 3: Only needed if lprnt(6) above is true, otherwise disregard.
# Enter either the number of time steps ipt(83) or the uniform
# print interval in seconds t(27).
#
# for examples 1, 2 and 3, print the mass balance every 10 days
t
f f
8.64d5
#
# Field 6 - Time series output file.
# Enter either 1 or 3 three lines of data.
# line 1: A logical switch to open and print time series results to
# the file 'outpre.plt' (lctrl(15) = t or f).
# line 2: Only needed if lctrl(15) above is true, otherwise disregard.
# Enter a logical variable lprnt(26) indicating if the print
# interval is set by the number of time steps (true) or by a
# constant time interval (false). Enter a second logical
# switch lprnt(26) indicating the concentration units for
# output (t=mole fraction, f=concentration).
# line 3: Only needed if lctrl(15) above is true, otherwise disregard.
# Enter either the number of time steps ipt(84) or the uniform
# print interval in seconds t(28).
#
# for examples 1, 2 and 3
f
#
# Field 7 - Restart file.
# Open and print restart data to the file 'outpre.rst'
# (lctrl(5) = t or f).
#
# for examples 1, 2 and 3
t
#
# Field 8 - Uniform print interval to the main output file.
# Enter two lines:
# (1) a logical variable indicating if the print interval is set
# by the number of time steps (true) or by a constant time
# interval (false);
# (2) either the number of time steps ipt(25) or the uniform print
# interval in seconds t(12).
#
# for examples 1, 2 and 3, print the main output file every 20 days
f
1.7280d6
```



```

#
#===== TITLE CARDS =====
#
# Define an unspecified number of title cards, including zero, if none
# are desired. Title or comment cards are indicated with an '&' symbol
# in column one. These can be placed anywhere in the main output file,
# provided they are positioned in the input files between data fields.
#
&
&Sample data file for input 1: examples 1, 2, and 3.
&
&
#
#===== BLOCK B: GENERAL MODEL CONTROL OPTIONS =====
#
# Field 1 - Coordinate system.
# Specify: (1) an integer switch: 0=cross-sectional (x-z);
#           1 = axial-symmetric (r-z);
#           (2) horizontal component of the gravitational constant
#           (m/s^2);
#           (3) vertical component of the gravitational constant
#           (m/s^2);
#
# for examples 1, 2 and 3
1 0.d0 9.81d0
#
# Field 2 - Equation solution options.
# Define 3 logical variables (t or f) indicating which balance equations
# are to be solved:
# (1) solve the phase balance eqs;
# (2) solve the component transport eqs;
# (3) solve napl equations
# (4) solve solid phase equations
# (5) solve biophase equations
# The flow equations can be solved without solving transport equations;
# the transport eqs can be solved with an input steady state flow
# distribution, option 2 must be true for options 3, 4, or 5 to be true;
# for example 1
t t t t f
# for example 2
# t t f f t
# for example 3
# t t t t t
#
# Field 3 - Mass lumping options.
# Define two logical variables indicating if lumping of the mass matrix
# is to be performed in the solution of the flow eqs and in the
# solution of the transport equations.
#
# for examples 1, 2 and 3
t t
#
# Field 4 - Flow solution skipping option.
# Read an integer parameter denoting the number of time steps to be
# skipped between solving for the flow equations.
#
# for examples 1, 2 and 3
0
#
# Field 5 - not used
#
# Field 6 - Coupling between flow and transport.
# Enter a logical variable indicating if mass exchange terms
# should be included in the solution of the flow equations.
#
# for examples 1, 2 and 3
t
#
# Field 7 - Element dimensionless numbers.
# Enter a logical variable indicating if element dimensionless
# numbers are to be calculated for the transport solution
#
# for examples 1, 2 and 3
t
#
#===== BLOCK C: TIME STEP AND ITERATION CONTROL =====
#
# Field 1 - Simulation time frame.

```

```

# The initial (t(1)) and final (t(2)) simulation time in seconds.
#
# for examples 1, 2 and 3, run simulation for 100 days
0.d0 8.640d6
#
# Field 2 - Time weighting.
# Specify the time weighting parameter (t(10)): 0=explicit; 1=implicit;
# 0.5 = Crank-Nicolson.
#
# for examples 1, 2 and 3
1.d0
#
# Field 3 - Maximum number of time steps (ipt((30))).
#
# for examples 1, 2 and 3
10000000
#
# Field 4 - Convergence tolerance.
# Convergence tolerance in the solution of:
# the phase balance eqs t(13); the component transport eqs t(14);
# the bioreaction eqs. t(15); the immobile phase eqs. t(16).
#
# for examples 1, 2 and 3
1.d-5 1.d-8 1.d-8 1.d-8
#
# Field 5 - Time step range.
# Initial time step size (t(3)) in seconds; the minimum time step size
# t(4); and the maximum time step size (t(5)).
#
# for examples 1, 2 and 3
1.d0 1.d-4 3.6d3
#
# Field 6 - Maximum iterations for convergence.
# Maximum number of iterations for convergence in the solution of:
# the phase balance eqs ipt(31); the component transport eqs
# ipt(32); the bioreaction eqs. ipt(33) (ipt(33) currently not used).
#
# for examples 1, 2 and 3
15 30 30
#
# Field 7 - Maximum iterations for time step amplification.
# Maximum number of iterations for time step amplification in the
# solution of: the phase balance solution ipt(34), and in the
# solution of the transport eqs. ipt(35).
# The minimum must be less than the maximum number of iterations.
#
# for examples 1, 2 and 3
7 7
#
# Field 8 - Time step multiplication factors.
# Empirical time step amplification t(6) and reduction (7) factors.
# The time step is increased by a factor t(6) if the number of
# iterations for convergence is less than the minimum; conversely the
# step is reduced by the factor t(7) if the number of iterations for
# convergence is greater than the maximum.
#
# for examples 1, 2 and 3
1.05d0 0.75d0
#
#===== BLOCK D: GRID PARAMETERS AND OPTIONS =====
#
# Field 1 - Output grid geometry.
# Enter a logical switch indicating if all grid information should be
# printed to the main output file?
#
# for examples 1, 2 and 3
f
#
# Field 2 - Grid specification options.
# Generate the grid for a rectangular homogeneous domain?
# Enter an integer value: 0 = don't generate the grid, input all
# element numbers and nodal coordinates.
#           1 = generate union jack grid.
#           2 = generate herring bone grid.
#
# for examples 1, 2 and 3
2
#

```

```

# Field 3 - Number of blocks.
# If a grid is to be generated (igrd>0), then enter the number of
# blocks in the horizontal (nx) and vertical (nz) directions,
# respectively. Skip if no grid is generated (igrd=0).
#
# for example 1
31 18
# for example 2
# 39 18
# for example 2
# 40 56
#
# Field 4 - Horizontal Block spacing in generated grid.
# If a grid is to be generated (igrd>0), then enter:
# first line - a logical variable indicating if the spacing is uniform,
# and the horizontal coordinate of the left boundary;
# succeeding lines - horizontal spacing (one value if uniform, or nx
# values if nonuniform). Units are assumed to be
# meters.
# Skip if no grid is generated (igrd=0).
#
# for example 1
f 0.25d0
0.01d0 0.02d0 0.03d0
0.04d0 0.05d0 0.06d0 0.07d0 0.08d0 0.09d0 0.1d0 0.12d0 0.15d0 0.20d0
0.25d0 0.3d0 0.4d0 0.5d0 0.6d0 0.75d0 0.95d0 1.2d0 1.5d0 1.85d0
2.25d0 2.75d0 3.35d0 4.4d0 4.75d0 5.0d0 5.0d0 5.0d0
# for example 2
# f 0.25d0
# 0.01d0 0.02d0 0.03d0 0.04d0 0.05d0 0.06d0 0.07d0 0.08d0 0.09d0 0.1d0
# 0.12d0 0.15d0 0.20d0 0.25d0 0.3d0 0.4d0 0.5d0 0.6d0 0.75d0 0.95d0
# 1.2d0 1.5d0 1.85d0
# 2.d0
# 2.d0 2.d0 2.d0 2.d0 2.d0 2.d0 2.d0 2.d0 2.d0 2.d0
# 2.d0 2.d0 2.d0 2.d0 2.d0
# for example 3
# f 0.25d0
# 0.0535701 0.0650491 0.0789878 0.0959133 0.1164657
# 0.1414220 0.1717259 0.2085235 0.2532059 0.3074629
# 0.3733462 0.4533470 0.5504901 0.6684494 0.8116848
# 0.9856129 1.4d0 1.4d0 1.4d0 1.4d0
# 1.d0 1.d0 1.d0 1.d0 1.d0
# 1.d0 1.d0 1.d0 1.d0 1.d0
# 1.2d0 1.4d0 1.6d0 1.8d0 2.0d0
# 2.d0 2.d0 2.d0 2.d0 2.d0
#
# Field 5 - Vertical Block spacing in generated grid.
# If a grid is to be generated (igrd>0), then enter:
# first line - a logical variable indicating if the spacing is uniform,
# and the horizontal coordinate of the left boundary;
# succeeding lines - horizontal spacing (one value if uniform, or nx
# values if nonuniform). Units are assumed to be
# meters.
# Skip if no grid is generated (igrd=0).
#
# for example 1 and 2
t 0.d0
0.25d0
# for example 3
# f 0.d0
# 0.25d0 0.25d0 0.25d0 0.25d0
# 0.25d0 0.25d0 0.25d0 0.25d0
# 0.25d0 0.25d0 0.25d0 0.25d0
# 0.25d0 0.25d0 0.25d0 0.25d0
# 0.25d0 0.25d0 0.25d0 0.25d0
# 0.25d0 0.25d0 0.25d0 0.25d0
# 0.20d0 0.20d0 0.10d0 0.10d0 0.10d0 0.10d0 0.10d0 0.10d0
# 0.10d0 0.10d0 0.10d0 0.10d0 0.15d0 0.15d0 0.15d0 0.20d0 0.20d0
# 0.25d0 0.25d0 0.25d0 0.25d0
# 0.30d0 0.40d0 0.50d0 0.50d0 0.50d0 0.50d0
# 0.50d0 0.50d0 0.50d0 0.50d0
#
# Field 6 - Horizontally aligned material property blocks.
# If a grid is to be generated (igrd>0), then enter an integer number
# of horizontal material property blocks (ipt(26)).
# If the number of blocks is greater than 1, then beginning on the
# following line, enter the material block number for each vertical

```

```

# spacing. There must be nz integer values in order from top to bottom.
# Skip if no grid is generated (igrd=0).
#
# for example 1 and 2
2
1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1
# for example 3
# 3
# 1 1 1 1 1 1 1
# 2 2 2 2 2 2 2
# 2 2 2 2
# 3 3 3 3 3 3 3
# 3 3 3 3 3 3 3
# 3 3 3 3 3 3 3
# 3 3 3 3
#
# Field 7 - Grid dimensions.
# If a grid is to be input (igrd=0), then enter:
# the number of elements (ipt(0)); the number of nodes (ipt(1)); and the
# number of material property blocks (ipt(26)).
# Skip if the grid is generated (igrd>0).
#
# Field 8 - Nodal incidence list.
# If a grid is to be input (igrd=0), then enter the nodal incidence
# list and the material property block number for each element.
# The element node incidence list consists of the arbitrary global
# element number followed by that element's three global node numbers.
# Each element has its own line. The element node numbers
# start at an arbitrary node. If the z coordinate is positive
# downwards proceed in the clockwise direction, otherwise proceed
# in the counterclockwise direction. If there is only one material
# property block for the entire domain, the material property input
# assignment for each element is omitted. The minimum material
# property block is a two element quadrilateral.
#
# Field 9 - Nodal coordinates
# If a grid is to be input (igrd=0), then enter the nodal coordinates.
# For each node enter 1 line of data giving: the node number; horizontal
# coordinate; and the vertical coordinate. Units are assumed to be
# meters. Skip if the grid is generated (igrd>0).
#
===== BLOCK E: COMPONENT INFORMATION =====
#
# Field 1 - Number of NAPL components.
# Specify the number of components in the NAPL (ipt(15))
#
# for example 1 and 2
1
# for example 3
# 2
#
# Field 2 - NAPL component chemical properties.
# For each NAPL component enter in order:
# (1) component number
# (2) component name (character variable in single quotes)
# (3) component molecular weight (g/mole)
# (4) component vapor pressure (atm)
# (5) component vapor viscosity (cPoise)
# (6) component liquid density (g/l)
# (7) component gas diffusivity (cm2/s)
# (8) component aqueous diffusivity (cm2/s)
# (9) component henry's constant (atm l/g)
# (10) component aqueous solubility (g/l)
# Data for each component must start on a new line.
# Note: a component can be excluded from the gas phase or the aqueous
# phase by entering a negative value for the vapor pressure or aqueous
# solubility, respectively. Organic components should be entered in
# order of volatility starting with the most volatile.
# Skip this item if no NAPL components are specified (ipt(15)=0).
#
# for example 1 and 2
1 'toluene' 92.1340d0 2.940d-2 7.0d-3 867.0d0
8.5d-2 9.540d-6 5.70d-2 .5150d0
# for example 3
# 1 'benzene' 78.10d0 0.102d0 7.5d-3 879.0d0
# 8.8d-2 9.0d-6 5.70d-2 1.78d0
# 2 'xylene' 106.2d0 0.0092d0 7.0d-3 880.1d0

```

```

# 6.2d-2 7.2d-6 5.70d-2 .175d0
#
# Field 3 - Chemical property data for water, oxygen, and nitrogen.
# Enter the 10 parameter values listed above. The ordering is assumed
# to be: water, oxygen, and nitrogen.
# The component number is always: water = ipt(15)+1; oxygen = ipt(15)+2;
# nitrogen = ipt(15)+3
# Any of these components can be eliminated by specifying negative
# values for both the vapor pressure and water solubility.
# Note, if water is eliminated then there is no aqueous phase pressure
# and the flow equations cannot be solved.
# If nitrogen is eliminated, then there is no gas phase, (i.e. the vapor
# pressure must be negative for all components). The nitrogen
# solubility is a dummy input; it is not used in computations.
# for example 1
2 'water' 18.0d0 -2.310d-2 9.750d-3 998.0d0
.2450d0 0.0d0 0.0d0 1.0d0
3 'oxygen' 32.0d0 -0.20d0 0.0192d0 998.0d0
8.5d-2 2.153d-5 0.0d0 -.009d0
4 'nitrogen' 28.02d0 1.0d0 0.0172d0 998.0d0
0.0d0 0.0d0 0.0d0 -1.0d0
# for example 2
# 2 'water' 18.0d0 -2.310d-2 9.750d-3 998.0d0
# .2450d0 0.0d0 0.0d0 1.0d0
# 3 'oxygen' 32.0d0 0.20d0 0.0192d0 998.0d0
# 8.5d-2 2.153d-5 0.0d0 .009d0
# 4 'nitrogen' 28.02d0 1.0d0 0.0172d0 998.0d0
# 0.0d0 0.0d0 0.0d0 -1.0d0
# for example 3
# 3 'water' 18.0d0 -0.0231d0 9.50d-3 998.2d0
# .2450d0 0.0d0 0.0d0 1.0d0
# 4 'oxygen' 32.0d0 0.20d0 0.02d0 998.0d0
# 8.5d-2 2.153d-5 0.0d0 .009d0
# 5 'nitrogen' 28.02d0 1.0d0 0.0174d0 998.0d0
# 0.0d0 0.0d0 0.0d0 -1.0d0
#
# Field 4 - Nutrient inclusion.
# Enter a logical variable indicating whether a nutrient component is
# to be modeled.
#
# for examples 1, 2 and 3
f
#
# Field 5 - Nutrient Chemical Properties.
# If a nutrient is modeled, then specify the 10 chemical
# property parameters listed above.
# The component number of nutrient is always = ipt(15)+4.
# Skip this item if no nutrient is modeled.
#
# ===== BLOCK F: MASS EXCHANGE INFORMATION =====
#
# Field 1 - Interphase mass exchange coefficients.
# Enter: (1) component number which is identical to the order entered
# above; nitrogen = ipt(15)+3; nutrient=ipt(15)+4)
# (2) aqueous/gas exchange coefficient (1/sec)
# (3) aqueous/NAPL exchange coefficient (1/sec)
# (4) gas/NAPL (1/sec)
# (5) aqueous/biophase (1/sec)
# (6) aqueous/solid (1/sec)
# Note: a zero value for aqueous/biophase mass exchange coefficient
# indicates that the component does not partition between that phase
# pair.
# On a second line enter minimum deviations from equilibrium for each
# exchange coefficient entered on the previous line. DO NOT ENTER A
# VALUE LESS THAN 0.05D0.
# Enter: (1) component number which is identical to the order entered
# above; nitrogen = ipt(15)+3; nutrient=ipt(15)+4)
# (2) aqueous/gas minimum deviations from equilibrium
# (3) aqueous/NAPL minimum deviations from equilibrium
# (4) gas/NAPL minimum deviations from equilibrium
# (5) aqueous/biophase minimum deviations from equilibrium
# (6) aqueous/solid minimum deviations from equilibrium
#
# for example 1
# toluene
1 5.0d-5 5.0d-4 5.0d-4 0.0d0 5.d-5
1 0.1d0 0.1d0 0.1d0 0.1d0 0.1d0
# water

```

```

2 0.d0 0.d0 0.d0 0.d0 0.d0
2 0.1d0 0.1d0 0.1d0 0.1d0 0.1d0
# oxygen
3 0.d0 0.d0 0.d0 0.d0 0.d0
3 0.1d0 0.1d0 0.1d0 0.1d0 0.1d0
# nitrogen
4 0.d0 0.d0 0.d0 0.d0 0.d0
4 0.1d0 0.1d0 0.1d0 0.1d0 0.1d0
# for example 2
# toluene
# 1 5.0d-6 5.0d-5 5.0d-5 0.d0 0.d0
# 1 0.1d0 0.1d0 0.1d0 0.1d0 0.1d0
# water
# 2 0.d0 0.d0 0.d0 0.d0 0.d0
# 2 0.1d0 0.1d0 0.1d0 0.1d0 0.1d0
# oxygen
# 3 5.0d-6 5.0d-5 5.0d-5 0.d0 0.d0
# 3 0.1d0 0.1d0 0.1d0 0.1d0 0.1d0
# nitrogen
# 4 0.d0 0.d0 0.d0 0.d0 0.d0
# 4 0.1d0 0.1d0 0.1d0 0.1d0 0.1d0
# for example 3
# benzene
# 1 1.157d-5 2.315d-4 5.787d-4 0.d0 1.157d-5
# 1 0.075d0 0.075d0 0.075d0 0.075d0 0.075d0
# o-xylene
# 2 1.157d-5 2.315d-4 5.787d-4 0.d0 1.157d-5
# 2 0.075d0 0.075d0 0.075d0 0.075d0 0.075d0
# water
# 3 0.d0 0.d0 0.d0 0.d0 0.d0
# 3 0.075d0 0.075d0 0.075d0 0.075d0 0.075d0
# oxygen
# 4 1.157d-5 2.315d-4 5.787d-4 0.d0 1.157d-5
# 4 0.075d0 0.075d0 0.075d0 0.075d0 0.075d0
# nitrogen
# 5 0.d0 0.d0 0.d0 0.d0 0.d0
# 5 0.075d0 0.075d0 0.075d0 0.075d0 0.075d0
#
# ===== BLOCK G: MATERIAL PROPERTY BLOCK DATA =====
#
# Field 1 - Soil physical properties:
# For each material property block specify:
# (1) material block number;
# (2) porosity;
# (3) horizontal permeability (m^2);
# (4) vertical permeability (m^2);
# (5) bulk soil density (gm/cm^3)
# (6) organic carbon content
# Data for all blocks must be defined.
# Data for each block begins on a new line
#
# for example 1 and 2
1 0.33d0 1.d-11 0.8d-11 1.7d0 0.001d0
2 0.35d0 0.6d-11 0.4d-11 1.7d0 0.001d0
# for example 3
# 1 0.33d0 1.0d-11 1.0d-11 1.7d0 0.001d0
# 2 0.33d0 5.0d-11 5.0d-11 1.7d0 0.001d0
# 3 0.33d0 0.7d-11 0.7d-11 1.7d0 0.001d0
#
# Field 2 - Water retention parameters:
# For each material property block specify:
# (1) material block number;
# (2) residual water saturation;
# (3) van Genuchten n for air/water retention data;
# (4) van Genuchten alpha for air/water retention data (1/Pa).
# Data for all blocks must be defined.
# Data for each block begins on a new line
#
# for example 1 and 2
1 0.12d0 7.0d0 .002d0
2 0.16d0 5.0d0 .0008d0
# for example 3
# 1 0.073d0 3.97d0 4.34d-4
# 2 0.073d0 3.97d0 7.47d-4
# 3 0.073d0 3.97d0 2.79d-4
#
# Field 3 - Dispersion parameters:
# For each material property block specify:

```

```

# (1) material block number;
# (2) longitudinal dispersivity (m);
# (3) transverse dispersivity (m);
#
# for example 1 and 2
1 1.0d0 0.01d0
2 1.0d0 0.01d0
# for example 3
# 1 0.5d0 0.01d0
# 2 0.5d0 0.01d0
# 3 0.5d0 0.01d0
#
# Field 4 - Dispersion tensor computation.
# Enter a logical variable (lctrl(21)) indicating that the
# hydrodynamic dispersion tensor should be calculated. Enter false if
# a known and constant dispersion tensor is to be input.
#
# for examples 1, 2 and 3
t
#
# Field 5 - Dispersion tensor.
# If lctrl(21) above is false then enter the hydrodynamic dispersion
# tensor for each component present with two lines, the first line is
# for the gas phase and the second line is for the aqueous phase. The
# first entry on each line is the component number.
#
#==== BLOCK H: SORPTION PARAMETERS =====
#
# Field 1 - Sorption model (required if lctrl(25)=true):
# Enter a logical variable (lctrl(19)) indicating if sorption is
# modeled as a one (false) or two (true) compartment process.
# Note: the two compartment model is currently limited to conditions of
# a homogeneous soil domain and a single component NAPL.
#
# for examples 1 and 3
f
#
# Field 2 - Single compartment Freundlich sorption parameters
# (required if lctrl(25)=true):
# For each material property block enter two groups of data:
# (1) the material block number, followed by the k parameter values
# for each organic component in order
# from 1 to the number of components (micrograms/gram solid,
# with aqueous concentration in mg/l)
# (2) the material block number, followed by the n=1/m parameter
# (dimensionless) values ordered in the same way.
#
# for example 1
1 7.72d0
1 0.544d0
2 7.72d0
2 0.544d0
# for example 3
# 1 1.16d0 0.36d0
# 1 0.862d0 1.07d0
# 2 1.16d0 0.36d0
# 2 0.862d0 1.07d0
# 3 1.16d0 0.36d0
# 3 0.862d0 1.07d0
#
# Field 3 - Two compartment sorption data:
# The two compartment model has a slow and a fast compartment.
# Both compartments are modeled with the Freundlich equation.
# Four parameters must be input:
# (1) multiplier to convert slow compartment kf parameter to the fast
# compartment value;
# (2) multiplier to convert slow compartment n=1/m parameter to the
# compartment value;
# (3) multiplier to convert slow compartment mass transfer
# coefficient to the fast compartment value;
# (4) mass fraction of solid phase in the fast compartment.
# Skip this item if the single compartment model is used
# (lctrl(19)=false)
#
# Field 4 - Include retardation factor (required if lctrl(25)=false).
# Enter a logical variable indicating if retardation factors are used.
#
# for example 2

```

```

# f
#
# Field 5 - Retardation factors (required if lctrl(25)=false and
# lretrd=true)
# For each component give the component number and the retardation
# factor. Retardation factors must be entered for all components,
# however 1.0 implies no sorption.
#
#==== BLOCK I: BIOLOGICAL PARAMETERS =====
#
# Field 1 - Number of biodegradable substrates.
# Specify the number of biodegradable substrates, ipt(17). The biophase
# always contains oxygen and nutrient if present.
#
# for example 2
# 1
# for example 3
# 2
#
# Field 2 - Biodegradation control switches.
# Specify 2 logical variables indicating:
# (1) if a steady state biomass is to be used (value=true), or if a
# transient biomass is to be modeled (value=false).
# (2) if biodegradation equations are modeled as a sink term
# in the aqueous transport equations (value=true),
# otherwise their modeled as rate-limited exchange to a
# separate biophase (value=false).
#
# for examples 2 and 3
# f t
#
# Field 3 - Growth kinetics option.
# Specify an integer value indicating the type of growth
# kinetics: (ONLY OPTIONS 1 AND 2 AVAILABLE NOW)
# 1 = standard Monod kinetics
# 2 = Monod kinetics with substrate inhibition
# 3 = Monod kinetics with lumped substrate inhibition
# 4 = Monod kinetics with saturation dependency
# and substrate inhibition
# 5 = Monod kinetics with saturation dependency and
# substrate inhibition
#
# for examples 2 and 3
# 2
#
# Field 4 - Monod parameters:
# For each component in the biophase provide the following information:
# (1) component number as defined in block E.
# (2) electron acceptor use coefficient (gm-O2/gm-substrate)
# (3) nutrient use coefficient (gm-nutrient/gm-substrate)
# (4) maximum substrate use rate (gm-substrate/gm-biomass/sec)
# (5) half saturation constant (gm-component/l)
# (6) yield coefficient (gm-biomass/gm-substrate)
# (7) inhibition constant (dimensionless) expressed as a fraction of
# the aqueous solubility. For substrate and nutrient this
# turns off metabolism when the concentration is above this
# threshold concentration and for electron acceptor this
# turns off metabolism when the concentration is below this
# threshold concentration. In both cases hyperbolic functions
# are used.
#
# for example 2
# toluene
# 1 2.19d0 1.5d0 1.157d-6 17.4d-3 0.5d0 0.25d0
# oxygen
# 3 2.19d0 1.5d0 1.157d-6 0.1d-3 0.5d0 0.2d0
# for example 3 (high set)
# benzene
# 1 2.19d0 0.d0 1.157d-5 0.5d-3 0.5d0 0.5d0
# o-xylene
# 2 2.19d0 0.d0 1.157d-5 0.5d-3 0.5d0 0.5d0
# oxygen
# 4 2.19d0 0.d0 1.157d-5 0.5d-3 0.5d0 0.5d0
#
# Field 5 - Decay and biomass range coefficients.
# Specify:
# (1) the decay coefficient (sec)
# (2) the minimum biomass (g/l)

```

```

# (3) the maximum biomass (g/l)
# (4) the initial uniform biomass (g/l)
# (5) delay period for initiation of bioreaction (sec)
#
# for example 2
# 1.157407d-7 1.d-3 100.0d-3 1.62d-3 8.64d4
# for example 3
# 1.157407d-6 1.d-6 2.d-2 1.d-6 8.64d4
#
#===== BLOCK J: PHASE PARAMETERS =====
#
# Field 1 - Water phase viscosity.
# Specify the water phase viscosity (cPoise)
#
# for examples 1 and 2
1.124d0
# for example 3
# 1.002d0
#
# Field 2 - Gas phase slip flow parameters.
# Specify:
# (1) a logical variable indicating if the Klinkenberg
# effect is to be modeled (value=true);
# (2) the Klinkenberg parameter (atm). Set the parameter to zero
# if the previous line is false.
#
# for examples 1, 2 and 3
f 0.d0
#
#===== BLOCK K: TEMPERATURE PARAMETERS =====
#
# Field 1 - Temperature distribution.
# Specify a logical variable indicating if temperature
# distribution is uniform (true).
#
# for examples 1, 2 and 3
t
#
# Field 2 - Uniform temperature distribution.
# If the temperature distribution is uniform, enter a single uniform
# value (degree C); otherwise enter a temperature value for each node
# along the vertical edge of the domain.
#
# for examples 1, 2 and 3
20.d0
#
# Field 3 - Nonuniform temperature distribution.
# A vertical nonuniform temperature distribution can be defined only in
# association with a generated rectangular grid.
# Temperature values are needed for each vertical node along the
# boundary starting at the surface, downward (ny+1 values).
# For each node enter 1 line of data giving the:
# 1) vertical depth of node (for error checking)
# 2) temperature at the node (degree C)
#
# Field 4 - Temperature dependent chemical properties.
# Temperature dependent chemical properties are needed for each
# component for the following 6 properties:
# (1) component vapor pressure
# (2) component vapor viscosity
# (3) component Henry's law constant
# (4) component aqueous solubility
# (5) component maximum specific utilization rate
# (6) biomass decay rate
# For each of these properties give the temperature dependent value at
# each of the ny+1 nodes along the vertical boundary.
# Provide the information for all 6 properties for a given component and
# then repeat for the next component.
# Use the original component ordering as in the component information
# section. After all the component values give the kd values.
#
# Vapor pressure of component 1 at ny+1 nodes
#
# Vapor viscosity of component 1 at ny+1 nodes
#
# Henry's law constant of component 1 at ny+1 nodes
#
# aqueous solubility of component 1 at ny+1 nodes
#
# maximum specific utilization rate of component 1 at ny+1 nodes
#
# Repeat above for all NAPL components, then water, oxygen, nitrogen and
# nutrient if present.
# After all components have been entered, provide ny+1 nodal values for
# the Kd coefficient.
#
#===== BLOCK L: OUTPUT CONTROL PARAMETERS =====
#
# Field 1 - Print initial conditions.
# Enter a logical variable (LPRNT(3)) indicating if initial conditions
# should be printed for selected variables.
#
# for examples 1, 2 and 3
t
#
# Field 2 - Print switches.
# Read print switches for specified variables, the first switch for
# each variable is for the printed output, the second switch for
# each variable is for the contouring output: Use component numbers
# based established in block E above:
# This input is currently set for minimal output. The user should
# specify output of interest.
t f print/contour output in molar form
t f print/contour nodal gas phase pressure
t f print/contour nodal aqueous phase pressure
f f print/contour nodal gas/aqueous capillary pressure
f f print/contour nodal gas phase density
f f print/contour nodal aqueous phase density
f f print/contour nodal NAPL phase density
t f print/contour nodal gas phase components
1 1
t t print/contour nodal aqueous phase components
1 1
f f print/contour nodal NAPL phase components
f f print/contour nodal solid phase loadings
f f print/contour nodal bio-phase components
t f print/contour element avg total organic soil mass fraction
t f print/contour nodal gas phase saturation
t f print/contour nodal aqueous phase saturation
t f print/contour nodal NAPL phase saturation
t f print/contour gas phase Darcy velocities
f f print/contour aqueous phase Darcy velocities
#
# Field 3 - Gas phase time series plot switches.
# line(1) - enter a logical variable indicating if time series plot
# output files should be generated for gas phase
# line(2) - If .true. enter the number of gas phase components;
# followed by the global component number and the
# associated node number for the location from which to
# output.
# A maximum of 6 components can be defined for the combined gas and
# aqueous phase below.
f time series plot nodal gas phase components
#
# Field 4 - Aqueous phase time series plot switches.
# line(1) - enter a logical variable indicating if time series plot
# output files should be generated for gas phase
# line(2) - If .true. enter the number of gas phase components;
# followed by the global component number and the
# associated node number for the location from which to
# output.
# A maximum of 6 components can be defined for the combined gas and
# aqueous phase below.
f time series plot nodal aqueous phase components

```

In this example the second input file is named 'd2.example1' and is located in a subdirectory called 'data.'

```

# This is the data file for input2 for example 1 (SVE)
# Data for examples 2 (BV) and 3 (field scale BV) are included.
# Data field can be separated by comment lines beginning with the '#'
# All data are input in free format.
#
#====Block M: RESTART IDENTIFIER =====
# Field 1 - restart control switches
# 1) indicates if this run is a restart (lctrl(26));
# 2) indicates if the mass balance is to be reset (lctrl(32)).
#
# for examples 1 and 2
# f f
# for example 3
# t f
#
# Field 2 - restart file identifier: only required if lctrl(26) = true
# This file is a renamed copy of the file 'outpre.rst'
#
# 'data/d3.example1'
# 'data/d3.example2'
# 'data/d3.example3'
#
#====Block N: INITIAL PRESSURE CONDITIONS =====
#
# Field 1 - Initial pressure distribution
# Enter an integer control variable (ipt(75)) indicating how the
# initial pressure distribution is to be input:
# 1 = Compute initial pressure distribution assuming P'g = 1 atm
# and P'a is hydrostatic referenced to atmospheric pressure
# at the water table. The water table is assumed to be flat
# and the z-axis is vertical.
# 2 = Input gas and aqueous pressures at all nodes;
#
# for examples 1, 2 and 3
# 1
#
# Field 2 - Depth to water table (m).
# This variable is read only if ipt(75)=1
#
# for examples 1 and 2
# 40.d0
# for example 3
# 8.0d0
#
# Field 3 - Initial Pressures.
# Required only read if ipt(75)=2
# For each node enter one line of data giving:
# (1) the node number; (2) the initial water pressure; and (3) the
# initial air pressure. All pressures are gauge pressures in Pascals.
# Nodes need not be in order. USE -NODE# FOR UNIFORM
#
#==== BLOCK O: VELOCITY COMPUTATION APPROACH =====
#
# Field 1 - Velocity computation method.
# Enter a logical variable indicating if nodal velocities should be
# computed (true), or if element average velocities are used (false).
#
# for examples 1, 2 and 3
# t
#
# Field 2 - Steady state velocity distribution.
# If the flow field is transient (lctrl(1) = true.) then no other input
# are required in this block. If a steady state flow field is assumed
# (i.e. lctrl(1) = false.) then enter a logical variable indicating if
# the velocity distribution should be computed from the pressure field
# (true) or if the velocities are to be input (false).
# Note: This input needed only if (lctrl(1)=false).
#
# Field 3 - Input velocity distribution.
# If the velocity distribution is SS the enter a logical variable (lcv)
# indicating if the velocity components are uniform (true) or
# nonuniform (false).
# Note: This input needed only if the flow field is not transient and
# previous input=false.
#
# Field 4 - Uniform velocity components.

```

```

# Required only if lcv = true.
# If the SS velocity components are uniform then enter
# the components of the specific discharge (m/s) for:
# (1) gas phase in the x-direction (horizontal)
# (2) gas phase in the z-direction (vertical)
# (3) aqueous phase in the x-direction (horizontal)
# (4) aqueous phase in the z-direction (vertical)
#
# Field 5 - Nonuniform velocity components.
# Required only if lcv = false.
# Otherwise if the velocity components are nonuniform, then for
# each node or element (depending if nodal or element velocities are
# used) provide one line of data giving the node or element number and
# the 4 components listed above.
#
#==== BLOCK P: INITIAL NAPL SATURATION AND COMPOSITION =====
#
# Field 1 - Elements containing NAPL saturation.
# Enter the number of elements (inoel) containing NAPL.
# A number less than zero indicates that a NAPL saturation is
# uniform and contained in all elements
#
# for examples 1 and 2
# 505
# for example 3
# 651
#
# Field 2 - Uniform NAPL saturation and composition.
# Enter:
# (1) The uniform nodal NAPL saturation;
# (2) NAPL mole fraction of each organic component.
# There must be ipt(15) mole fractions specified.
# Mole fractions are entered in sequential order (i.e. component
# numbers 1 to ipt(15). The mole fractions must sum to 1.
#
# Field 3 - Non uniform NAPL saturation and composition.
# If inoel>0, then for each element containing NAPL provide the
# following information:
# (1) element number
# (2) The uniform nodal NAPL saturation;
# (3) NAPL mole fraction of each organic component.
# There must be ipt(15) mole fractions specified.
# Mole fractions are entered in sequential order (i.e. component
# numbers 1 to ipt(15). The mole fractions must sum to 1.
#
# for examples 1 and 2 - this is a partial file.
# 1 .531656E-02 .1000E+01
# 2 .562525E-02 .1000E+01
# ...
# 572 .566822E-09 .1000E+01
# 573 .566507E-09 .1000E+01
# for example 3 - this is a partial file.
# 1 0.326712E-01 0.5000E+00 0.5000E+00
# 2 0.285502E-01 0.5000E+00 0.5000E+00
# ...
# 2168 0.703352E-15 0.5000E+00 0.5000E+00
# 2169 0.696703E-15 0.5000E+00 0.5000E+00
#
#==== BLOCK Q: OXYGEN AND NUTRIENT INITIAL CONDI-
# TIONS =====
#
# Field 1 - Gas phase initial conditions.
# Read a logical variable (lunfx) indicating if the gas phase initial
# conditions for oxygen and/or nutrient are uniform.
# Skip this input if oxygen is absent from the gas phase (i.e. the
# oxygen vapor is assigned a negative value)
#
# for examples 2 and 3
# t
#
# Field 2 - Uniform gas phase conditions.
# Enter the uniform gas phase initial conditions for oxygen and
# nutrient.
# Initial conditions are read as partial pressures (i.e. mole
# fractions). Nutrient can only be present if oxygen is present.
# (1) the uniform oxygen partial pressure in the gas phase
# (2) the uniform nutrient partial pressure in the gas phase. This
# item is omitted if nutrient is absent.

```

```

#
# for examples 2 and 3
# 0.20d0
#
# Field 3 - Nonuniform gas phase initial conditions for oxygen and
# nutrient.
# If lunfx=F, then for each node enter:
# (1) node number;
# (2) the uniform oxygen partial pressure in the gas phase
# (3) the uniform nutrient partial pressure in the gas phase
# Nodes need not be in order.
#
# Field 4 - Aqueous phase initial conditions.
# Read a logical variable indicating if the aqueous phase initial
# conditions are uniform.
# Skip this input if oxygen is absent from the aqueous phase (i.e. the
# oxygen aqueous solubility is assigned a negative value)
#
# for examples 2 and 3
# t
#
# Field 5 - Uniform aqueous phase conditions.
# Read the aqueous phase initial conditions for oxygen and nutrient.
# Initial conditions are read as concentrations (g/L). Nutrient
# can only be present if oxygen is present.
# If lunfx=T, then enter:
# (1) the uniform oxygen concentration (g/L) in the aqueous phase
# (2) the uniform nutrient concentration (g/L) in the aqueous phase
#
# for examples 2 and 3
# 0.0090d0
#
# Nonuniform aqueous phase initial conditions for oxygen and nutrient.
# If lunfx=F, then for each node enter:
# (1) node number;
# (2) the uniform oxygen concentration (g/L) in the aqueous phase
# (3) the uniform nutrient concentration (g/L) in the aqueous phase
# Nodes need not be in order.
#
#==== BLOCK R: BOUNDARY CONDITIONS =====
#
# Field 1 - Constant gas pressure nodes equivalent to the initial
# pressure.
# (1) Enter the number of nodes with a constant gas pressure equal
# to the initial gas pressure.
# (2) If nonzero, then starting on a new line, specify the node
# number of all such nodes.
#
# for example 1
26
457 476 495 514 533 552 571 590 591 592 593 594 595 596 597 598
599 600 601 602 603 604 605 606 607 608
# for example 2
# 34
# 742
# 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760
# 457 476 495 514 533 552 571 590 609 628 647 666 685 704 723
# for example 3
# 76
# 22 23 24 25 26 27 28 29 30
# 31 32 33 34 35 36 37 38 39 40
# 41 82 123 164 205 246 287 328 369 410
# 451 492 533 574 615 656 697 738 779 820
# 861 902 943 984 1025 1066 1107 1148 1189 1230
# 1271 1312 1353 1394 1435 1476 1517 1558 1599 1640
# 1681 1722 1763 1804 1845 1886 1927 1968 2009 2050
# 2091 2132 2173 2214 2255 2296 2337
#
# Field 2 - Constant gas pressure nodes different from the initial
# pressure.
# (1) Enter the number of nodes with a constant gas pressure that is
# different from the initial gas pressure.
# (2) If nonzero, then for each such node provide one line of data
# giving the node number and constant gas pressure (Pa gauge).
#
# for examples 1, 2 and 3
0
#

```

```

# Field 3 - Constant aqueous pressure nodes equivalent to the initial
# pressure.
# (1) Enter the number of nodes with a constant aqueous pressure
# equal to the initial gas pressure.
# (2) If nonzero, then starting on a new line, specify the node
# number of all such nodes.
#
# for example 1
26
457 476 495 514 533 552 571 590 591 592 593 594 595 596 597 598
599 600 601 602 603 604 605 606 607 608
# for example 2
# 34
# 742
# 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760
# 457 476 495 514 533 552 571 590 609 628 647 666 685 704 723
# for example 3
# 76
# 22 23 24 25 26 27 28 29 30
# 31 32 33 34 35 36 37 38 39 40
# 41 82 123 164 205 246 287 328 369 410
# 451 492 533 574 615 656 697 738 779 820
# 861 902 943 984 1025 1066 1107 1148 1189 1230
# 1271 1312 1353 1394 1435 1476 1517 1558 1599 1640
# 1681 1722 1763 1804 1845 1886 1927 1968 2009 2050
# 2091 2132 2173 2214 2255 2296 2337
#
# Field 4 - Constant aqueous pressure nodes different from the initial
# pressure.
# (1) Enter the number of nodes with a constant aqueous pressure
# that is different from the initial gas pressure.
# (2) If nonzero, then for each such node provide one line of data
# giving the node number and constant aqueous pressure (Pa gauge)
#
# for examples 1, 2 and 3
0
#
# Field 5 - Gas phase component boundary conditions.
# (1) Enter the number of nodes for which gas phase component
# boundary conditions are specified.
# (2) For each such node, starting on a new line
# enter the following information:
# (2a) the node number
# (2b) an integer variable indicating the boundary condition
# type for all gas phase components at the node.
# 1 = constant mole fraction
# 2 = specified diffusive flux
# 3 = mixed type (contact with a known fluid).
# (2c) the boundary condition values for each component in the
# gas phase. The values are listed in sequential order
# corresponding to the component numbers. Only components
# that are present in the gas phase are listed. Component
# boundary conditions are not provided for components which
# are excluded from the gas phase (i.e. negative vapor
# pressure). Two values are needed for each boundary node
# component. These values are used as needed to specify the
# boundary condition.
# 1 = specified gas phase concentration (partial pressure)
# in contacting fluid. The partial pressures must sum
# to one (used for first type boundary).
# 2 = user supplied value of Dm/length.
#
# for example 1
35
4 2 0.d0 0.d0 1.d0 0.d0
5 2 0.d0 0.d0 1.d0 0.d0
6 2 0.d0 0.d0 1.d0 0.d0
7 2 0.d0 0.d0 1.d0 0.d0
8 2 0.d0 0.d0 1.d0 0.d0
9 2 0.d0 0.d0 1.d0 0.d0
10 2 0.d0 0.d0 1.d0 0.d0
11 2 0.d0 0.d0 1.d0 0.d0
12 2 0.d0 0.d0 1.d0 0.d0
457 3 0.d0 0.d0 1.d0 0.d0
476 3 0.d0 0.d0 1.d0 0.d0
495 3 0.d0 0.d0 1.d0 0.d0
514 3 0.d0 0.d0 1.d0 0.d0
533 3 0.d0 0.d0 1.d0 0.d0

```

```

552 3 0.d0 0.d0 1.d0 0.d0
571 3 0.d0 0.d0 1.d0 0.d0
590 3 0.d0 0.d0 1.d0 0.d0
591 3 0.d0 0.d0 1.d0 0.d0
592 3 0.d0 0.d0 1.d0 0.d0
593 3 0.d0 0.d0 1.d0 0.d0
594 3 0.d0 0.d0 1.d0 0.d0
595 3 0.d0 0.d0 1.d0 0.d0
596 3 0.d0 0.d0 1.d0 0.d0
597 3 0.d0 0.d0 1.d0 0.d0
598 3 0.d0 0.d0 1.d0 0.d0
599 3 0.d0 0.d0 1.d0 0.d0
600 3 0.d0 0.d0 1.d0 0.d0
601 3 0.d0 0.d0 1.d0 0.d0
602 3 0.d0 0.d0 1.d0 0.d0
603 3 0.d0 0.d0 1.d0 0.d0
604 3 0.d0 0.d0 1.d0 0.d0
605 3 0.d0 0.d0 1.d0 0.d0
606 3 0.d0 0.d0 1.d0 0.d0
607 3 0.d0 0.d0 1.d0 0.d0
608 3 0.d0 0.d0 1.d0 0.d0
# for example 2
# 47
# 4 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 5 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 6 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 7 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 8 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 9 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 10 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 11 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 12 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 13 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 14 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 15 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 16 3 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 742 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 743 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 744 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 745 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 746 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 747 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 748 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 749 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 750 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 751 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 752 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 753 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 754 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 755 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 756 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 757 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 758 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 759 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 760 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 457 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 476 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 495 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 514 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 533 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 552 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 571 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 590 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 609 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 628 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 647 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 666 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 685 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 704 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 723 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# for example 3
# 33
# 22 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 23 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 24 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 25 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 26 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 27 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0

```

```

# 28 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 29 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 30 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 31 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 32 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 33 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 34 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 35 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 36 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 37 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 38 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 39 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 40 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 41 2 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 493 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 534 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 575 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 616 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 657 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 698 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 739 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 780 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 821 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 862 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 903 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 944 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
# 985 3 0.d0 0.d0 0.d0 0.d0 0.2d0 0.d0 0.8d0 0.d0
#

```

```

# Field 6 - Aqueous phase component boundary conditions.
# (1) Enter the number of nodes for which aqueous phase component
# boundary conditions are specified.
# (2) For each such node, starting on a new line
# enter the following information:
# (2a) the node number
# (2b) an integer variable indicating the boundary condition type
# for all aqueous phase components at the node.
# 1 = constant mole fraction
# 2 = specified diffusive flux
# 3 = mixed type (contact with a known fluid).
# (2c) the boundary condition values for each component in the
# gas phase. The values are listed in sequential order
# corresponding to the component numbers. Only components
# that are present in the aqueous phase are listed.
# Component boundary conditions are not provided for
# components which are excluded from the aqueous phase
# (i.e. negative solubility). Two values are needed for each
# boundary node component. These values are used as needed
# to specify the boundary condition.
# 1 = specified aqueous phase concentration (g/l)
# in contacting fluid (used for first type boundary).
# 2 = user supplied value of Dm/length.
# Note: use value of 1.d0 for water
#

```

```

# for example 1
35
4 2 0.d0 0.d0 0.d0 0.d0
5 2 0.d0 0.d0 0.d0 0.d0
6 2 0.d0 0.d0 0.d0 0.d0
7 2 0.d0 0.d0 0.d0 0.d0
8 2 0.d0 0.d0 0.d0 0.d0
9 2 0.d0 0.d0 0.d0 0.d0
10 2 0.d0 0.d0 0.d0 0.d0
11 2 0.d0 0.d0 0.d0 0.d0
12 2 0.d0 0.d0 0.d0 0.d0
457 3 0.d0 0.d0 1.d0 0.d0
476 3 0.d0 0.d0 1.d0 0.d0
495 3 0.d0 0.d0 1.d0 0.d0
514 3 0.d0 0.d0 1.d0 0.d0
533 3 0.d0 0.d0 1.d0 0.d0
552 3 0.d0 0.d0 1.d0 0.d0
571 3 0.d0 0.d0 1.d0 0.d0
590 3 0.d0 0.d0 1.d0 0.d0
591 3 0.d0 0.d0 1.d0 0.d0
592 3 0.d0 0.d0 1.d0 0.d0
593 3 0.d0 0.d0 1.d0 0.d0
594 3 0.d0 0.d0 1.d0 0.d0
595 3 0.d0 0.d0 1.d0 0.d0
596 3 0.d0 0.d0 1.d0 0.d0

```



```

597 3 0.d0 0.d0 1.d0 0.d0
598 3 0.d0 0.d0 1.d0 0.d0
599 3 0.d0 0.d0 1.d0 0.d0
600 3 0.d0 0.d0 1.d0 0.d0
601 3 0.d0 0.d0 1.d0 0.d0
602 3 0.d0 0.d0 1.d0 0.d0
603 3 0.d0 0.d0 1.d0 0.d0
604 3 0.d0 0.d0 1.d0 0.d0
605 3 0.d0 0.d0 1.d0 0.d0
606 3 0.d0 0.d0 1.d0 0.d0
607 3 0.d0 0.d0 1.d0 0.d0
608 3 0.d0 0.d0 1.d0 0.d0
# for example 2
# 47
# 4 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 5 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 6 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 7 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 8 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 9 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 10 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 11 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 12 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 13 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 14 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 15 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 16 3 0.d0 0.d0 1.d0 0.d0 0.009d0 0.d0
# 742 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 743 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 744 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 745 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 746 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 747 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 748 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 749 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 750 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 751 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 752 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 753 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 754 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 755 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 756 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 757 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 758 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 759 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 760 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 457 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 476 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 495 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 514 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 533 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 552 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 571 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 590 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 609 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 628 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 647 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 666 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 685 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 704 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# 723 2 0.d0 0.d0 0.d0 0.d0 0.d0 0.d0
# for example 3
# 33
# 22 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 23 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 24 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 25 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 26 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 27 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 28 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 29 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 30 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 31 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 32 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 33 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 34 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 35 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 36 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0

```

```

# 37 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 38 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 39 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 40 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 41 2 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 493 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 534 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 575 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 616 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 657 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 698 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 739 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 780 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 821 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 862 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 903 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 944 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
# 985 3 0.d0 0.d0 0.d0 0.d0 1.d0 0.0d0 0.009d0 0.d0
#
# Field 7 - Gas phase boundary flux.
# Enter two lines of data:
# (1) Enter the number of nodes with a constant gas phase volumetric
# flux.
# (2) If nonzero, then for each such node provide one line of data
# giving the node number and gas phase flux referenced to
# atmospheric pressure and the steady temperature at the
# injected node (m3/s).
#
# for examples 1, 2 and 3
0
#
# Field 8 - Aqueous phase boundary flux.
# Enter two lines of data:
# (1) Enter the number of nodes with a constant aqueous phase
# volumetric flux.
# (2) If nonzero, then for each such node provide one line of data
# giving the node number and aqueous phase flux (m3/s).
#
# for examples 1, 2 and 3
0
#
===== BLOCK S: EXTRACTION / INJECTION WELL =====
#
# Field 1 - Include an extraction / injection well.
# Enter a logical variable (LCTRL(12)) indicating if an
# extraction/injection well is to be simulated.
#
# for examples 1, 2 and 3
t
# for example 3, generation of initial condition
# f
# Field 2 - Extraction / injection rate.
# Enter the total volumetric extraction/injection rate (scfm). A
# negative value indicates extraction, and a positive value indicates
# injection.
# Required only if LCTRL(12)=F
#
# for example 1
-100.d0
# for examples 2 and 3
# 1.0d0
# for example 3, generation of initial condition
# 0.0d0
#
# Field 3 - Well coordinates.
# Specify:
# (1) the well radius (m);
# (2) two integers identifying the minimum and maximum node
# numbers along the vertical well screen.
# Note all nodes along the well screen must have a horizontal
# coordinate equal to the well radius
# Not required if LCTRL(12)=F
#
# for example 1
0.25d0 4 12
# for example 2
# 0.25d0 4 16
# for example 3

```

```
# 0.25d0 493 985
#
#===== BLOCK R: VELOCITY BOUNDARY CONDITIONS =====
#
# Field 1 - Bottom boundary of domain.
# Enter a logical variable (LCTRL(28)) if the bottom boundary is
# impervious.
#
# for examples 1, 2 and 3
t
#
# Field 2 - R.H.S. boundary of domain.
# Enter a logical variable (LCTRL(29)) if the R.H.S. boundary is
# impervious.
#
# for examples 1, 2 and 3
f
#
# Field 3 - L.H.S. boundary of domain.
# Enter a logical variable (LCTRL(30)) if the L.H.S. boundary is
# impervious. Note that this boundary will be adjusted in the
# presence of a well
```

```
#
# for examples 1, 2 and 3
t
#
# Field 4 - Top boundary of domain.
# Enter a logical variable (LCTRL(31)) if the top boundary is
# impervious.
#
# for examples 1, 2 and 3
t
#
# if lctrl(31) is true read the length of the cap, CAPLEN. The cap is
# assumed to extend from the well to the input value.
# Not required if LCTRL(31)=F
#
# for examples 1 and 2
10.0d0
# for example 3
9.585d0
END OF DATA2
```

Appendix H

SOURCE CODE LISTING

Following is a source code listing of MISER. The main program is given first followed by all other routines in alphabetical order. The Harwell sparse matrix package is excluded.

Main Routine - miser.f

```

C-----
C
C MISER.f - Numerical model of two-dimensional multiphase,
C multicomponent flow and transport. Designed
C for soil vapor extraction (SVE) and bioventing
C (BV) simulations. Nonequilibrium interphase
C mass exchange and kinetic bioreaction terms are
C included. Version 1.0 - nonadaptive.
C
C Computational domain: xz, rz
C
C Project directed by: L.M. Abriola
C
C Written by: J.R. Lang and K.M. Rathfelder
C
C Version 1.0, January, 1997
C
C Required Control Flags:
C
C ipt(0) - number of elements
C ipt(1) - number of nodes
C ipt(2) - number of stacked variables
C ipt(25) - print results every ipt(25) time steps if lprnt(0)
C is true
C ipt(30) - maximum number of time steps
C
C Control Flags computed internally in routine:
C
C lctrl(14) - logical variable controlling coupling of flow and
C transport solutions
C lctrl(14) = .true. - exchange couples flow and
C transport solutions
C lctrl(14) = .false. - flow and transport
C solutions not coupled
C
C-----
C program MISER
C
C Include parameter and type declarations, common block definitions,
C and dimension statements.
C
C include 'dimen.inc'
C character*20 infile(4),outpre,outfile(8+ncmp)
C
C Declare and define common block variables.
C
C common /cb2/ p(nn3)
C common /cb2b/ pt(nn3)
C common /cb3/ sat(nnstk3)
C common /cb3b/ satt(nnstk3)
C common /cb8/ vis(nnmx),pmw(nn3)
C common /cb9/ xmft(nmf)
C
C common /cb9b/ xmft(nmf)
C common /cb10/ den(nn6)
C common /cb10b/ dden(nn6),pmwt(nn3),dent(nn6)
C common /cb11/ pex(nns10),rxnp(nn2)
C common /cb62/ rxn(nmf),cex(nmfs)
C common /cb62b/ rhsex(nmfs)
C common /cb90/ infile,outpre,outfile
C dimension pkeep(nnstk2)
C
C Open input devices. The input and output devices are summarized
C as follows: input: 11, file='infile(1)' - input for INPUT1.f,
C read from console input
C 13, file='infile(2)' - input for INPUT2.f,
C read from ifile(1) by INPUT1.f
C 14, file='infile(3)' - input for ERRMESSAGE.f,
C read from infile(1) by INPUT1.f
C 28, file='infile(4)' - input for restarts,
C read from infile(2) by INPUT2.f
C output: 21, file='outpre.out' - echo of input,'outpre'
C read from infile(1) by INPUT1.f
C 22, file='outpre.err' - error messages,
C 'outpre' read from infile(1) by INPUT1.f
C 23, file='outpre.cnv' - convergence data when
C ipt(28).gt. 0,
C 'outpre' read from infile(1) by INPUT1.f
C 24, file='outpre.con' - plotting data
C 'outpre' read from infile(1) by INPUT1.f
C 25, file='outpre.mb' - mass balance data,
C 'outpre' read from infile(1) by INPUT1.f
C 26, file='outpre.plt' - time series plot,
C 'outpre' read from infile(1) by INPUT1.f
C 27, file='outpre.rst' - restart data,
C 'outpre' read from infile(1) by INPUT1.f
C
C open (11,file='miser.d1',status='unknown')
C infile(1)='miser.d1'
C 501 format (a)
C
C Read the input from main input file.
C
C call INPUT1
C
C Initialize vectors as needed.
C
C lkeep = .false.
C do 1 i = 1,10*ipt(2)
C 1 pex(i) = zero
C do 2 i = 1,(ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(7)-1)*ipt(2)
C cex(i) = zero
C 2 rhsex(i) = zero
C do 3 i = 1, ipt(1)*3
C dent(i+3*ipt(1)) = zero

```

```

3  dden(i) = zer0
  do 4 i = 1, ipt(1)
    pmwt(i) = pmw(i)
4  rxn(i) = zer0
  if (.not.lprnt(0).and.(int(t(3)/t(12)).gt.0)) then
    lprn = .true.
  else
    lprn = .false.
  end if
  if (lprnt(6).and..not.lprnt(25).and.(int(t(3)/t(27)).gt.0))
+  then
    lpcbal = .true.
  else
    lpcbal = .false.
  end if
C
C— Finish input with INPUT2.f.
C
C  call INPUT2
C
C— Use ATRI.f to compute element areas.
C
C  call ATRI
C
C— Initialize the simulation time, time step, and time step number
C— if this run is not a restart.
C
if(.not.lctrl(26)) then
  t(9) = zer0      ! initialize the simulation time
  t(8) = t(3)      ! initialize the time step
  istart = 1
else
  istart = ipt(76) + 1
end if
C
C— Initialize variables.
C
lskip = .false.
itskip = 0
tskip = zer0
do 33 i = 1, ipt(49)
33  pkeep(i)=zer0
  ipt(36)=0
  ipt(37)=0
  ipt(38)=0
  if(.not.lctrl(1)) ipt(36)=1
  if(.not.lctrl(2)) ipt(37)=1
  if(.not.lctrl(24)) ipt(38)=1
C
C— Loop over maximum number of time steps.
C
if (ipt(29).ne.0) write (ipt(29),*) 'Starting Computation Loop'
C
C— Perform initial mass balance calculation.
C
if(.not.lctrl(32).and.lprnt(6))
+  call CBAL(0,.true.,.true.,.true.,.true.)
C
C— Initial starting time.
C
c  call TSYS(time0)
  do 10 its = istart, ipt(30)
    itskip = itskip + 1
C
C— Save variables from previous time step: pressures, saturations,
C— mole fractions, molecular weights.
C
  do 5 i = 1, 3*ipt(1)
5    pt(i) = p(i)
  do 6 i = 1, 3*ipt(2)
6    sat(i) = sat(i)
  do 7 i = 1, ipt(1)*(ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(7))
7    xmf(i) = xmf(i)
C
C— Solve the phase mass balance equations. This is not
C— needed when a steady-state flow/saturation field has been input.
C— When the component mass balance equations are also solved,
C— update the Darcy velocities.

```

```

C
if(mod(itskip.ipt(85)+1).eq.0) then
  if(ipt(85).ne.0) then
    tkeep = t(8)
    t(8) = t(8) + tskip
    w1 = t(8)/tskip
    w2 = rone - w1
    do 44 i = 1, ipt(49)
44    pex(ipt(52)+i) = w1*pex(ipt(52)+i) + w2*pkeep(i)
  end if
end if
100 t(9) = t(9) + t(8) ! increment the current simulation time
if(mod(itskip.ipt(85)+1).eq.0) then
  if(lctrl(14).and.its.eq.1) then
    lctrl(14) = .false.
    lkeep = .true.
  end if
  if (lctrl(1)) then
    if (lctrl(2).and.its.gt.1) call MQLLEWT
    call FLOW (its,iter)
    inter = 1
    if(ipt(36).lt.0.and.ipt(85).gt.0) then
      ipt(36) = 0
46    t(8) = t(8)/2.0d0
      if(t(8).lt.t(4)) then
        write(ipt(28),500) t(4)
        stop
      end if
      inter = 2*inter
      do 47 i = 1, inter
        if(i.gt.1) call CBAL(its,.true.
+        ,.false.,.false.,i.eq.1)
47    if(ipt(36).ge.0) call FLOW(its,iter)
        if(ipt(36).lt.0) goto 46
      end if
      if(ipt(36).ge.0) then
        call CBAL(its,.true.,.false.,.false.,inter.eq.1)
        inter = 1
      end if
      if(lkeep) then
        lctrl(14) = .true.
        lkeep = .false.
      end if
      if (lctrl(2).and.ipt(36).gt.0) then
        call BCFLOW
        call VEL
      end if
      end if
      if(ipt(85).ne.0) t(8) = tkeep
    else
      write (ipt(29),*) 'Skipping Flow Computation'
      tskip = tskip + t(8)
      w1 = t(8)/tskip
      w2 = rone - w1
      do 66 i = 1, ipt(49)
66    if(.not.lskip) pkeep(i) = w1*pex(ipt(52)+i) + w2*pkeep(i)
        if(lskip) lskip = .false.
      end if
C
C— Save the values of the density vectors after pressure effects
C— have been included for calculation of the compositional density
C— derivatives. Save the phase molecular weights before compositional
C— effects are included.
C
if(mod(itskip.ipt(85)+1).eq.0) then
  do 12 i=1, ipt(1)
12    pmwt(i) = pmw(i)
  do 13 i=1, 6*ipt(1)
13    dent(i) = den(i)
  end if
C
C— Use TRANS.f to solve the component mass balance equations. This
C— is not needed when only the multiphase flow field is being
C— calculated.
C
if (lctrl(2).and.(ipt(36).ge.0))
+  call TRANS (its,ibconv)
C

```

C— Check for time step reduction in solution of flow or transport
 C— eqs. Adjust time step size and re-solve current time step.

```
C
if ((ipt(36).lt.0).or.(ipt(37).lt.0).or.(ipt(38).lt.0)) then
  if (t(8).le.t(4)) then ! If dt = dtmin, don't reduce dt.
    write (ipt(28),500) t(4) ! Print error and terminate.
    stop
  endif
  t(9) = t(9)-t(8) ! Return simulation time to old value.
  tskip = tskip-t(8) ! Reset cumulative time step
  t(8) = t(8)*t(7) ! Decrease time step.
  itskip = itskip - 1 ! Reset flow skipping counter
  lskip = .true.
  if (t(8).lt.t(4)) t(8)=t(4) ! If dt< dtmin, set dt=dtmin.
  if (lctrl(1)) then ! Reset pressure and saturation values.
    if(mod(itskip,ipt(85)+1).eq.0) then
      do 15 i = 1,3*ipt(1)
        p(i) = pt(i)
      end if
      do 16 i = 1,3*ipt(2)
        sat(i) = satt(i)
      end if
      call SATW
    endif
    if(lprn) lprn = .false. ! don't print after next iteration.
    goto 100 ! Re-solve flow and transport with new time step.
  endif
endif
```

C
 C— Reset skipping variables.

```
C
if(mod(itskip,ipt(85)+1).eq.0) then
  tskip = zer0
  do 77 i = 1, ipt(49)
    pkeep(i) = zer0
  end if
end if
```

C
 C— Check for time step increase.

```
C
itsum = ipt(36)+ipt(37)+ipt(38)
101 if(itsum.ge.3) then
  t(8) = t(8) * t(6) ! increase time step
  if (t(8).gt.t(5)) t(8)=t(5) ! constrain time step to dtmax
endif
```

C
 C— Always print at end of simulation.

```
C
if (t(9) .ge. t(2)) then
```

C
 C— Determine execution time of simulation.

```
C
c call TSYS(tend)
c write(*,*) tend-time0
c call prnt(its)
stop
endif
```

C
 C— Perform mass balance calculation if desired.

```
C
if(lprnt(6)) then
  if(lprnt(25)) then
    if(mod(its,ipt(83)) .eq. 0) lpcbal = .true.
    call CBAL(its,.not.lctrl(1))
  + ,lctrl(2),lpcbal,.not.lctrl(1))
  lpcbal = .false.
  else if(.not.lprnt(25)) then
    call CBAL(its,.not.lctrl(1))
```

```
+ ,lctrl(2),lpcbal,.not.lctrl(1))
if(lpcbal) then
  t(8) = t(3) ! reset time step to previous value
  lpcbal = .false.
endif
if(int((t(9)+t(8))/t(27)).gt.int(t(9)/t(27))) then
  lpcbal = .true.
  t(3) = t(8)
  t(8) = t(8) - (t(9) + t(8)
+ - int((t(9)+t(8))/t(27))*t(27))
endif
if (t(9)+t(8) .gt. t(2)) then !Does time exceed tmax?
  t(8) = t(2) - t(9) !Reduce time step so time=tmax.
  lpcbal = .true.
endif
endif
```

C
 C— Output results at intermediate times.

```
C
if (lprnt(0)) then ! output results after 'ipt(25)' time steps
  if (mod(its,ipt(25)) .eq. 0) then
    call prnt (its)
  endif
else ! output results after a time increment of 't(11)'
  if (lprn) then
    call prnt (its)
    t(8) = t(3) ! reset time step to previous value
    lprn = .false.
  end if
  ! check if print occurs at next time step
  if (int((t(9)+t(8))/t(12)) .gt. int(t(9)/t(12))) then
    if(lpcbal) then
      if(t(9)+t(8)-int((t(9)+t(8))/t(12))*t(12).gt.zer0)
+ lpcbal = .false.
      else
        t(3) = t(8)
        end if
        lprn = .true.
        t(8)=t(8)-(t(9) + t(8) - int((t(9)+t(8))/t(12))*t(12))
        end if
        if (t(9)+t(8) .gt. t(2)) then ! Does time exceed tmax?
          t(8) = t(2) - t(9) ! Reduce time step so time=tmax.
          lprn = .true.
        endif
      endif
    endif
  endif
```

C
 C— Loop to next time step.

```
C
10 continue
```

C
 C— Print the results and determine the execution time if the maximum
 C— number of time steps is reached.

```
C
c call TSYS(tend)
c write(*,*) tend-time0
call prnt(its)
stop
```

C
 C— Formats:

```
500 format (/ 'Solution failed to converge'/
+ ' at the minimum time step size =',e12.4)
502 format (/ 'Maximum iterations of ',i4,' between solution of '/
+ ' flow and transport equations was exceeded')
end
```

Subroutine - atri.f

```

C-----
C
C ATRI.f - Subroutine which computes the area of each element.
C These areas are used in the element matrices for both
C the flow and transport equations. Nodes are numbered
C counterclockwise in an element starting with an
C arbitrary node when z is positive downwards. This
C routine stops program execution if one or more of the
C element areas is less than zero. In this case, all
C the element areas are printed out, as well as more
C detailed information about the problem elements.
C-----
C Required Control Flags:
C
C ipt(27) - integer variable indicating type of domain
C ipt(27) = 0 - xz domain
C ipt(27) = 1 - rz domain
C lctrl(2) - logical variable controlling presence of transport
C solution
C lctrl(2) = .true. - compute transport solution
C lctrl(2) = .false. - skip transport solution
C-----
C subroutine ATRI
C include 'dimen.inc'
C-----
C Declare and define common block variables.
C
C common /cb1/ matel(nelmx),nodel(nel3),nodept(nnmx),nelpt(nel3),
C + matp(nn6)
C common /cb1c/ xnode(nnmx),znode(nnmx),rbar(nelmx),area(nelmx)
C common /cb1e/ aby12(nelmx),aby30(nelmx)
C-----
C Loop over the elements.
C
C do 100 i=1,ipt(0)
C-----
C Compute areas by taking the standard area coordinate determinant,
C but subtract row 1 from rows 2 and 3 before taking the determinant

```

Subroutine - bcflux.f

```

C-----
C
C BCFLUX.f - Subroutine which computes the boundary fluxes using
C the flow solution.
C-----
C Required Control Flags:
C
C lctrl(12) - logical variable denoting presence of a well
C lctrl(12) = .true. - well present
C lctrl(12) = .false. - well not present
C-----
C subroutine BCFLUX
C include 'dimen.inc'
C common /cb2/ p(nn3)
C common /cb6c/ temp(nnmx)
C common /cb30/ ibc(nnmx)
C common /cb31/ source(nn2)
C common /cb32/ bcf(nn2)
C common /cb42/ amb(icnl),fmb(ksolve)
C-----
C Initial variables.
C
C do 100 i = 1, ipt(40)
C 100 bcf(i) = zero
C-----
C Compute gas phase boundary fluxes at all specified first type
C boundary nodes.
C

```

```

C----- (i.e. area=det[L]/2).
C
C i3=i*3
C n3=nodel(i3)
C n2=nodel(i3-1)
C n1=nodel(i3-2)
C x1=xnode(n1)
C x2=xnode(n2)
C x3=xnode(n3)
C z1=znodel(n1)
C z2=znodel(n2)
C z3=znodel(n3)
C area(i)=((x2-x1)*(z3-z1)-(x3-x1)*(z2-z1))/2.0d0
C if (area(i).le.zerf) Call ErrMessage (46,0,ipt(29))
C 100 continue
C-----
C Compute the radial centroid of each element for axisymmetric
C coordinates. This is set to one if the xz coordinates are used.
C
C if (ipt(27) .eq. 0) then
C do 151 i = 1,ipt(0)
C 151 rbar(i) = rone
C else
C do 152 i = 1,ipt(0)
C i3 = 3*(i-1)
C 152 rbar(i) = pthird * (xnode(nodel(i3+1)) +
C + xnode(nodel(i3+2)) + xnode(nodel(i3+3)))
C endif
C-----
C Compute terms needed in component balance solution.
C
C if(lctrl(2)) then
C do 150 i = 1,ipt(0)
C aby12(i) = rbar(i)*area(i)/12.0d0
C 150 aby30(i) = rbar(i)*area(i)/30.0d0
C endif
C-----
C return
C end

```

```

C if (ipt(18) .gt. 0) then
C do 300 jj = 1,ipt(18)
C irowd2 = ibc(jj)
C irow = 2 * irowd2
C 300 bcf(irowd2) = fmb(irow)
C endif
C-----
C Compute aqueous phase boundary fluxes at all specified first type
C boundary nodes.
C
C if (ipt(19) .gt. 0) then
C do 320 jj = 1,ipt(19)
C jjj = ipt(18) + jj
C irowd2 = ibc(jjj)
C irow = 2*irowd2 - 1
C irowd2 = irowd2 + ipt(1)
C 320 bcf(irowd2) = fmb(irow)
C endif
C-----
C Compute gas phase boundary fluxes at all nodes with constant flux.
C
C if (ipt(22) .gt. 0) then
C ii = ipt(62)
C do 340 i = 1,ipt(22)
C ig1 = ibc(ii+i)
C 340 bcf(ig1) = source(i) * patm * temp(ig1)
C + / (( patm + p(ig1) ) * tstd)
C endif
C-----
C

```

C— Compute aqueous phase boundary fluxes at all nodes with constant
C— flux.

```
C
if (ipt(23) .gt. 0) then
  ii = ii + ipt(22)
  do 350 i = 1, ipt(23)
    ia1 = ipt(1) + ibc(ii+i)
    iii = ipt(22) + i
350    bcf(ia1) = source(iii)
  endif
```

C
C— Compute gas phase boundary fluxes at all well nodes.

```
C
if (lctrl(12)) then
  ii = ipt(64)
  jstrt = ipt(22) + ipt(23)
  do 360 i = 1, ipt(24)
```

```
    ig1 = ibc(ii+i)
    bcf(ig1) = source(jstrt+i) * patm * temp(ig1)
    + / (( patm + p(ig1) ) * tstd )
C
C— Compute aqueous phase boundary fluxes at all well nodes when
C— operating in extraction mode.
C
if (qwell .lt. zer0) then
  ia1 = ipt(1) + ibc(ii+i)
  jstrt2 = jstrt + ipt(24) + i
  bcf(ia1) = source(jstrt2)
endif
360 continue
endif
return
end
```

Subroutine - bio.f

C
C
C BIO.f - Subroutine which computes the biological reaction
C terms using Monod kinetics. Also update the biophase
C mole fractions using the finite element method by
C computing a mole balance if a biophase is considered.
C The equations are solved sequentially at each node
C until convergence. Only bioreactive components are
C included in the biophase.

C Arguments: iconv - integer flag for global convergence
C ibconv - integer flag for bioreaction convergence

C Required Control Flags:

C t(15) - convergence criterion for bioreactions
C ipt(39) - integer flag determining the kinetics type
C 1 - standard monod kinetics
C 2 - monod kinetics with substrate inhibition
C 3 - monod kinetics with lumped substrate
C inhibition
C 4 - monod kinetics with saturation dependency
C 5 - monod kinetics with saturation dependency
C and substrate inhibition
C lctrl(8) - logical variable controlling type of FEM
C solution for transport
C lctrl(8) = .true. - use mass lumping
C lctrl(8) = .false. - full FEM solution
C lctrl(9) - logical variable denoting presence of nutrient
C lctrl(9) = .true. - nutrient considered
C lctrl(9) = .false. - nutrient not considered
C lctrl(16) - logical variable denoting method of including
C biological reaction
C lctrl(16) = .true. - include bioreaction in
C aqueous transport
C lctrl(16) = .false. - solve FEM solution for
C rate limited biophase
C lctrl(17) - logical variable indication steady state biomass
C lctrl(17) = .true. - steady state biomass
C lctrl(17) = .false. - transient biomass

```
subroutine BIO(iconv,ibconv)
include 'dimen.inc'
character*10 cname(ncmp)
```

C— Declare and define common block variables.

```
C
common /cb1/ matel(nelmx),nodel(nel3),nodept(nnmx),nelpt(nel3),
+ matpt(nn6)
common /cb1e/ aby12(nelmx),aby30(nelmx)
common /cb3/ sat(nnstk3)
```

```
common /cb6b/ por(nelmx),srw(nnstk)
common /cb6d/ dtemp(nzmax6),idepth(nnmx)
common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
+ chen(ncmp),casol(ncmp),cmdif(ncmp2)
common /cb8/ vis(nnmx),pmw(nn3)
common /cb9/ xmf(nmf)
common /cb9b/ xmf1(nmf)
common /cb10/ den(nn6)
common /cb11/ pex(nns10),rxnp(nn2)
common /cb40/ a(icnl),rhs(ksolve),w(icnl)
common /cb41/ irr(icnl),icn(icnl),iw(icnl,8),ikeep(icnl,5)
common /cb41b/ nbw(0:2),ia
common /cb60/ khalf(ncmp),fuse(ncmp2),umax(ncmp),xyield(ncmp),
+ kinhib(ncmp)
common /cb62/ rxn(nmf),cex(nmfs)
common /cb62b/ rhsex(nmfs)
common /cb63/ kex(ncmp5),kmax(ncmp5)
common /cb64/ bok(nbcmp),bom(nbcmp),krtid(ncmp)
common /cb91/ cname
```

C
C— Dimension local arrays.

```
C
dimension sum(3),xmonod(ncmp),xinhib(ncmp)
+ ,xtmin(ncmp)
```

C— Data the minimum detectable limit.

```
C
data dlimit / 1.0d-9 /
```

C— Set pointers: iptbc points to the start of the biophase section
C— in icp; ib points to the start of biophase phase storage; ibs
C— points to the start of biophase stacked storage.

```
C
iptbc = ipt(60)
ipt1 = ipt(1)
ipt2 = ipt(2)
ig = ipt(3)
C
C— Set the biophase concentrations equal to the corresponding
C— aqueous phase concentrations when no mass transfer rate
C— is considered for aqueous/biophase interactions.
```

```
C
if (lctrl(16)) then
  isum = 0
  do 2 ia = 1, ipt(4)
    if (icp(ig+ia).eq.icp(iptbc+isum+1)) then
      na = ipt(9) + ipt1 * (ia-1)
      nb = ipt(12) + ipt1 * isum
      isum = isum + 1
      do 3 i = 1, ipt1
        xmf(nb+i) = xmf(na+i)
3      end if
```

```

2 continue
end if
ibconv = 0
C
C— Iterate over the nodes to compute the reaction terms..
C
do 100 i = 1, ipt1
C
C— Calculate the lumped organic substrate concentration and lumped
C— solubility for inhibition.
C
if(ipt(39).eq.3) then
subsum = zer0
solsum = zer0
do 101 icb = 1, ipt(17)
ic = icp(iptbc+icb)
if(ipt(39).eq.3) then
if(lctrl(10)) then
solsum = casol(ic) + solsum
else
itmp = (ic-1)*ipt(89)+3*ipt(88)+idepth(i)
solsum = casol(ic) + dtemp(itmp) + solsum
end if
end if
nbc = ipt(12) + (icb-1)*ipt1 + i
101 subsum = subsum + xmf(nbc)
end if
if(lctrl(10)) then
kdt = kd
else
itmp = ipt(65)*ipt(89)+5*ipt(88)+idepth(i)
kdt = kd + dtemp(itmp)
end if
rxnp(i) = zer0
rxnp(i+ipt1) = zer0
C
C— Update the Monod expressions. xmonod(1) to xmonod(ipt(17)) are
C— for the substrates, xmonod(ipt(17)+1) is for the electron
C— acceptor, and xmonod(ipt(17)+2) is for the nutrient when present.
C
do 105 icb = 1, ipt(7)-1
nbc = ipt(12) + (icb-1)*ipt1 + i
ic = icp(iptbc+icb)
xmonod(icb) = xmf(nbc) / ( xmf(nbc)
+
+ khalf(ic)/(cmw(ic)*den(ipt1+i)))
C
C— Include oxygen, nutrient, and substrate inhibition kinetics.
C— Aerobic metabolism is turned off when oxygen is below a threshold
C— concentration. Metabolism is also turned off when the substrate
C— or nutrient concentration is above a threshold concentration
C— Also include a substrate threshold.
C
if(ipt(39).ne.1) then
if(ipt(39).eq.4.or.ipt(39).eq.5) then
C
C— Put a functional expression for saturation dependent inhibition
C— here.
C
+
saterm
=0.10d0+0.90d0*(sat(i+ipt2)-srw(i))/(rone-srw(i))
else
saterm = rone
end if
if(lctrl(10)) then
casolt = casol(ic)
else
itmp = (ic-1)*ipt(89)+3*ipt(88)+idepth(i)
casolt = casol(ic) + dtemp(itmp)
end if
xinhib(ic) = casolt*kinhib(ic)
if(icb.ne.ipt(17)+1) then
inaplc = ipt(15)
xtmin(ic) = (cmw(inaplc+1)/cmw(ic))*dlimit
if(ipt(39).eq.3) then
if(lctrl(9).and.icb.eq.ipt(7)-1) then
xmf(nbc) = xmf(nbc)
else
xmf(nbc) = subsum

```

```

xinhib(ic) = solsum*kinhib(ic)
end if
else
xmf(nbc) = xmf(nbc)
end if
if(xmf(nbc).gt.xinhib(ic)) then
xeff = xinhib(ic)
else if(xmf(nbc).lt.xtmin(ic).and.(.not.lctrl(9).or.
+
+ (lctrl(9).and.icb.ne.ipt(7)-1))) then
xeff = xtmin(ic)
else
xeff = xmf(nbc)
end if
if(lctrl(9).and.(icb.eq.ipt(7)-1))
+
+ xtmin(ic) = zer0
if(ipt(39).eq.2.or.ipt(39).eq.3.or.ipt(39).eq.5)
+
+ xmonod(icb) = xmonod(icb)
+
+ *(rone-xeff/xinhib(ic))
+
+ *(rone-xtmin(ic)/xeff) * saterm
else
if(xmf(nbc).lt.xinhib(ic)) then
xeff = xinhib(ic)
else
xeff = xmf(nbc)
end if
if((ipt(39).eq.2).or(ipt(39).eq.3)
+
+ .or(ipt(39).eq.5))
+
+ xmonod(icb) = xmonod(icb)
+
+ *(rone-xinhib(ic)/xeff)
end if
end if
105 continue
C
C— Update the biomass before updating the biophase concentrations.
C— This form assumes all the organics are biodegradable. ibcx points
C— to the location in xmf of the nodal biomass value (Note: the
C— nodal biomass value is in mass concentration).
C
sum(1) = zer0
if(lctrl(9)) sum(2) = zer0
sum(3) = zer0
do 110 icb = 1, ipt(17)
ic = icp(iptbc+icb)
if(lctrl(10)) then
umaxt = umax(ic)
else
itmp = (ic-1)*ipt(89)+4*ipt(88)+idepth(i)
umaxt = (umax(ic) + dtemp(itmp))
end if
sum(1) = sum(1) + fuse(ic)*umaxt*xmonod(icb)
if(lctrl(9)) sum(2)=
+
+ sum(2)+fuse(ic+ipt(65))*umaxt*xmonod(icb)
110 sum(3) = sum(3) + xyield(ic)*umaxt*xmonod(icb)
do 115 icb = ipt(17)+1, ipt(7)-1
115 sum(3) = sum(3)*xmonod(icb)
nbcx = ipt(12) + (ipt(7)-1)*ipt1 + i
xmfold = xmf(nbcx)
C
C— Include inhibition terms for biomass. This term restricts the
C— biomass between a maximum and minimum level.
C
xterm1 = rone - ( xmf(nbcx) / xbmax )
xterm2 = rone - ( xbmin / xmf(nbcx) )
dxmf = xmf(nbcx) * t(8) * (sum(3)*xterm1-kdt*xterm2)
C
C— Limit bioreaction during initial startup period.
C
if (t(9).lt.t(11)) dxmf = (t(9)/t(11))*dxmf
xmf(nbcx) = xmf(nbcx) + dxmf
if(xmf(nbcx).lt.xbmin) xmf(nbcx) = xbmin
C
C— Use constant biomass if desired.
C
if(lctrl(17)) xmf(nbcx) = xinit
C
C— Update the oxygen/nutrient Monod term.
C
if (lctrl(9)) then

```



```

ibo = ipt(7) - 2
ibn = ipt(7) - 1
term = xmonod(ibo) * xmonod(ibn)
else
  ibo = ipt(7) - 1
  term = xmonod(ibo)
end if
C
C— Update the biophase oxygen and nutrient (if present) mole
C— fractions.
C
isum = 0
do 120 icb = ipt(17)+1, ipt(7)-1
  ic = icp(iptbc+icb)
  isum = isum + 1
  nacrx = icb*ipt(1) + i
  nac = ipt(9) + nacrx
  nbc = ipt(12) + (icb-1)*ipt1 + i
  xmfold = xmf(nbc)
  dxmf = xmf(nbcx) * sum(isum) * term / cmw(ic)
C
C— Limit bioreaction during initial startup period.
C
  if ((t(9)).lt.t(11)) dxmf = (t(9)/t(11))*dxmf
C
C— Update the bio-reaction term.
C
  if(xmf(nbc).ne.zer0) then
    rxn(nac) = -dxmf/xmf(nbc)
    rxn(nbc) = -dxmf/xmf(nbc)
  else
    rxn(nac) = zer0
    rxn(nbc) = zer0
  end if
  rxnp(i) = rxnp(i) - dxmf
  rxnp(i+ipt1) = rxnp(i+ipt1) - dxmf
120 continue
C
C— Update the biophase substrate mole fractions.
C
isum = 0
do 125 ia = 1, ipt(14)
  if(icp(ig+ia).eq.icp(iptbc+isum+1)) then
    nac = ipt(9) + ipt(1) * (ia-1) + i
    nbc = ipt(12) + ipt(1) * isum + i
    isum = isum + 1
    ic = icp(iptbc+isum)
    if(lctrl(10)) then
      umaxt = umax(ic)
    else
      itmp = (ic-1)*ipt(89)+4*ipt(88)+idepth(i)
      umaxt = (umax(ic) + dtemp(itmp))
    end if
    xmfold = xmf(nbc)
    dxmf = umaxt*xmf(nbcx)*xmonod(isum)*term / cmw(ic)
C
C— Limit bioreaction during initial startup period.
C
  if ((t(9)).lt.t(11)) dxmf = (t(9)/t(11))*dxmf
C
C— Update the bio-reaction term.
C
  if(xmf(nbc).ne.zer0) then
    rxn(nac) = -dxmf/xmf(nbc)
    rxn(nbc) = -dxmf/xmf(nbc)
  else
    rxn(nac) = zer0
    rxn(nbc) = zer0
  end if
  rxnp(i) = rxnp(i) - dxmf
  rxnp(i+ipt1) = rxnp(i+ipt1) - dxmf
end if
125 continue
100 continue
C
C— Check the size of the bioreaction term.
C
isumg = 0

```

```

isuma = 0
isumb = 0
nbcx = ipt(12) + ipt1 * (ipt(7)-1)
do 150 ic = 1, ipt(65)
  if(ic.eq.icp(isumg+1)) isumg = isumg + 1
  if(ic.eq.icp(ig+isuma+1)) isuma = isuma + 1
  if(ic.eq.icp(iptbc+isumb+1)) isumb = isumb + 1
  if(ic.eq.icp(isumg).and.ic.eq.icp(ig+isuma)
  + .and.ic.eq.icp(iptbc+isumb)) then
    ngc = ipt(8) + ipt1 * (isumg-1)
    nac = ipt(9) + ipt1 * (isuma-1)
    nbc = ipt(12) + ipt1 * (isumb-1)
    do 155 iel = 1, ipt(0)
      iel3 = iel*3
      iel1 = iel3-2
      iel2 = iel3-1
      i1 = nodel(iel1)
      i1s = nodept(i1)+nelpt(iel1)
      i2 = nodel(iel2)
      i2s = nodept(i2)+nelpt(iel2)
      i3 = nodel(iel3)
      i3s = nodept(i3)+nelpt(iel2)
      if(lctrl(10)) then
        cvpt = cvp(ic)
        casolt = casol(ic)
      else
        itemp = (ic-1)*ipt(89)
        itemp2 = itemp + ipt(88)*2
        itmp1 = itemp+idepth(i1)
        itmp2 = itemp+idepth(i2)
        itmp3 = itemp+idepth(i3)
        itmp21 = itemp2+idepth(i1)
        itmp22 = itemp2+idepth(i2)
        itmp23 = itemp2+idepth(i3)
        cvpt = cvp(ic) + third*(dtemp(itmp1)
        + dtemp(itmp2) + dtemp(itmp3) )
        casolt = casol(ic) + third*(dtemp(itmp21)
        + dtemp(itmp22) + dtemp(itmp23) )
      end if
      ngc1 = ngc + i1
      nac1 = nac + i1
      nbc1 = nbc + i1
      ngc2 = ngc + i2
      nac2 = nac + i2
      nbc2 = nbc + i2
      ngc3 = ngc + i3
      nac3 = nac + i3
      nbc3 = nbc + i3
      den1 = den(i1+ipt1)
      den2 = den(i2+ipt1)
      den3 = den(i3+ipt1)
      keq = patm * casolt / cvpt
      if(lctrl(16)) then
        if(ic.le.ipt(16)) then
          is = (ipt(3)+ipt(4)+ipt(5)+ic-1)*ipt1
          ism = (matel(iel)-1)*ipt(15)+ic
          if(xmf(is+i1).le.zer0) then
            efrac1 = zer0
          else
            efrac1 = ((xmf(is+i1) / bok(ism)) ** bom(ism) )
            / ( cmw(ic) * den(ipt1+i1) * 1.0d3 )
          end if
          if(xmf(is+i2).le.zer0) then
            efrac2 = zer0
          else
            efrac2 = ((xmf(is+i2) / bok(ism)) ** bom(ism) )
            / ( cmw(ic) * den(ipt1+i2) * 1.0d3 )
          end if
          if(xmf(is+i3).le.zer0) then
            efrac3 = zer0
          else
            efrac3 = ((xmf(is+i3) / bok(ism)) ** bom(ism) )
            / ( cmw(ic) * den(ipt1+i3) * 1.0d3 )
          end if
          extots = t(8) * por(iel) * kex(5*(ic-1)+5) *
          + ( den1 * (efrac1-xmf(nac1)) +
          + den2 * (efrac2-xmf(nac2)) +
          + den3 * (efrac3-xmf(nac3)) )

```

```

else
  extots = zero
end if
xtot = por(iel) * (
+   den1 * sat(i1s+ipt2) * xmf(nac1) +
+   den2 * sat(i2s+ipt2) * xmf(nac2) +
+   den3 * sat(i3s+ipt2) * xmf(nac3) )
+   extot = t(8) * por(iel) * kex(5*(ic-1)+4) *
+   ( den1 * (keq*xmf(ngc1)-xmf(nac1)) +
+   den2 * (keq*xmf(ngc2)-xmf(nac2)) +
+   den3 * (keq*xmf(ngc3)-xmf(nac3)) ) + extots
rxtot = t(8) * (rxn(nac1)*xmf(nac1) +
+   rxn(nac2)*xmf(nac2) + rxn(nac3)*xmf(nac3))
else
  sb1 = xbmax / pmw(ipt1+i1)
  sb2 = xbmax / pmw(ipt1+i2)
  sb3 = xbmax / pmw(ipt1+i3)
  xtot = xbmax * (
+   xmf(nbc1) / pmw(ipt1+i1) +
+   xmf(nbc2) / pmw(ipt1+i2) +
+   xmf(nbc3) / pmw(ipt1+i3) )
+   extot = t(8) * por(iel) * kex(5*(ic-1)+4) *
+   ( den1 * (xmf(nac1)-xmf(nbc1)) +
+   den2 * (xmf(nac2)-xmf(nbc2)) +
+   den3 * (xmf(nac3)-xmf(nbc3)) )
+   rxtot = t(8) * (rxn(nbc1)*xmf(nbc1) +
+   rxn(nbc2)*xmf(nbc2) + rxn(nbc3)*xmf(nbc3))
end if
if(-rxtot.gt.xtot+extot) then
  rxnp(i1) = rxnp(i1) - rxn(nac1) * xmf(nbc1)
  rxnp(i2) = rxnp(i2) - rxn(nac2) * xmf(nbc2)
  rxnp(i3) = rxnp(i3) - rxn(nac3) * xmf(nbc3)
  rxnp(i1+ipt1) = rxnp(i1+ipt1)
+   - rxn(nbc1) * xmf(nbc1)
  rxnp(i2+ipt1) = rxnp(i2+ipt1)
+   - rxn(nbc2) * xmf(nbc2)
  rxnp(i3+ipt1) = rxnp(i3+ipt1)
+   - rxn(nbc3) * xmf(nbc3)
  rxn(nac1) = zero
  rxn(nac2) = zero
  rxn(nac3) = zero
  if(isumb.gt.ipt(17)) then
    do 156 ii = 1, ipt(4)
      nica = (ii-1)*ipt1
      rxn(nica+i1) = zero
      rxn(nica+i2) = zero
      rxn(nica+i3) = zero
156    continue
      do 157 ii = 1, ipt(7)
        nicb = (ii-1)*ipt1
        rxn(nicb+i1) = zero
        rxn(nicb+i2) = zero
        rxn(nicb+i3) = zero
157    continue
      end if
    end if
155  continue
  end if
150  continue
C
C— Return when no mass transfer rate limitation is considered for
C— aqueous/biophase interactions.
C
  if(lctrl(16)) then
    RETURN
  end if
C
C— Now sequentially solve for the biophase mole fractions.
C
  do 200 icbio = 1, ipt(7)-1
    iptr = iptbc + icbio
    icpt = (iptc-1)*ipt1
    icpts = (iptc-1)*ipt2
    ic = icp(icpt)
C
C— Zero the finite element matrices.
C
    do 210 i = 1, ipt1

```

```

      rhs(i) = zero
      nrow = (i-1)*nbw(1)
      do 210 j = 1, nbw(1)
210        a(nrow+j) = zero
C
C— Compute the local finite element matrices.
C
      do 220 i = 1, ipt(0)
C
C— Set the pointers to the local nodes 11, 12, and 13. The
C— postscript s is for stacked local nodes, p is for phase, and
C— c is for component. Compute constants.
C
      iel3 = i*3
      iel1 = iel3-2
      iel2 = iel3-1
      i1 = nodel(iel1)
      i1s = nodept(i1)+nelpt(iel1)
      i1c = i1
      i1cs = icpts + i1s
      i2 = nodel(iel2)
      i2s = nodept(i2)+nelpt(iel2)
      i2c = i2
      i2cs = icpts + i2s
      i3 = nodel(iel3)
      i3s = nodept(i3)+nelpt(iel3)
      i3c = i3
      i3cs = icpts + i3s
C
C— Compute the biomass "saturation". This is based on the
C— maximum allowable biomass and uses water properties.
C
      sb = xbmax / por(i)
      sb1 = sb / den(ipt(41)+i1)
      sb2 = sb / den(ipt(41)+i2)
      sb3 = sb / den(ipt(41)+i3)
      term = aby30(i)/t(8)
C
C— Compute the mass matrix in lumped form.
C
      a11 = term*( sb1*3.0d0 + sb2 + sb3 )
      a12 = term*( sb1 + sb2 + sb3/2.0d0 )
      a13 = term*( sb1 + sb2/2.0d0 + sb3 )
      a21 = a12
      a22 = term*( sb1 + sb2*3.0d0 + sb3 )
      a23 = term*( sb1/2.0d0 + sb2 + sb3 )
      a31 = a13
      a32 = a23
      a33 = term*( sb1 + sb2 + sb3*3.0d0 )
      if(lctrl(8)) then
        a11 = a11 + a12 + a13
        a12 = zero
        a13 = zero
        a22 = a21 + a22 + a23
        a21 = zero
        a23 = zero
        a33 = a31 + a32 + a33
        a31 = zero
        a32 = zero
      end if
C
C— Now compute the exchange matrix. First consider the terms
C— multiplied by the biophase mole fraction. These terms are the
C— phase mole exchange and the phase mole reaction.
C
      pex1 = -rxn(i1c)
      pex2 = -rxn(i2c)
      pex3 = -rxn(i3c)
      b11 = aby30(i) * (pex1*3.0d0 + pex2 + pex3 )
      b12 = aby30(i) * (pex1 + pex2 + pex3/2.0d0)
      b13 = aby30(i) * (pex1 + pex2/2.0d0 + pex3 )
      b21 = b12
      b22 = aby30(i) * (pex1 + pex2*3.0d0 + pex3 )
      b23 = aby30(i) * (pex1/2.0d0 + pex2 + pex3 )
      b31 = b13

```

```

b32 = b23
b33 = aby30(i) * (pex1 + pex2 + pex3*3.0d0)
C
C— Now compute the right hand side terms. These terms are the
C— component mole exchange and the component mole reaction.
C
rhs1 = por(i)*rhsex(i1cs)
rhs2 = por(i)*rhsex(i2cs)
rhs3 = por(i)*rhsex(i3cs)
f1 = aby12(i) * (2.0d0*rhs1 + rhs2 + rhs3)
f2 = aby12(i) * ( rhs1 + 2.0d0*rhs2 + rhs3)
f3 = aby12(i) * ( rhs1 + rhs2 + 2.0d0*rhs3)
C
C— Assemble global matrix and right hand side vector in banded form.
C
irow1 = (i1-1)*nbw(1)
irow2 = (i2-1)*nbw(1)
irow3 = (i3-1)*nbw(1)
icol11 = 1 + nbw(0)
icol12 = icol11 + (i2 - i1)
icol13 = icol11 + (i3 - i1)
icol22 = icol11
icol21 = icol22 + (i1 - i2)
icol23 = icol22 + (i3 - i2)
icol33 = icol11
icol31 = icol33 + (i1 - i3)
icol32 = icol33 + (i2 - i3)
ab11 = a11 + t(10)*b11
ab12 = a12 + t(10)*b12
ab13 = a13 + t(10)*b13
ab21 = a21 + t(10)*b21
ab22 = a22 + t(10)*b22
ab23 = a23 + t(10)*b23
ab31 = a31 + t(10)*b31
ab32 = a32 + t(10)*b32
ab33 = a33 + t(10)*b33
a(irow1 + icol11) = a(irow1 + icol11) + ab11
a(irow1 + icol12) = a(irow1 + icol12) + ab12
a(irow1 + icol13) = a(irow1 + icol13) + ab13
a(irow2 + icol21) = a(irow2 + icol21) + ab21
a(irow2 + icol22) = a(irow2 + icol22) + ab22
a(irow2 + icol23) = a(irow2 + icol23) + ab23
a(irow3 + icol31) = a(irow3 + icol31) + ab31
a(irow3 + icol32) = a(irow3 + icol32) + ab32
a(irow3 + icol33) = a(irow3 + icol33) + ab33
rhs(i1) = rhs(i1) + f1 - b11 * xmf(i1c)
+ - b12 * xmf(i2c) - b13 * xmf(i3c)
rhs(i2) = rhs(i2) + f2 - b21 * xmf(i1c)
+ - b22 * xmf(i2c) - b23 * xmf(i3c)
rhs(i3) = rhs(i3) + f3 - b31 * xmf(i1c)

```

```

+ - b32 * xmf(i2c) - b33 * xmf(i3c)
220 continue
C
C— Collapse full matrix into sparse form used by Harwell. Also
C— scale array by dividing rows through by the diagonal value.
C
ia = 0
do 230 irow = 1, ipt(1)
nrow = (irow-1)*nbw(1)
aii = rone / a(nrow+1+nbw(0))
rhs(irow) = rhs(irow) * aii
do 230 icol = 1, nbw(1)
if (a(nrow+icol) .ne. zer0) then
ia = ia + 1
a(ia) = a(nrow+icol) * aii
irn(ia) = irow
icn(ia) = icol+irow-nbw(0)-1
endif
230 continue
C
C— Solve the linear system using Harwell routines.
C
call ma28ad(ipt(1), ia, a, icn1, irn, icn, u, ikeep, iw, wiflag)
if (iflag .lt. 0) then
write (ipt(28),*) 'iflag return from harwell is ', iflag
write (ipt(28),*) 'bio component is: ', cname(ic)
endif
call ma28cd (ipt(1), a, icn1, icn, ikeep, rhs, w, mtype)
C
C— Update the solution and determine the max norm of the updated
C— solution.
C
dxmf = zer0
xmfmax = xround
do 240 i = 1+iptc, ipt1+iptc
xmfold=xmf(i)
xmf(i) = xmf(i) + rhs(i-iptc)
xmfmax = dmax1(xmfmax, dabs(xmf(i)))
240 dxmf = dmax1(dxmf, dabs(xmf(i)-xmfold))
if(dxmf/xmfmax.gt.t(15)) ibconv = ibconv + 1
200 continue
C
C— Biological equation set is converged if each component equation
C— has converged (i.e. ibconv = 0).
if(ibconv.gt.0) iconv = iconv + 1
return
end

```

Subroutine - cbal.f

```

C
C
C CBAL.f - Subroutine which computes the mole balance check for the
C aqueous and gas phase components and phases.
C
C Arguments: its - integer time step
C lbalp - compute the phase mass balance when true
C lbalc - compute the component mass balance when true
C lpcbal - print the mass balance information when true
C lfirst - indicates the first call during a time step
C
C Required Control Flags:
C
C lctrl(1) - logical variable controlling presence of flow
C solution
C lctrl(1) = .true. - compute flow solution
C lctrl(1) = .false. - skip flow solution
C lctrl(2) - logical variable controlling presence of transport
C solution

```

```

C
C lctrl(2) = .true. - compute transport solution
C lctrl(2) = .false. - skip transport solution
C lctrl(12) - logical variable denoting presence of a well
C lctrl(12) = .true. - well present
C lctrl(12) = .false. - well not present
C lctrl(16) - logical variable denoting method of including
C biological reaction
C lctrl(16) = .true. - include bioreaction in
C aqueous transport
C lctrl(16) = .false. - solve FEM solution for
C rate limited biophase
C
C WARNING: THIS ROUTINE IS DIMENSIONED FOR 9 COMPO-
C NENTS WHEN USING
C REPORT STYLE OUTPUT
C
subroutine CBAL (its, lbalp, lbalc, lpcbal, lfirst)
include 'dimen.inc'
character*10 cname(ncmp)

```

```

common /cb1/ matel(nelmx),nodel(nel3),nodept(nnmx),nelpt(nel3),
+ matpt(nn6)
common /cb1c/ xnode(nnmx),znode(nnmx),rbar(nelmx),area(nelmx)
common /cb2/ p(nn3)
common /cb3/ sat(nnstk3)
common /cb6b/ por(nelmx),srw(nnstk)
common /cb6c/ temp(nnmx)
common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
+ chen(ncmp),casol(ncmp),cmdif(ncmp2)
common /cb8/ vis(nnmx),pmw(nn3)
common /cb9/ xmf(nmf)
common /cb10/ den(nn6)
common /cb11/ pex(nns10),rxnp(nn2)
common /cb30/ ibc(nnmx)
common /cb32/ bcf(nn2)
common /cb31/ source(nn2)
common /cb42/ amb(icnt),fmb(ksolve)
common /cb62/ rxn(nmf),cex(nmfs)
common /cb64/ bok(ncmp),bom(ncmp),krt(ncmp)
common /cb64b/ bsden(nmbk)
common /cb84/ ibcxmf(nmbc),bcxmf(nmbc),dfxmf(nmbc)
common /cb85/ flux(ncmpp5),sflux(ncmpp5),first(ncmp)
common /cb86/ str1(ncmpp5),str0(ncmpp5),cmf(ncmpp5),csink(ncmpp5)
+ ,cwsink(ncmpp5),csflux(ncmpp5),cmass1(ncmpp5),cmass0(ncmpp5)
+ ,cphex(ncmpp5),crsink(ncmpp5),tmass1,tmass0
common /cb91/ cname
dimension str(ncmpp5),resid(ncmpp5),sink(ncmpp5),wsink(ncmpp5)
+ ,cmass(ncmpp5),phex(ncmpp5),perr1(ncmpp5),perr2(ncmpp5)
+ ,rsink(ncmpp5)

```

C
C— Describe variables: str1(1-5) are the total mass storage of each
C— phase at t+1, str1(6..) are the total mass storage of each
C— component at t+1; str(...) are the total mass storages at t;
C— str0(...) are the initial mass storages; resid(...) are the
C— changes in phase and component mass not accounted for by sinks,
C— sources, phase exchange, etc.; cmf(...) are the cumulative total
C— mass fluxes for the phases and components; sink(...) are the
C— total mass sinks for the phases and components; csink(...) are
C— the cumulative total mass sinks for the the phases and components;
C— wsink(...) are the total mass sinks at the well for the phases
C— and components; cwsink(...) are the cumulative total mass sinks
C— at the well for the phases and components; sflux(...) are the
C— total mass fluxes at the surface for the phases and components;
C— csflux(...) are the cumulative total mass fluxes at the surface
C— for the phases and components; cmass1(...) are the mass storages
C— of each component in each phase at t+1; cmass0(...) are the mass
C— storages of each component in each phase at t; cmass0(...) are
C— the initial mass storages of each component in each phase;
C— rsink(...) are the total mass reaction sinks for the phases
C— and components; crsink(...) are the cumulative total mass reaction
C— sinks for the phases and components

```

C
istop = ipt(65)
ipt1x3 = ipt(41)
ipt2x5 = ipt(52)

```

C
C— Initialize mass storage vectors.

```

C
if((lfirst) then
if(its.ne.0) tmass = tmass1
tmass1 = zero
iphex = zero
bmass = zero

```

C
C— Save previous mass storage.

```

C
do 10 i = 1, istop+5
if(lctrl(1)) then
flux(i) = zero
sflux(i) = zero
end if
sink(i) = zero
rsink(i) = zero
wsink(i) = zero
resid(i) = zero
phex(i) = zero
if(its.ne.0) str(i) = str1(i)
10 str1(i) = zero

```

```

do 11 i = 1, istop*5
if(its.ne.0) cmass(i) = cmass1(i)
11 cmass1(i) = zero
end if
C
C— First compute the mass storage of the phases.
C
do 100 jel = 1, ipt(0)
n3 = (jel-1)*3
i1 = nodel(n3+1) ! element node numbers
i2 = nodel(n3+2)
i3 = nodel(n3+3)
ig1 = i1 ! nodel gas phase storage locations
ig2 = i2
ig3 = i3
ia1 = ipt(1)+ig1 ! nodel aqueous phase storage locations
ia2 = ipt(1)+ig2
ia3 = ipt(1)+ig3
in1 = ipt(1)+ia1 ! nodel napl phase storage locations
in2 = ipt(1)+ia2
in3 = ipt(1)+ia3
ndstk1 = nodept(i1) + nelpt(n3+1) ! node position in stack
ndstk2 = nodept(i2) + nelpt(n3+2)
ndstk3 = nodept(i3) + nelpt(n3+3)
ig1s = ndstk1 ! gas phase stacked node numbers
ig2s = ndstk2
ig3s = ndstk3
ia1s = ipt(2)+ig1s ! aqueous phase stacked node numbers
ia2s = ipt(2)+ig2s
ia3s = ipt(2)+ig3s
in1s = ipt(2)+ia1s ! napl phase stacked node numbers
in2s = ipt(2)+ia2s
in3s = ipt(2)+ia3s
is1s = ipt(2)+in1s ! solid phase stacked node numbers
is2s = ipt(2)+in2s
is3s = ipt(2)+in3s
ib1s = ipt(2)+is1s ! bio phase stacked node numbers
ib2s = ipt(2)+is2s
ib3s = ipt(2)+is3s
xbio = rbar(jel) * area(jel) * third
x = xbio * por(jel)

```

C
C— First the gas phase mass storage and exchange terms.

```

C
if(lbalp) then
str1(1) = str1(1) + x * (
+ den(ipt1x3+ig1) * sat(ig1s) +
+ den(ipt1x3+ig2) * sat(ig2s) +
+ den(ipt1x3+ig3) * sat(ig3s) )
phex(1) = phex(1) + x * t(8) * ( pex(ipt2x5+ig1s)
+ pex(ipt2x5+ig2s) + pex(ipt2x5+ig3s) )

```

C
C— Now the aqueous phase mass storage and exchange terms.

```

C
str1(2) = str1(2) + x * (
+ den(ipt1x3+ia1) * sat(ia1s) +
+ den(ipt1x3+ia2) * sat(ia2s) +
+ den(ipt1x3+ia3) * sat(ia3s) )
phex(2) = phex(2) + x * t(8) * ( pex(ipt2x5+ia1s)
+ pex(ipt2x5+ia2s) + pex(ipt2x5+ia3s) )

```

C
C— Now the gas phase component mass storages.

```

C
end if
if(lbalc) then
do 15 i = 1, ipt(3)
ic = icp(i)
igasc = (i-1)*ipt(1)
cel = x * cmw(ic) * (
+ den(ig1) * sat(ig1s) * xmf(igasc+i1) +
+ den(ig2) * sat(ig2s) * xmf(igasc+i2) +
+ den(ig3) * sat(ig3s) * xmf(igasc+i3) )
cmass1(ic) = cmass1(ic) + cel
str1(5+ic) = str1(5+ic) + cel
15 continue

```

C
C— Now the aqueous phase component mass storages.

```

do 20 i = 1, ipt(4)
  ipt3 = i + ipt(3)
  ic = icp(ipt3)
  iaqc = (ipt3-1)*ipt(1)
  cel = krt(d(ic) * x * cmw(ic) * (
+   den(ia1) * sat(ia1s) * xmf(iaqc+i1) +
+   den(ia2) * sat(ia2s) * xmf(iaqc+i2) +
+   den(ia3) * sat(ia3s) * xmf(iaqc+i3) )
  cmass1(istop+ic) = cmass1(istop+ic) + cel
  str1(5+ic) = str1(5+ic) + cel
20 continue
C
C— Now the napl phase mass storage and exchange terms.
C
if(ipt(5).gt.0) then
  str1(3) = str1(3) + x * (
+   den(ipt1x3+in1) * sat(in1s) +
+   den(ipt1x3+in2) * sat(in2s) +
+   den(ipt1x3+in3) * sat(in3s) )
  phex(3) = phex(3) + x * t(8) * ( pex(ipt2x5+in1s)
+   + pex(ipt2x5+in2s) + pex(ipt2x5+in3s) )
C
C— Now the napl phase component mass storages. This form assumes
C— that all the organic components are present in the napl phase.
C
do 25 i = 1, ipt(5)
  ipipt5 = i + ipt(58)
  ic = icp(ipt5)
  inc = (ipipt5-1)*ipt(1)
  cel = x * cmw(ic) * (
+   den(in1) * sat(in1s) * xmf(inc+i1) +
+   den(in2) * sat(in2s) * xmf(inc+i2) +
+   den(in3) * sat(in3s) * xmf(inc+i3) )
  cmass1(2*istop+ic) = cmass1(2*istop+ic) + cel
  str1(5+ic) = str1(5+ic) + cel
25 continue
end if
C
C— Now the solid phase mass balance and exchange terms.
C
if(ipt(6).gt.0) then
  phex(4) = phex(4) + xbio * t(8) * ( pex(ipt2x5+is1s)
+   + pex(ipt2x5+is2s) + pex(ipt2x5+is3s) )
C
C— Now the solid phase component mass balances.
C
do 30 i = 1, ipt(6)
  ipipt5 = i + ipt(59)
  ic = icp(ipt5)
  isc = (ipipt5-1)*ipt(1)
  if(xbok.gt.zer0) then
    if(i.eq.1) then
      solden = (rone-xden)*bsden(matel(i))
    else if(i.eq.2) then
      isum = ipt(3)+ipt(4)+ipt(5)
      ic = icp(isum+i-1)
      solden = xden*bsden(matel(i))
    end if
    cel = solden *
+   ( xmf(isc+i1) + xmf(isc+i2) + xmf(isc+i3) )
+   * xbio
  else
    cel = bsden(matel(jel)) *
+   ( xmf(isc+i1) + xmf(isc+i2) + xmf(isc+i3) )
+   * xbio
  end if
  cmass1(3*istop+ic) = cmass1(3*istop+ic) + cel
  str1(5+ic) = str1(5+ic) + cel
  str1(4) = str1(4) + cel
30 continue
end if
C
C— Now the biophase mass balance and reaction terms.
C— This assumes a constant biophase volume.
C
if(ipt(7).gt.0) then
  do 35 i = 1, ipt(7)-1
    ipipt6 = i + ipt(60)

```

```

    ic = icp(ipt6)
    ibioc = (ipipt6-1)*ipt(1)
  C
  C— First consider a separate biophase.
  C
  if (.not.lctrl(16)) then
    str1(5) = str1(5) + xbio * xamax * 3.0d0
    phex(5) = phex(5) + xbio * t(8) * ( pex(ipt2x5+ib1s)
+   + pex(ipt2x5+ib2s) + pex(ipt2x5+ib3s) )
    cel = xbio * cmw(ic) * (
+   + xamax/pmw(ia1) * xmf(ibioc+i1) +
+   + xamax/pmw(ia2) * xmf(ibioc+i2) +
+   + xamax/pmw(ia3) * xmf(ibioc+i3) )
  C
  C— Now the bio reaction mass sinks.
  C
  sel = xbio * cmw(ic) * ( rxn(ibioc+i1)
+   * xmf(ibioc+i1) + rxn(ibioc+i2)
+   * xmf(ibioc+i2) + rxn(ibioc+i3)
+   * xmf(ibioc+i3) ) * t(8)
  C
  C— Now sum the bio reaction mass sinks into the mass balance.
  C
  cmass1(4*istop+ic) = cmass1(4*istop+ic) + cel
  rsink(5) = rsink(5) + sel
  rsink(5+ic) = rsink(5+ic) + sel
  str1(5) = str1(5) + cel
  str1(5+ic) = str1(5+ic) + cel
  C
  C— Otherwise sum the bioreaction sinks when a separate biophase
  C— is not considered. Use biophase reaction terms since they are
  C— the same as the aqueous phase reaction terms.
  C
  else if (lctrl(16)) then
    sel = xbio * cmw(ic) * ( rxn(ibioc+i1)
+   * xmf(ibioc+i1) + rxn(ibioc+i2)
+   * xmf(ibioc+i2) + rxn(ibioc+i3)
+   * xmf(ibioc+i3) ) * t(8)
  C
  C— Now sum the bio reaction mass sinks into the mass balance. When
  C— a separate biophase is not considered the reaction terms are
  C— used for the aqueous phase mass balance.
  C
  rsink(2) = rsink(2) + sel
  rsink(5+ic) = rsink(5+ic) + sel
  end if
35 continue
C
C— Now sum the biomass. Do not include the biomass in the phase mass.
C
  ibioc = ipt(12) + (ipt(7)-1) * ipt(1)
  cel = xbio * ( xmf(ibioc+i1) + xmf(ibioc+i2) +
+   xmf(ibioc+i3) )
  bmass = bmass + cel
  end if
  end if
100 continue
C
C— Now initialize variables and return if this is the first call.
C
  if (its .eq. 0) then
    do 200 i = 1, 5+istop
      cphex(i) = zer0
      csink(i) = zer0
      cwsink(i) = zer0
      crsink(i) = zer0
      csflux(i) = zer0
      cmf(i) = zer0
200   str0(i) = str1(i)
    do 210 i = 1, 5*istop
210   cmass0(i) = cmass1(i)
      tmass0 = zer0
      tcsink = zer0
      tcwsink = zer0
      tcrsink = zer0
      tclflux = zer0
      tcephex = zer0

```

```

do 220 i = 1,5
220  tmass0 = tmass0 + str0(i)
    tmass1 = tmass0
    if(iprt(27)) then
        write(25,501) its,t(9)
        write(25,502) 'Phase mass (kg) '
    +   (str0(i),i=1,5),tmass0
    do 230 ii = 1, istop
        i = ii + 5
230  write(25,503) cname(ii)
    +   (cmass0(iii*istop+ii),iii=0,4),str0(i)
    if(lctr(3)) write(25,503) cname(ii)
    +   ,zer0,zer0,zer0,zer0,bmass,zer0
    else
        write(25,504)
        write(25,500) zer0,(str0(i),i=1,5),tmass0
    +   ,zer0,zer0,zer0,zer0,zer0
    do 240 i = 1, ipt(15)
        write(28+i,505) cname(i)
240  write(28+i,500) zer0,(cmass0(ii*istop+i),ii=0,4)
    +   ,str0(i+5),zer0,zer0,zer0,zer0,zer0
    end if
    do 250 i = 1, ipt(65)
250  first(i) = zer0
    return
end if
C
C— Now compute the boundary fluxes for the flow solution.
C
if (lctr(1)) then
c
c— Compute gas phase boundary fluxes; i.e. all specified first type
c— nodes.
    if (ipt(18) .gt. 0) then
        do 300 jj = 1, ipt(18)
            irow = 2 * ibc(jj)
            nbc = irow/2
            nbed = ipt(41) + nbc
            deng = den(nbed)
C
C— Compute gas phase component boundary flux at first type pressure
C— nodes. Note: the surface flux is included in the boundary flux.
C
            pmwbc = zer0
            do 310 i = 1, ipt(3)
                ic = icp(i)
                icxmf = (i-1)*ipt(1) + nbc
                if(ibcxmf(nbc).eq.3) then
                    fkeep = den(nbc) * bcf(nbc) * bcxmf(icxmf)
                +   * cmw(ic) * t(8)
                if(lbale) flux(ic+5) = flux(ic+5) + fkeep
                pmwbc = pmwbc + bcxmf(icxmf) * cmw(ic)
                else if(ibcxmf(nbc).eq.2) then
                    fkeep = den(nbc) * bcf(nbc) * xmf(icxmf)
                +   * cmw(ic) * t(8)
                if(lbale) flux(ic+5) = flux(ic+5) + fkeep
                else if(ibcxmf(nbc).eq.1) then
                    fkeep = first(icxmf)
                    if(lbale) flux(ic+5) = flux(ic+5) + fkeep
                end if
                if(znode(nbc).eq.zer0.and.lbale)
                +   sflux(ic+5) = sflux(ic+5) + fkeep
310  continue
            if(lctr(2).and.ibcxmf(nbc).eq.3) deng = den(nbc) * pmwbc
300  if(lbalp) flux(1) = flux(1) + fmb(irow) * deng * t(8)
        endif
C
C— Compute aqueous phase boundary fluxes; i.e. all specified first
C— type nodes.
C
        if (ipt(19) .gt. 0) then
            do 320 jj = 1, ipt(19)
                ipt18 = ipt(18)
                nbc = ibc(ipt18+jj) + ipt(1)
                irow = 2*ibc(ipt18+jj) - 1
C
C— Compute aqueous phase component boundary flux at first type
C— pressure nodes. Note: the surface flux is included in the

```

```

C— boundary flux.
C
        if(lbale) then
            do 330 i = 1, ipt(4)
                ipt3 = i + ipt(3)
                ic = icp(ipt3)
                icxmf = ipt(9) + (i-1)*ipt(1) + ibc(ipt18+jj)
                if(ibcxmf(nbc).eq.3) then
                    fkeep = den(nbc) * bcf(nbc) * bcxmf(icxmf)
                +   * cmw(ic) * t(8)
                    flux(ic+5) = flux(ic+5) + fkeep
                else if(ibcxmf(nbc).eq.2) then
                    fkeep = den(nbc) * bcf(nbc) * xmf(icxmf)
                +   * cmw(ic) * t(8)
                    if(lbale) flux(ic+5) = flux(ic+5) + fkeep
                else if(ibcxmf(nbc).eq.1) then
                    fkeep = first(icxmf)
                    if(lbale) flux(ic+5) = flux(ic+5) + fkeep
                end if
                if(znode(nbc-ipt(1)).eq.zer0)
                +   sflux(ic+5) = sflux(ic+5) + fkeep
330  continue
            end if
            if(lbalp) then
                nbed = nbc + ipt(41)
                flux(2) = flux(2) + fmb(irow) * den(nbed)
                +   * t(8)
            end if
320  continue
        end if
C
C— Compute gas phase sources and sinks at nodes with constant gas
C— flux.
C
        if (ipt(22) .gt. 0) then
            ii = ipt(62)
            do 340 jj = 1, ipt(22)
                nbc = ibc(ii+jj)
                nbed = ipt(41) + nbc
                deng = den(nbed)
C
C— Compute gas phase component boundary flux at constant flux
C— nodes. Note: the surface flux is included in the boundary flux.
C
                pmwbc = zer0
                do 345 i = 1, ipt(3)
                    ic = icp(i)
                    icxmf = (i-1)*ipt(1) + nbc
                    if(ibcxmf(nbc).eq.3) then
                        fkeep = den(nbc) * bcf(nbc) * bcxmf(icxmf)
                    +   * cmw(ic) * t(8)
                    if(lbale) flux(ic+5) = flux(ic+5) + fkeep
                    pmwbc = pmwbc + bcxmf(icxmf) * cmw(ic)
                    else if(ibcxmf(nbc).eq.2) then
                        fkeep = den(nbc) * bcf(nbc) * xmf(icxmf)
                    +   * cmw(ic) * t(8)
                    if(lbale) flux(ic+5) = flux(ic+5) + fkeep
                    else if(ibcxmf(nbc).eq.1) then
                        fkeep = first(icxmf)
                        if(lbale) flux(ic+5) = flux(ic+5) + fkeep
                    end if
                    if(znode(nbc).eq.0.d0.and.lbale)
                    +   sflux(ic+5) = sflux(ic+5) + fkeep
345  continue
                if(ibcxmf(nbc).eq.3) deng = den(nbc) * pmwbc
340  if(lbalp) sink(1) = sink(1) + deng * t(8) *
                +   source(jj) * patm * temp(nbc)
                +   / ( ( patm + p(nbc) ) * tstd )
            endif
C
C— Compute aqueous phase sources and sinks at nodes with constant
C— aqueous flux.
C
            if (ipt(23) .gt. 0) then
                ii = ii + ipt(22)
                do 350 jj = 1, ipt(23)
                    nbc = ipt(1) + ibc(ii+jj)

```

C— Compute aqueous phase component boundary flux at constant flux nodes. Note: the surface flux is included in the boundary flux.

```

C
  if(lbalc) then
    do 355 i = 1, ipt(4)
      ipt3 = i + ipt(3)
      ic = icp(ipt3)
      icxmf = ipt(9) + (i-1)*ipt(1) + ibc(jj+ii)
      if(ibcxmf(nbc).eq.3) then
        fkeep = den(nbc) * bcf(nbc) * bcxmf(icxmf)
        * cmw(ic) * t(8)
      +
        flux(ic+5) = flux(ic+5) + fkeep
      else if(ibcxmf(nbc).eq.2) then
        fkeep = den(nbc) * bcf(nbc) * xmf(icxmf)
        * cmw(ic) * t(8)
      +
        flux(ic+5) = flux(ic+5) + fkeep
      else if(ibcxmf(nbc).eq.1) then
        fkeep = first(icxmf)
        if(lbalc) flux(ic+5) = flux(ic+5) + fkeep
      end if
      if(znode(nbc-ipt(1)).eq.zer0)
      +
        sflux(ic+5) = sflux(ic+5) + fkeep
    355 continue
  end if
  if(lbalp) then
    nbcd = ipt(41)+nbc
    ipt22 = ipt(22)
    sink(2) = sink(2) + den(nbcd)
    +
      * source(ipt22+jj) * t(8)
  end if
  350 continue
endif

```

C— Now compute the gas phase flux at the well.

```

C
  if (lctrl(12)) then
    ii = ipt(64)
    jstr = ipt(22) + ipt(23)
    do 360 jj = 1, ipt(24)
      nbc = ibc(ii+jj)
      nbcd = ipt(41) + nbc
      deng = den(nbcd)

```

C— Now compute the gas phase component flux at the well.

```

C
  pmwbc = zer0
  do 365 i = 1, ipt(3)
    ic = icp(i)
    icxmf = (i-1)*ipt(1)+ibc(ii+jj)
    if(ibcxmf(nbc).eq.3) then
      xmfbc = bcxmf(icxmf)
      pmwbc = pmwbc + bcxmf(icxmf) * cmw(ic)
      fkeep = den(nbc) * bcf(nbc) * xmfbc
      +
        * cmw(ic) * t(8)
    else if(ibcxmf(nbc).eq.2) then
      fkeep = den(nbc) * bcf(nbc) * xmf(icxmf)
      +
        * cmw(ic) * t(8)
    +
      flux(ic+5) = flux(ic+5) + fkeep
    else if(ibcxmf(nbc).eq.1) then
      fkeep = first(icxmf)
      if(lbalc) flux(ic+5) = flux(ic+5) + fkeep
    else
      xmfbc = xmf(icxmf)
    end if
    if(lbalc) wsink(5+ic) = wsink(5+ic) + fkeep
  365 continue
  if(ibcxmf(nbc).eq.3) deng = den(nbc) * pmwbc
  if(lbalp) wsink(1) = wsink(1) + deng * t(8) *
  +
    source(jstr+jj) * patm * temp(nbc)
  +
    / (( patm + p(nbc) ) * tstd )

```

C— Now compute the aqueous phase flux at the well when extracting.

```

C
  if (qwell .lt. zer0) then
    nbcw = ipt(1) + nbc

```

C— Now compute the aqueous phase component flux at the well.

C

```

  if(lbalc) then
    do 370 i = 1, ipt(4)
      ipt3 = i + ipt(3)
      ic = icp(ipt3)
      icxmf = ipt(9) + (i-1)*ipt(1) + nbc
      fkeep = den(nbcw) * bcf(nbcw) * xmf(icxmf)
      +
        * cmw(ic) * t(8)
      wsink(5+ic) = wsink(5+ic) + fkeep
    370 continue
  end if
  if(lbalp) then
    nbcwd = ipt(41)+nbcw
    ipt24 = ipt(24)
    wsink(2) = wsink(2) + den(nbcwd)
    +
      * source(jstr+ipt24+jj) * t(8)
  end if
  end if
  360 continue
endif
end if
C
C— Sum the phase mass fluxes and sinks. Change the molar
C— component fluxes into mass form and account for the time
C— step. Also sum the cumulative mass component fluxes.
C

```

```

  if(lbalp) then
    csink(1) = csink(1) + sink(1)
    csink(2) = csink(2) + sink(2)
    cwsink(1) = cwsink(1) + wsink(1)
    cwsink(2) = cwsink(2) + wsink(2)
    tsink = sink(1) + sink(2)
    tcsink = csink(1) + csink(2)
    twsink = wsink(1) + wsink(2)
    tcwsink = cwsink(1) + cwsink(2)
    cmf(1) = cmf(1) + flux(1)
    cmf(2) = cmf(2) + flux(2)
    tflux = flux(1) + flux(2)
    tcflux = cmf(1) + cmf(2)
    do 379 ic = 1,2
      tphe = tphe + phex(ic)
      cphe(ic) = cphe(ic) + phex(ic)
      tpehe = tpehe + phex(ic)
    379 tmass1 = tmass1 + str1(ic)
    do 380 ic = 3,5
      tphe = tphe + phex(ic)
      cphe(ic) = cphe(ic) + phex(ic)
      tpehe = tpehe + phex(ic)
    380 tmass1 = tmass1 + str1(ic)
  end if
  if(lbalc) then
    crsink(2) = crsink(2) + rsink(2)
    csink(5) = csink(5) + sink(5)
    tsink = tsink + sink(5)
    tcsink = tcsink + csink(5)
    crsink(5) = crsink(5) + rsink(5)
    trsink = rsink(2) + rsink(5)
    tersink = crsink(2) + crsink(5)
    do 390 ic = 1, istop
      csink(5+ic) = csink(5+ic) + sink(5+ic)
      cwsink(5+ic) = cwsink(5+ic) + wsink(5+ic)
      crsink(5+ic) = crsink(5+ic) + rsink(5+ic)
      flux(5+ic) = flux(5+ic)
      csflux(5+ic) = csflux(5+ic) + sflux(5+ic)
    390 cmf(5+ic) = cmf(5+ic) + flux(5+ic)
  end if
  if(lpcbal) then
    if(lprnt(27)) then
    C
    C— Write the phase material balance report.
    C
      write (25,507) its,t(9)
      write (25,502) 'Phase mass (kg)',
      +
        (str1(i),i=1,5),tmass1
      write (25,508)
      write (25,502) 'Delta storage (kg)',
      +
        (str1(i)-str(i),i=1,5),tmass1-tmass
      write (25,502) 'Boundary flux (kg)',
      +
        (flux(i),i=1,5),tflux
    end if
  end if

```

```

write (25,502) 'Sources (kg)',
+ (sink(i),i=1,5),tsink
write (25,502) 'Well sources (kg)',
+ (wsink(i),i=1,5),twsink
write (25,502) 'Reactions (kg)',
+ (rsink(i),i=1,5),trsink
write (25,502) 'Phase transfer (kg)',
+ (phex(i),i=1,5),tphex
do 400 i = 1,5
400 resid(i) = str1(i) - str(i) - flux(i)
+ - wsink(i) - phex(i)
write (25,502) 'Residual (kg)'
+ ,(resid(i),i=1,5),tmass1 - tmass0
+ - tflux - twsink - tphex
do 405 i = 1,5
if(str0(i).gt.zer0) then
perr1(i) = 1.0d2*dabs(resid(i))/str0(i)
else
perr1(i) = zer0
end if
405 continue
if(tmass0.gt.zer0) then
write (25,502) 'Time step error1(%)',
+ (perr1(i),i=1,5), 1.0d2*dabs(tmass1 - tmass0
+ - tflux - twsink - tphex)/tmass0
else
write (25,502) 'Time step error1(%)',
+ (perr1(i),i=1,5), zer0
end if
do 406 i = 1,5
term = dmax1(dabs(wsink(i))+dabs(flux(i))+dabs(phex(i)),
+ dabs(str1(i)-str(i)))
if(term.ne.zer0) then
perr1(i) = 1.0d2*dabs(resid(i))/term
else
perr1(i) = zer0
end if
406 continue
term = dmax1(dabs(twsink)+dabs(tflux)+dabs(tphex),
+ dabs(tmass1-tmass0))
if(term.ne.zer0) then
tperr1 = 1.0d2*dabs(tmass1 - tmass0 - tflux - twsink
+ - tphex)/term
else
tperr1 = zer0
end if
write (25,502) 'Time step error2(%)',
+ (perr1(i),i=1,5), tperr1
do 407 i = 1,5
if(str1(i)-str(i).ne.zer0) then
perr1(i) = 1.0d2*(rone-(dabs(wsink(i))+flux(i)
+ +phex(i)) / (dabs(str1(i)-str(i))))
else
perr1(i) = zer0
end if
407 continue
if(tmass1-tmass0.ne.zer0) then
tperr1 = 1.0d2*(rone-(dabs(twsink+tflux+tphex)
+ / (dabs(tmass1-tmass0))))
else
tperr1 = zer0
end if
write (25,502) 'Time step error3(%)',
+ (perr1(i),i=1,5), tperr1
write (25,509)
write (25,502) 'Delta storage (kg)',
+ (str1(i)-str0(i),i=1,5),tmass1-tmass0
write (25,502) 'Boundary flux (kg)',
+ (cmf(i),i=1,5),tflux
write (25,502) 'Sources (kg)',
+ (csink(i),i=1,5),tcsink
write (25,502) 'Well sources (kg)',
+ (cwsink(i),i=1,5),tcwsink
write (25,502) 'Reactions (kg)',
+ (crsink(i),i=1,5),tcrsink
write (25,502) 'Phase transfer (kg)',
+ (cphex(i),i=1,5),tcphex
do 410 i = 1,5

```

```

410 resid(i) = str1(i) - str0(i) - cmf(i)
+ - cwsink(i) - crsink(i) - cphex(i)
write (25,502) 'Residual (kg)'
+ ,(resid(i),i=1,5),tmass1 - tmass0
+ - tcwsink - tflux
do 425 i = 1,5
if(str0(i).gt.zer0) then
perr2(i) = 1.0d2*dabs(resid(i))/str0(i)
else
perr2(i) = zer0
end if
425 continue
if(tmass0.ne.zer0) then
write (25,502) 'Cumulative err1 (%)',
+ (perr2(i),i=1,5),dabs(tmass1 - tmass0
+ - tcwsink - tflux - tcphex)*1.0d2/tmass0
else
write (25,502) 'Cumulative err1 (%)',
+ (perr2(i),i=1,5),zer0
end if
do 426 i = 1,5
term = dmax1(dabs(cwsink(i))+dabs(cmf(i))
+ +dabs(cphex(i)),dabs(str1(i)-str0(i)))
if(term.ne.zer0)
then
perr2(i) = 1.0d2*dabs(resid(i))/term
else
perr2(i) = zer0
end if
426 continue
term = dmax1(dabs(tcwsink)+dabs(tflux)+dabs(tcphex),
+ dabs(tmass1 - tmass0))
if(term.ne.zer0) then
tperr2 = 1.0d2*dabs(tmass1 - tmass0 - tcwsink - tflux
+ - tcphex)/term
else
tperr2 = zer0
end if
write (25,502) 'Cumulative err2 (%)',
+ (perr2(i),i=1,5), tperr2
do 427 i = 1,5
if(str1(i)-str0(i).ne.zer0) then
perr2(i) = 1.0d2*(rone-(dabs(cwsink(i))+cmf(i)
+ +cphex(i)) / (dabs(str1(i)-str0(i))))
else
perr2(i) = zer0
end if
427 continue
if(tmass1 - tmass0.ne.zer0) then
tperr2 = 1.0d2*(rone-(dabs(tcwsink+tflux+tcphex)
+ / (dabs(tmass1 - tmass0))))
else
tperr2 = zer0
end if
write (25,502) 'Cumulative err3 (%)',
+ (perr2(i),i=1,5), tperr2
if(lctrl(3)) write(25,502) 'Biomass (kg)',
+ zer0,zer0,zer0,zer0,bmass,zer0
C
C— Write the component material balance report.
C
if(lctrl(2)) then
write (25,506) its,t(9),(cname(i),i=1,istop)
write (25,502) 'Gas Phase (kg)',
+ (cmass1(i),i=1,istop)
write (25,502) 'Aqueous Phase (kg)',
+ (cmass1(istop+i),i=1,istop)
write (25,502) 'NAPL (kg)',
+ (cmass1(2*istop+i),i=1,istop)
write (25,502) 'Solid Phase (kg)',
+ (cmass1(3*istop+i),i=1,istop)
write (25,502) 'Biophase (kg)',
+ (cmass1(4*istop+i),i=1,istop)
write (25,502) 'Total (kg)',
+ (str1(5+i),i=1,istop)
write (25,508)
write (25,502) 'Delta storage (kg)',
+ (str1(i+5)-str(i+5),i=1,istop)

```



```

write (25,502) 'Boundary flux (kg)',
+   (flux(i+5),i=1,istop)
write (25,502) 'Surface flux (kg)',
+   (sflux(i+5),i=1,istop)
write (25,502) 'Sources (kg)',
+   (sink(i+5),i=1,istop)
write (25,502) 'Well sources (kg)',
+   (wsink(i+5),i=1,istop)
write (25,502) 'Reactions (kg)',
+   (rsink(i+5),i=1,istop)
do 430 i = 6,istop+5
  resid(i) = str1(i) - str(i)
+   - wsink(i) - rsink(i) - flux(i)
write (25,502) 'Residual (kg)'
+   ,(resid(i+5),i=1,istop)
do 435 i = 6,istop+5
  if(str0(i).gt.zer0) then
    perr1(i) = 1.0d2*dabs(resid(i))/str0(i)
  else
    perr1(i) = zer0
  end if
  continue
write (25,502) 'Time step error1(%)',
+   (perr1(i+5),i=1,istop)
do 436 i = 6,istop+5
  term = dmax1(dabs(rsink(i))+dabs(wsink(i))
+   +dabs(flux(i)),dabs(str1(i)-str(i)))
  if(term.ne.zer0) then
    perr1(i) = 1.0d2*dabs(resid(i))/term
  else
    perr1(i) = zer0
  end if
  continue
write (25,502) 'Time step error2(%)',
+   (perr1(i+5),i=1,istop)
do 437 i = 6,istop+5
  if(str1(i)-str(i).ne.zer0) then
    perr1(i) = 1.0d2*(rone-(dabs(rsink(i)+wsink(i))
+   +flux(i)) / (dabs(str1(i)-str(i))))
  else
    perr1(i) = zer0
  end if
  continue
write (25,502) 'Time step error3(%)',
+   (perr1(i+5),i=1,istop)
write (25,509)
write (25,502) 'Delta storage (kg)',
+   (str1(i+5)-str0(i+5),i=1,istop)
write (25,502) 'Boundary flux (kg)',
+   (cmf(i+5),i=1,istop)
write (25,502) 'Surface flux (kg)',
+   (csflux(i+5),i=1,istop)
write (25,502) 'Sources (kg)',
+   (csink(i+5),i=1,istop)
write (25,502) 'Well sources (kg)',
+   (cwsink(i+5),i=1,istop)
write (25,502) 'Reactions (kg)',
+   (crsink(i+5),i=1,istop)
do 440 i = 6,istop+5
  resid(i) = str1(i) - str0(i) -
+   cwsink(i) - crsink(i) - cmf(i)
write (25,502) 'Residual (kg)'
+   ,(resid(i+5),i=1,istop)
do 445 i = 6,istop+5
  if(str0(i).gt.zer0) then
    perr2(i) = 1.0d2*dabs(resid(i))/str0(i)
  else
    perr2(i) = zer0
  end if
  continue
write (25,502) 'Cumulative err1 (%)',
+   (perr2(i+5),i=1,istop)
do 446 i = 6,istop+5
  term = dmax1(dabs(crsink(i))+dabs(cwsink(i))
+   +dabs(cmf(i)),dabs(str1(i)-str0(i)))
  if(term.ne.zer0) then

```

```

    perr2(i) = 1.0d2*dabs(resid(i))
    /term
  else
    perr2(i) = zer0
  end if
  continue
write (25,502) 'Cumulative err2 (%)',
+   (perr2(i+5),i=1,istop)
do 447 i = 6,istop+5
  if(crsink(i)+cwsink(i)+cmf(i).ne.zer0) then
    perr2(i) = 1.0d2*(rone-(dabs(crsink(i)+cwsink(i))
+   +cmf(i))/(dabs(str1(i)-str0(i))))
  else
    perr2(i) = zer0
  end if
  continue
write (25,502) 'Cumulative err3 (%)',
+   (perr2(i+5),i=1,istop)
end if
else
C
C— Write the time series style phase material balance.
C
  if(tmass0.ne.zer0) then
    tperr1 = dabs(tmass1 - tmass - tsink
+   - twsink - tflux)*1.0d2/tmass0
  else
    tperr1 = zer0
  end if
  write(25,500) t(9)/3600.0d0/24.0d0,(str1(i),i=1,5)
+   ,tmass1,tcflux,tcsink,tcwsink,tcrsink,tperr1
C
C— Write the time series style component material balance.
C
  if(lctrl(2)) then
    do 455 i = 1, ipt(15)
      write(28+i,500) t(9)/3600.0d0/24.0d0
+   ,(cmass1(ii*istop+i),ii=0,4),str1(i+5),cmf(i+5)
+   ,csink(i+5),cwsink(i+5),crsink(i+5),csflux(i+5)
    end if
  end if
  do 450 i = 1, ipt(65)
    first(i) = zer0
  return
C
C— formats
500 format(e11.5,11e11.4)
501 format(/'*** TIME STEP =',i6,5x,'SIMULATION TIME (s) =',e15.5/
+ 'INITIAL REPORT',8x,'Gas',8x,'Aqueous',4x,'NAPL',7x,'Solid'
+ ,6x,'Biophase',3x,'Total')
c 502 format(
c + ' ',a19,9e11.4)
502 format(
+ ' ',a,9e11.4)
503 format (
+ ' ',a10 ,(kg) ',9e11.4)
504 format ('Phase totals (kg); errors (%)',/
+ ' time (day) ',Gas',8x,'Aqueous',4x,'NAPL',7x,'Solid',6x
+ ,'Biophase',3x,'Total',6x,'Flux',7x,'Source',5x,'Well',7x,
+ 'Reaction',3x,'Step err')
505 format (a10,'totals (kg); errors (%)',/
+ ' time (day) ',Gas',8x,'Aqueous',4x,'NAPL',7x,'Solid',6x
+ ,'Biophase',3x,'Total',6x,'Flux',7x,'Source',5x,'Well',7x,
+ 'Reaction',3x,'Surface')
506 format(/'*** TIME STEP =',i6,5x,'SIMULATION TIME (s) =',e15.5/
+ 'COMPONENT REPORT',6x,9a11)
507 format(/'*** TIME STEP =',i6,5x,'SIMULATION TIME (s) =',e15.5/
+ 'PHASE REPORT',10x,'Gas',8x,'Aqueous',4x,'NAPL',7x,'Solid'
+ ,6x,'Biophase',3x,'Total')
508 format('Time step balance')
509 format('Cumulative balance')
c
end

```

Subroutine - commnt.f

```
C-----
C
C COMMNT.f - Subroutine which reads comment lines from 'infrom'
C and ignores lines starting with '#' and prints to
C 'outto' lines starting with '&'.
C
C Argument list - infrom: integer; number of input device
C ioutto: integer; number of output device
C-----
C
C subroutine COMMNT(infrom,ioutto)
C integer infrom,ioutto
C character*80 echo
```

```
1 read (infrom,501) echo
if (echo(:1) .eq. '#') then
  goto 1
else if (echo(:1) .eq. '&') then
  write(ioutto,501) echo(2:)
  goto 1
end if
backspace infrom
501 format (a)
return
end
```

Include File - dimen.inc

```
C-----
C
C DIMEN.inc - Declares array dimensions of common block variables.
C-----
C implicit real*8 (a-h,k,o-x,z), logical (l), integer (i-j,m-n)
C + , character*20 (y)
C
C Define array dimensions:
C nmx = maximum number of nodes
C nelmx = maximum number of elements
C nmbk = maximum number of material property blocks
C nxmax = maximum number of horizontal blocks in the generated grid
C nzmax = maximum number of vertical blocks in the generated grid
C nnstk = maximum number nodal variables in stacked storage
C ncmpo = maximum number of organic components to be simulated
C ncmpb = 1 if biodegradation is to be considered
C ncmp = maximum number of total components to be simulated
C = 3(always) + ncmpb + ncmpo + 1(if nutrient is considered)
C ncmp2 = 2 x maximum number of total components to be simulated
C ncmp5 = 5 x maximum number of total components to be simulated
C nsolve = maximum number of unknowns in the linear system
C = nn2
C icnl = number of nonzero entries in the coefficient matrix
C irnl = number of nonzero entries in the coefficient matrix
C maxdima = maximum number of elements in matrix A in Ax=b
C nmd = maximum number of dispersion coefficients
C nmbc = maximum number of component boundary conditions
C !nnstk = nmx+nnmx/3,
C parameter (nmx = 2350, nelmx = 4500, nxmax = 100, nzmax = 100,
C + nmbk = 3, ncmpb = 1, ncmpo = 2, ncmp = ncmpo+ncmpb+3,
C + nel3 = 3*nelmx, nel4 = 4*nelmx,
C + nn2 = 2*nnmx, nn3 = 3*nnmx, nn4 = 4*nnmx, nn6 = 6*nnmx,
```

```
+ nnstk = nmx*1.05, nns10 = 10*nnstk,
+ nnstk2 = 2*nnstk, nnstk3 = 3*nnstk,
+ nnstk4 = 4*nnstk, nnstk8 = 8*nnstk,
+ ncmp2 = 2*ncmp, ncmp5 = 5*ncmp, nbcmp = ncmp*nmbk,
+ ncmp5 = ncmp+5, ncsqd = ncmp*ncmp*nnmx,
+ nzmax6 = 6*nzmax, nmd = 6*nelmx, nmbc = 2*ncmp*nnmx,
+ nmf = nmx*(ncmp+2*(ncmp-1+ncmpo)),
+ nmfs = nnstk*(ncmp+2*(ncmp-1+ncmpo))
parameter (nsolve = nn2, icnl = 900000, irnl = 900000)
parameter (patm=101327.0d0, pi=3.14159265360d0, third=1.0d0/3.d0,
+ pthird=2.0d0*pi*third, tstd = 293.150d0
+ , tabs = 273.150d0, zer0=0.0d0, rone=1.0d0)
C
C R in (m3 Pa) / (mole degree K)
parameter (r = 0.08206d0 * 101327.0d0 / 1000.0d0)
parameter (srwmin = 1.0d-16, sgtest = 0.05d0)
parameter (u=0.10d0, mtype=1)
C
C Parameter values to modify code
C
parameter (xmino=1.0d-3, xround=1.0d-16, smino=1.0d-16)
C
C Include logical control variables
common /cbl/ lctrl(50), lprnt(0:30), lcon(50), lplt(20)
C
C Include integer control variables
common /cbl/ ipt(0:90), icp(0:50)
C
C Include real*8 scalars
common /cbs/ t(50), b, xkex, xden, xbom, xbok, wvis, kd, xinit, xbmim
+ , xbmam, qwel, rwell, zwell, trefqg, wtdpth, caplen
```

Subroutine - disper.f

```
C-----
C
C DISPER.f - Subroutine which computes the phase dependent
C portion of the dispersion tensor and the tortuosity.
C-----
C Arguments: iphase - integer scalar denoting the phase
C gas phase: iphase = 1
C aqueous phase: iphase = 2
C
C Required Control Flags:
```

```
C lctrl(18) - logical variable denoting type of q calculation
C lctrl(18) = .true. - nodal FEM q
C lctrl(18) = .false. - element average q
C-----
C subroutine DISPER(iphase)
C include 'dimen.inc'
C
C Declare and define common block variables.
C
common /cbl/ matel(nelmx), nodel(nel3), nodept(nmx), nelpt(nel3),
```

```

+      matpt(nn6)
common /cb2c/ q(ne14)
common /cb3/ sat(nnstk3)
common /cb6b/ por(nelmx),srw(nnstk)
common /cb70/ d(nmd),tort(nelmx),bdist(nmb1k),bdisl(nmb1k)
ipt2s = (iphase-1)*ipt(2)
ipt1 = ipt(1)
ipt0 = ipt(0)
C
C— Compute the tortuosity after Millington and Quirk (1961).
C— Averaged saturations are used since the dispersion tensor is an
C— element wise constant.
C
do 100 i= 1,ipt(0)
  iel3 = i*3
  iel1 = iel3-2
  iel2 = iel3-1
  i1 = nodel(iel1)
  i1s = nodept(i1)+nelpt(iel1)
  i1ps = i1s + ipt2s
  i2 = nodel(iel2)
  i2s = nodept(i2)+nelpt(iel2)
  i2ps = i2s + ipt2s
  i3 = nodel(iel3)
  i3s = nodept(i3)+nelpt(iel3)
  i3ps = i3s + ipt2s
  satavg = third * (sat(i1ps)+sat(i2ps)+sat(i3ps))
  poravg = por(i)
C
C— Millington and Quirk (1959).
C
c      tort(i) = ((poravg*satavg)**(third * 10.0d0)) / poravg**2
C
C— Millington and Quirk (1961) for variably saturated media.
C
      tort(i) = ((poravg*satavg)**(third * 7.0d0)) / poravg**2
C
C— Calculate averaged q.
C
      if (lctrl(18)) then

```

```

      iptq = (iphase-1)*ipt1*2
      qx = dabs(third * (q(iptq+i1)+q(iptq+i2)+q(iptq+i3)))
      qz = dabs(third * (q(iptq+ipt1+i1)+q(iptq+ipt1+i2)
+      +q(iptq+ipt1+i3)))
      else
      iptq = i+(iphase-1)*ipt(67)
      qx = dabs(q(iptq))
      qz = dabs(q(iptq+ipt0))
      end if
      qxz = dsqrt(qx**2 + qz**2)
C
C— Convert q to velocity.
C
      if (satavg.ge.sgtest) then
      vxz = qxz/(poravg*satavg)
      else
      vxz = zero
      end if
      id = (iphase-1)*ipt(68)
C
C— Calculate dispersion.
C
      if (dabs(qxz).gt.1.0d-15) then
      idp3xi = id+3*i
      imat = matel(i)
      d(idp3xi-2) = bdist(imat)*vxz
+      + bdisl(imat)*(qx*qx)/(qxz*poravg*satavg)
      d(idp3xi-1) = bdisl(imat)*dabs(qx*qz)/(qxz*poravg*satavg)
+      + bdist(imat)*vxz
+      + bdisl(imat)*(qz*qz)/(qxz*poravg*satavg)
      else
      idp3xi = id+3*i
      d(idp3xi-2) = zero
      d(idp3xi-1) = zero
      d(idp3xi) = zero
      end if
100 continue
      return
      end

```

Subroutine - error.f

```

C
C
C ERRMESSAGE.f - Finds and prints error and warning messages
C from the error file.
C Warning if itype = 1; fatal error if itype = 0
C
C
subroutine ErrorMessage (MessNum,itype,ierr)
implicit real*8 (a-h,k,o-z), logical (l), integer (i-j,m-n)
character line*80
c
c— Find the specified error or warning in the error file.
rewind (14)
1 read (14,800) line
800 format (a)
If (line(:1).eq. '#') then
  GoTo 1
Else
  backspace 14
  read (14,*) number,nlines
Endif
c

```

```

c— Print the error or warning to the screen and the error file if
c— opened.
If (number .eq. MessNum) then
  write (ierr,*)
  If (itype .eq. 1) then
    write (ierr,*) '***** WARNING *****'
  Else if (itype .eq. 0) then
    write (ierr,*) '***** FATAL ERROR *****'
  EndIf
  Do i = 1,nlines
    read (14,800) line
    write (ierr,*) line(2:80)
  EndDo
  Else
    GoTo 1
  EndIf
c
c— If a fatal error then terminate execution.
If (itype .eq. 0) stop
return
end

```

Subroutine - flow.f

```

C-----
C
C FLOW.f - Subroutine which solves the phase balance equations.
C           Solution of the mobile aqueous and gaseous phases is
C           done simultaneously.
C
C Required Control Flags:
C
C i(13) - convergence criterion for pressures
C ipt(27) - integer variable indicating type of domain
C           ipt(27) = 0 - xz domain
C           ipt(27) = 1 - rz domain
C ipt(31) - maximum phase balance iterations, also used as the
C           criterion for decreasing dt in phase balance
C ipt(34) - maximum number of iterations in phase balance for
C           increasing dt
C lctrl(7) - logical variable controlling type of FEM
C           solution for flow
C           lctrl(7) = .true. - use mass lumping
C           lctrl(7) = .false. - full FEM solution
C lctrl(12) - logical variable denoting presence of a well
C           lctrl(12) = .true. - well present
C           lctrl(12) = .false. - well not present
C lctrl(13) - logical variable denoting compositional dependence
C           of the gas phase viscosity
C           lctrl(13) = .true. - gas phase viscosity is
C           dependent on composition
C           lctrl(13) = .false. - gas phase viscosity is not
C           dependent on composition
C lctrl(14) - logical variable controlling coupling of flow and
C           transport solutions
C           lctrl(14) = .true. - exchange couples flow and
C           transport solutions
C           lctrl(14) = .false. - flow and transport
C           solutions not coupled
C
C Control Flags computed internally in routine:
C
C ipt(36) - flag specifying time step modification

```

```

C-----
C
C subroutine FLOW (its,it)
C include 'dimen.inc'
C common /cb1/ matel(nelm),nodel(nel3),nodept(nnm),nelpt(nel3),
C + matpt(nn6)
C common /cb1e/ xnode(nnm),znode(nnm),rbar(nelm),area(nelm)
C common /cb1d/ gama(nel3),beta(nel3)
C common /cb2/ p(nn3)
C common /cb2b/ pt(nn3)
C common /cb3/ sat(nnstk3)
C common /cb4/ satk(nnstk2),cc(nnstk)
C common /cb5a/ bphi(nmb),bpermh(nmb),bpermv(nmb)
C common /cb6/ pmob(nnstk4)
C common /cb6b/ por(nelm),srw(nnstk)
C common /cb6c/ temp(nnm)
C common /cb6d/ dtemp(nzmax6),idepth(nnm)
C common /cb7/ cvvis(ncmp),gamma(ncsqd)
C common /cb8/ vis(nnm),pmw(nn3)
C common /cb9/ xmf(nmf)
C common /cb10/ den(nn6)
C common /cb10b/ dden(nn6),pmwt(nn3),dent(nn6)
C common /cb11/ pex(nns10),rxnp(nn2)
C common /cb30/ ibc(nnm)
C common /cb31/ source(nn2)
C common /cb40/ a(icnl),rhs(nsolve),w(icnl)
C common /cb41/ im(icnl),icn(icnl),iw(icnl,8),ikeep(icnl,5)
C common /cb41b/ nbw(0:2),ia
C common /cb42/ amb(icnl),fmb(nsolve)
C
C
C Local arrays.
C dimension ael(36),bel(36),fel(6),nd(3),dn(3),c(11)
C data isparge /0/
C
C
C Initialize current iterations saturations.
C Do i = 1,2*ipt(2)
C   satk(i) = sat(i)
C EndDo
C
C

```

```

C-----
C
C Update gas phase viscosity.
C If (lctrl(13)) then ! compute a mixture viscosity
C   Do node = 1,ipt(1)
C     vis(node) = zero
C     Do j = 1,ipt(3)
C       ii = jcp(i)
C       if (lctrl(10)) then
C         cvvist = cvvis(ii)
C       else
C         cvvist = cvvis(ii)
C       + dtemp(ipt(88)+(ii-1)*ipt(89)+idepth(node))
C     end if
C     i3 = (i-1) * ipt(3) + (node-1)*ipt(3)*ipt(3)
C     sum = 0.0d0
C     Do j = 1,ipt(3)
C       jj = jcp(j)
C       sum = sum + xmf((jj-1)*ipt(1)+node) * gamma(i3+j)
C     EndDo
C     vis(node) = vis(node) + xmf((ii-1)*ipt(1)+node) *
C   + cvvist / sum
C   EndDo
C   vis(node) = 1.0d0 / vis(node)
C EndDo
C EndIf
C
C Loop over maximum number of iterations.
C do 100 it = 1,ipt(31)
C
C Update chord slope approx for capacity coeffs and phase mobilities.
C call mobil (it)
C
C Update flow distribution in the well bore.
C if (lctrl(12)) then
C   iptbc = ipt(18)+ipt(19)+ipt(20)+ipt(21)+ipt(22)+ipt(23)
C   sum = 0.0d0
C   do 60 i = 1,ipt(24)-1
C     jel = ibc(iptbc+ipt(24)+i) ! well screen element number
C     nd1 = ibc(iptbc+i) ! top node number in element
C     nd2 = ibc(iptbc+i+1) ! bottom node number in element
C     zel = (znode(nd2)-znode(nd1)) ! element vertical height
C     i3 = 3*(jel-1)
C     Do j = 1,3 ! determine stacked nodel position
C       if (nd1 .eq. nodel(i3+j))
C     + ndstk1 = nodept(nd1) + nelpt(i3+j)
C       if (nd2 .eq. nodel(i3+j))
C     + ndstk2 = nodept(nd2) + nelpt(i3+j)
C     EndDo
C     sum = sum + zel * (pmob(ndstk1) + pmob(ndstk2))
C     if (qwell .lt. 0.0d0) sum = sum + zel * (
C     + pmob(2*ipt(2)+ndstk1) + pmob(2*ipt(2)+ndstk2))
C   60 continue
C   if (ipt(27) .eq. 1) sum = sum * 2.0d0 * pi * rwell
C   if (sum .eq. 0.0d0) then
C     isparge = 1
C   else
C     sum = 1.0d0 / sum
C   endif
C
C
C Allocate specific discharge along well screen.
C ipt2 = ipt(22) + ipt(23)
C nsrwell = ipt(24)
C if (qwell .lt. 0.0d0) nsrwell = nsrwell + ipt(24)
C do 62 i = 1,nsrwell ! zero specific discharges at well bore
C   source(ipt2+i) = 0.0d0
C   if (qwell .lt. 0.0d0) source(ipt2+ipt(24)+i) = 0.0d0
C 62 continue
C do 63 i = 1,ipt(24)-1 ! allocate specific discharge
C   jel = ibc(iptbc+ipt(24)+i) ! well screen element number
C   nd1 = ibc(iptbc+i) ! top node number in element
C   nd2 = ibc(iptbc+i+1) ! bottom node number in element
C   zel = znode(nd2) - znode(nd1)
C   if (ipt(27) .eq. 1) zel = zel * 2.0d0 * pi * rwell
C   i3 = 3*(jel-1)
C   do 64 j = 1,3 ! determine stacked nodel position
C     if (nd1 .eq. nodel(i3+j))
C   + ndstk1 = nodept(nd1) + nelpt(i3+j)
C     if (nd2 .eq. nodel(i3+j))
C   64 + ndstk2 = nodept(nd2) + nelpt(i3+j)

```

```

if (ispurge .eq. 0) then
  source(ipt2+i) = source(ipt2+i) + zel * pmob(ndstk1) *
+   qwell * sum ! gas discharge at top node
  source(ipt2+i+1) = source(ipt2+i+1) + zel *
+   pmob(ndstk2) * qwell * sum ! lower node gas discharge
else if (ispurge .eq. 1) then
  source(ipt2+i) = source(ipt2+i) +
+   zel*qwell/zwell/2.0d0
  source(ipt2+i+1) = source(ipt2+i+1) +
+   zel*qwell/zwell/2.0d0
endif
if (qwell .lt. 0.0d0) then
  source(ipt2+ipt(24)+i) = source(ipt2+ipt(24)+i) +
+   zel * pmob(2*ipt(2)+ndstk1) * qwell * sum
  source(ipt2+ipt(24)+i+1) = source(ipt2+ipt(24)+i+1) +
+   zel * pmob(2*ipt(2)+ndstk2) * qwell * sum
endif
63 continue
end if

c
c— Zero the local 'a' and 'rhs' vectors.
Do i = 1,2*ipt(1)*nbw(2)
  a(i) = zero
EndDo
Do i = 1,2*ipt(1)
  rhs(i) = zero
EndDo

c
c— Loop over the number of elements.
do 101 jel = 1, ipt(0)
  i3 = (jel-1)*3
  mblk = matel(jel)
  nd1 = nodel(i3+1) ! element node numbers
  nd2 = nodel(i3+2)
  nd3 = nodel(i3+3)
  nd(1) = nodel(i3+1) ! element node numbers
  nd(2) = nodel(i3+2)
  nd(3) = nodel(i3+3)
  ndstk1 = nodept(nd1) + nelpt(i3+1) ! node position in stack
  ndstk2 = nodept(nd2) + nelpt(i3+2)
  ndstk3 = nodept(nd3) + nelpt(i3+3)
  c(9) = rbar(jel) / (12.0d0 * area(jel)) ! element constants
  c(10) = rbar(jel) * area(jel) * por(jel) / (t(8) * 12.0d0)
  c(11) = rbar(jel) / 24.0d0

c
c— Compute the element mass matrix; either lumped or consistent.
if (lctrl(7)) then
  Do i = 1,36
    ael(i) = zero
  EndDo
  c(1) = -4.d0 * c(10) * cc(ndstk1)
  c(3) = 4.d0 * c(10) * sat(ndstk1)
  ael(1) = c(1)
  ael(4) = -c(1)
  ael(7) = -c(1)
  ael(10) = c(3) * (1.0d0 / (r*temp(nd1)*den(nd1))) + c(1)
  c(1) = -4.d0 * c(10) * cc(ndstk2)
  c(3) = 4.d0 * c(10) * sat(ndstk2)
  ael(14) = c(1)
  ael(17) = -c(1)
  ael(20) = -c(1)
  ael(23) = c(3) * (1.0d0 / (r*temp(nd2)*den(nd2))) + c(1)
  c(1) = -4.d0 * c(10) * cc(ndstk3)
  c(3) = 4.d0 * c(10) * sat(ndstk3)
  ael(27) = c(1)
  ael(30) = -c(1)
  ael(33) = -c(1)
  ael(36) = c(3) * (1.0d0 / (r*temp(nd3)*den(nd3))) + c(1)
else
  c(1) = -c(10) * cc(ndstk1)
  c(3) = c(10) * sat(ndstk1)
  ael(1) = 2.0d0 * c(1)
  ael(2) = c(1)
  ael(3) = c(1)
  ael(4) = -2.0d0 * c(1)
  ael(5) = -c(1)
  ael(6) = -c(1)
  ael(7) = -2.0d0 * c(1)

```

```

  ael(8) = -c(1)
  ael(9) = -c(1)
  ael(11) = c(3) * (1.0d0 / (r*temp(nd1)*den(nd1))) + c(1)
  ael(12) = ael(11)
  ael(10) = 2.0d0 * ael(11)
  c(1) = -c(10) * cc(ndstk2)
  c(3) = c(10) * sat(ndstk2)
  ael(13) = c(1)
  ael(14) = 2.0d0 * c(1)
  ael(15) = c(1)
  ael(16) = -c(1)
  ael(17) = -2.0d0 * c(1)
  ael(18) = -c(1)
  ael(19) = -c(1)
  ael(20) = -2.0d0 * c(1)
  ael(21) = -c(1)
  ael(22) = c(3) * (1.0d0 / (r*temp(nd2)*den(nd2))) + c(1)
  ael(23) = 2.0d0 * ael(22)
  ael(24) = ael(22)
  c(1) = -c(10) * cc(ndstk3)
  c(3) = c(10) * sat(ndstk3)
  ael(25) = c(1)
  ael(26) = c(1)
  ael(27) = 2.d0 * c(1)
  ael(28) = -c(1)
  ael(29) = -c(1)
  ael(30) = -2.d0 * c(1)
  ael(31) = -c(1)
  ael(32) = -c(1)
  ael(33) = -2.d0 * c(1)
  ael(34) = c(3) * (1.d0 / (r*temp(nd3)*den(nd3))) + c(1)
  ael(35) = ael(34)
  ael(36) = 2.d0 * ael(34)
endif

```

c
c— Compute the element stiffness matrix for the aqueous equation.

```

  imbx1 = 2*ipt(2)+ndstk1
  imbx2 = 2*ipt(2)+ndstk2
  imbx3 = 2*ipt(2)+ndstk3
  imbz1 = 3*ipt(2)+ndstk1
  imbz2 = 3*ipt(2)+ndstk2
  imbz3 = 3*ipt(2)+ndstk3
  c(3) = pmob(imbz1) + pmob(imbz2) + pmob(imbz3)
  c(4) = bpermh(mblk) * c(3) ! c(4)=sum of x mobilities
  dn(1) = den(4*ipt(1)+nd1)
  dn(2) = den(4*ipt(1)+nd2)
  dn(3) = den(4*ipt(1)+nd3)
  c(5) = pmob(imbz1)/dn(1) + pmob(imbz2)/dn(2) +
+   pmob(imbz3)/dn(3)
  c(6) = bpermh(mblk) * c(5) ! c(6)=x mobil/density
  c(7) = beta(i3+1)*dn(1) + beta(i3+2)*dn(2) +
+   beta(i3+3)*dn(3) ! c(7)=sum of density * beta
  c(8) = gama(i3+1)*dn(1) + gama(i3+2)*dn(2) +
+   gama(i3+3)*dn(3) ! c(8)=sum of density * gamma
  c(1) = c(9) * (c(4)*beta(i3+1) - 0.250d0 *
+   (c(6)+pmob(imbx1)/dn(1)) * c(7))
  c(2) = c(9) * (c(3)*gama(i3+1) - 0.250d0 *
+   (c(5)+pmob(imbz1)/dn(1)) * c(8))
  bel(1) = beta(i3+1)*c(1) + gama(i3+1)*c(2)
  bel(13) = beta(i3+2)*c(1) + gama(i3+2)*c(2)
  bel(25) = beta(i3+3)*c(1) + gama(i3+3)*c(2)
  c(1) = c(9) * (c(4)*beta(i3+2) - 0.250d0 *
+   (c(6)+pmob(imbx2)/dn(2)) * c(7))
  c(2) = c(9) * (c(3)*gama(i3+2) - 0.250d0 *
+   (c(5)+pmob(imbz2)/dn(2)) * c(8))
  bel(2) = beta(i3+1)*c(1) + gama(i3+1)*c(2)
  bel(14) = beta(i3+2)*c(1) + gama(i3+2)*c(2)
  bel(26) = beta(i3+3)*c(1) + gama(i3+3)*c(2)
  c(1) = c(9) * (c(4)*beta(i3+3) - 0.250d0 *
+   (c(6)+pmob(imbx3)/dn(3)) * c(7))
  c(2) = c(9) * (c(3)*gama(i3+3) - 0.250d0 *
+   (c(5)+pmob(imbz3)/dn(3)) * c(8))
  bel(3) = beta(i3+1)*c(1) + gama(i3+1)*c(2)
  bel(15) = beta(i3+2)*c(1) + gama(i3+2)*c(2)
  bel(27) = beta(i3+3)*c(1) + gama(i3+3)*c(2)

```

c
c— Compute the element rhs vector for the aqueous equation.
c(2) = 4.d0 * (dn(1)*pmob(imbz1) + dn(2)*pmob(imbz2) +

```

+      dn(3)*pmob(imbz3)      ! haz
c(1) = bpermh(mblk) * c(2)    ! hax
fel(1) = (t(21) * (beta(i3+1)*c(1) -
+      (c(4)+pmob(imbx1))*c(7)) +
+      t(22) * (gama(i3+1)*c(2) -
+      (c(3)+pmob(imbz1))*c(8))) * c(11)
fel(2) = (t(21) * (beta(i3+2)*c(1) -
+      (c(4)+pmob(imbx2))*c(7)) +
+      t(22) * (gama(i3+2)*c(2) -
+      (c(3)+pmob(imbz2))*c(8))) * c(11)
fel(3) = (t(21) * (beta(i3+3)*c(1) -
+      (c(4)+pmob(imbx3))*c(7)) +
+      t(22) * (gama(i3+3)*c(2) -
+      (c(3)+pmob(imbz3))*c(8))) * c(11)
c
c— Include mass exchange terms for aqueous phase equation.
If (lctrl(14)) then
  ipt2x6 = ipt(53)
  iex1 = ipt2x6+ndstk1
  iex2 = ipt2x6+ndstk2
  iex3 = ipt2x6+ndstk3
  c(1) = pex(iex1)/dn(1) + pex(iex2)/dn(2) +
+      pex(iex3)/dn(3)
  c(2) = c(10) * t(8)
  fel(1) = fel(1) + c(2) * (c(1) + pex(iex1)/dn(1))
  fel(2) = fel(2) + c(2) * (c(1) + pex(iex2)/dn(2))
  fel(3) = fel(3) + c(2) * (c(1) + pex(iex3)/dn(3))
EndIf
c
c— Include compositional effects on density in the aqueous RHS matrix.
If (lctrl(22)) then
  ipa1 = ipt(1)+nd1
  ipa2 = ipt(1)+nd2
  ipa3 = ipt(1)+nd3
  isa1 = ipt(2)+ndstk1
  isa2 = ipt(2)+ndstk2
  isa3 = ipt(2)+ndstk3
  c(1) = sat(isa1)*dden(ipa1)/dn(1) +
+      sat(isa2)*dden(ipa2)/dn(2) +
+      sat(isa3)*dden(ipa3)/dn(3)
  c(2) = c(10) * t(8)
  fel(1) = fel(1) - c(2)*(c(1)+sat(isa1)*dden(ipa1)/dn(1))
  fel(2) = fel(2) - c(2)*(c(1)+sat(isa2)*dden(ipa2)/dn(2))
  fel(3) = fel(3) - c(2)*(c(1)+sat(isa3)*dden(ipa3)/dn(3))
EndIf
c
c— Compute the element stiffness matrix for the gas equation.
imbx1 = ndstk1
imbx2 = ndstk2
imbx3 = ndstk3
imbz1 = ipt(2)+ndstk1
imbz2 = ipt(2)+ndstk2
imbz3 = ipt(2)+ndstk3
c(3) = pmob(imbz1) + pmob(imbz2) + pmob(imbz3)
c(4) = bpermh(mblk) * c(3) ! c(4)=sum of x mobilities
dn(1) = den(3*ipt(1)+nd1)
dn(2) = den(3*ipt(1)+nd2)
dn(3) = den(3*ipt(1)+nd3)
c(5) = pmob(imbz1)/dn(1) + pmob(imbz2)/dn(2) +
+      pmob(imbz3)/dn(3) ! c(5)=sum of z mobil/density
c(6) = bpermh(mblk) * c(5) ! c(6)=x mobil/density
c(7) = beta(i3+1)*dn(1) + beta(i3+2)*dn(2) +
+      beta(i3+3)*dn(3) ! c(7)=sum of density * beta
c(8) = gama(i3+1)*dn(1) + gama(i3+2)*dn(2) +
+      gama(i3+3)*dn(3) ! c(8)=sum of density * gama
c(1) = c(9) * (c(4)*beta(i3+1) - 0.250d0 *
+      (c(6)+pmob(imbx1)/dn(1)) * c(7))
c(2) = c(9) * (c(3)*gama(i3+1) - 0.250d0 *
+      (c(5)+pmob(imbz1)/dn(1)) * c(8))
bel(10) = beta(i3+1)*c(1) + gama(i3+1)*c(2)
bel(22) = beta(i3+2)*c(1) + gama(i3+2)*c(2)
bel(34) = beta(i3+3)*c(1) + gama(i3+3)*c(2)
c(1) = c(9) * (c(4)*beta(i3+2) - 0.250d0 *
+      (c(6)+pmob(imbx2)/dn(2)) * c(7))
c(2) = c(9) * (c(3)*gama(i3+2) - 0.250d0 *
+      (c(5)+pmob(imbz2)/dn(2)) * c(8))
bel(11) = beta(i3+1)*c(1) + gama(i3+1)*c(2)
bel(23) = beta(i3+2)*c(1) + gama(i3+2)*c(2)

```

```

bel(35) = beta(i3+3)*c(1) + gama(i3+3)*c(2)
c(1) = c(9) * (c(4)*beta(i3+3) - 0.250d0 *
+      (c(6)+pmob(imbx3)/dn(3)) * c(7))
c(2) = c(9) * (c(3)*gama(i3+3) - 0.250d0 *
+      (c(5)+pmob(imbz3)/dn(3)) * c(8))
bel(12) = beta(i3+1)*c(1) + gama(i3+1)*c(2)
bel(24) = beta(i3+2)*c(1) + gama(i3+2)*c(2)
bel(36) = beta(i3+3)*c(1) + gama(i3+3)*c(2)
c
c— Compute the element rhs vector for the gas equation.
c(2) = 4.d0 * (dn(1)*pmob(imbz1) + dn(2)*pmob(imbz2) +
+      dn(3)*pmob(imbz3)) ! hgz
c(1) = bpermh(mblk) * c(2) ! hgx
fel(4) = (t(21) * (beta(i3+1)*c(1) -
+      (c(4)+pmob(imbx1))*c(7)) +
+      t(22) * (gama(i3+1)*c(2) -
+      (c(3)+pmob(imbz1))*c(8))) * c(11)
fel(5) = (t(21) * (beta(i3+2)*c(1) -
+      (c(4)+pmob(imbx2))*c(7)) +
+      t(22) * (gama(i3+2)*c(2) -
+      (c(3)+pmob(imbz2))*c(8))) * c(11)
fel(6) = (t(21) * (beta(i3+3)*c(1) -
+      (c(4)+pmob(imbx3))*c(7)) +
+      t(22) * (gama(i3+3)*c(2) -
+      (c(3)+pmob(imbz3))*c(8))) * c(11)
c
c— Include mass exchange terms for gas phase equation.
If (lctrl(14)) then
  iex1 = 5*ipt(2)+ndstk1
  iex2 = 5*ipt(2)+ndstk2
  iex3 = 5*ipt(2)+ndstk3
  c(1) = pex(iex1)/dn(1) + pex(iex2)/dn(2) +
+      pex(iex3)/dn(3)
  c(2) = c(10) * t(8)
  fel(4) = fel(4) + c(2) * (c(1) + pex(iex1)/dn(1))
  fel(5) = fel(5) + c(2) * (c(1) + pex(iex2)/dn(2))
  fel(6) = fel(6) + c(2) * (c(1) + pex(iex3)/dn(3))
EndIf
c
c— Include compositional effects on density in the gas RHS matrix.
If (lctrl(22)) then
  isg1 = ndstk1
  isg2 = ndstk2
  isg3 = ndstk3
  c(1) = sat(isg1)*p(nd1)*dden(nd1)/(dn(1)*r*temp(nd1)) +
+      sat(isg2)*p(nd2)*dden(nd2)/(dn(2)*r*temp(nd2)) +
+      sat(isg3)*p(nd3)*dden(nd3)/(dn(3)*r*temp(nd3))
  c(2) = c(10) * t(8)
  fel(4) = fel(4) - c(2)*(c(1)+
+      sat(isg1)*p(nd1)*dden(nd1)/(dn(1)*r*temp(nd1)))
  fel(5) = fel(5) - c(2)*(c(1)+
+      sat(isg2)*p(nd2)*dden(nd2)/(dn(2)*r*temp(nd2)))
  fel(6) = fel(6) - c(2)*(c(1)+
+      sat(isg3)*p(nd3)*dden(nd3)/(dn(3)*r*temp(nd3)))
EndIf
c
c— Assemble local matrices in the global matrix for picard iteration.
c— The global 'a' matrix is in banded storage form by rows.
irow1 = 0
do 20 ik = 1,2
do 20 i = 1,3
  irow1 = irow1 + 1
  irowg = (nd(i)-1)*2 + ik
  sum = 0.0d0
do 21 j = 1,3
  jcol = irow1 + (j-1)*12
  sum = sum + bel(jcol) * pt(ipt(1)+nd(j)) +
+      bel(jcol+6) * pt(nd(j))
do 21 ikk = 1,2
  jcol = (nd(j)-1)*2 + ikk
  node = irow1 + ((j-1)*2+ikk-1)*6
  nx = (irowg-1)*nbw(2) + jcol - irowg + nbw(1) + 1
  a(nx) = a(nx) + ael(node) + t(10)*bel(node)
21   rhs(irowg) = rhs(irowg) + fel(irow1) - sum
20   continue
c
c— Adjust rhs vector for nodes with constant volumetric flux.
jj = ipt(18)+ipt(19)+ipt(20)+ipt(21)

```

```

ii = jj+ipt(22)
If (ipt(23) .gt. 0) then ! nodes with constant aqueous flux
  Do i = 1, ipt(23)
    irowg = 2*(ibc(ii+i)-1) + 1
    rhs(irowg) = rhs(irowg) + source(ipt(22)+i)
  EndDo
EndIf
If (ipt(22) .gt. 0) then ! nodes with constant gas flux
  Do i = 1, ipt(22)
    irowg = 2*ibc(jj+i)
    rhs(irowg) = rhs(irowg) + source(i) * patm *
+    temp(ibc(jj+i)) / ((patm + p(ibc(jj+i)))) * tstd
  EndDo
EndIf

```

```

c
c— Adjust rhs vector for flow at nodes along the well screen.
If (lctrl(12)) then
  iptbc = ipt(18)+ipt(19)+ipt(20)+ipt(21)+ipt(22)+ipt(23)
  ii = ipt(22)+ipt(23)
  jj = ii+ipt(24)
  Do i = 1, ipt(24)
    irowg = 2*(ibc(iptbc+i)-1) + 1
    if (qwell.lt.0.0d0) rhs(irowg) = rhs(irowg)+source(jj+i)
    irowg = irowg + 1
    rhs(irowg) = rhs(irowg) + source(ii+i) * patm *
+    temp(ibc(iptbc+i)) / ((patm+
+    p(ibc(iptbc+i)))) * tstd
  EndDo
EndIf

```

```

c
c— Save matrices for mass balance computation.
If (lprnt(6).or.lctrl(2)) then
  Do i = 1, 2*ipt(1)*nbw(2)
    amb(i) = a(i)
  EndDo
  Do i = 1, 2*ipt(1)
    fmb(i) = rhs(i)
  EndDo
EndIf

```

```

c
c— Adjust rows of global matrix for constant pressure conditions
If (ipt(18).gt.0) then ! Constant gas pressure nodes
  Do i = 1, ipt(18)
    ii = (2*ibc(i)-1)*nbw(2)
    Do j = ii+1, ii+nbw(2)
      a(j) = zer0
    EndDo
    a(ii+nbw(1)+1) = 1.0d0
    rhs(2*ibc(i)) = zer0
  EndDo
EndIf
If (ipt(19).gt.0) then ! Constant aqueous pressure nodes
  Do i = 1, ipt(19)
    ii = 2*(ibc(ipt(18)+i)-1)*nbw(2)
    Do j = ii+1, ii+nbw(2)
      a(j) = zer0
    EndDo
    a(ii+nbw(1)+1) = 1.0d0
    rhs(2*ibc(ipt(18)+i)-1) = zer0
  EndDo
EndIf

```

```

c
c— Collapse banded storage of linear system into sparse matrix form
c— used by Harwell. At the same time scale array by dividing rows
c— through by value on the main diagonal.
ia = 0
nx = 0
do 29 irow = 1, 2*ipt(1)
  aii = 1.0d0 / a(irow*nbw(2)-nbw(1))
  rhs(irow) = rhs(irow) * aii
  do 30 j = 1, nbw(2)
    nx = nx + 1
    if (a(nx) .ne. 0.0d0) then
      ia = ia + 1
      a(ia) = a(nx) * aii
      irn(ia) = irow
      icn(ia) = nx - (irow-1)*nbw(2) + irow - nbw(1) - 1
    endif
  enddo
enddo

```

```

30 continue
29 continue
c
c— Solve linear system using Harwell.
call ma28ad (2*ipt(1), ia, a, icnl, irn, irnl, icn, u,
+ ikeep, iw, w, iflag)
if (iflag .lt. 0)
+ write (ipt(28), *) 'flow iflag return from harwell is ', iflag
call ma28cd (2*ipt(1), a, icnl, icn, ikeep, rhs, w, mtype)
c
c— Transfer solutions to the pressure vectors. Calculate the capillary
c— pressure and the maximum relative differences.
pa = zer0
dpa = zer0
pg = zer0
dpg = zer0
Do i = 1, ipt(1)
  pkp1 = pt(ipt(1)+i) + rhs(2*i-1)
  pa = dmax1 (pa, abs(pkp1))
  dpa = dmax1 (dpa, abs(pkp1-p(ipt(1)+i)))
  p(ipt(1)+i) = pkp1
  pkp1 = pt(i) + rhs(2*i) ! gas P at k+1 iteration
  pg = dmax1 (pg, abs(pkp1))
  dpg = dmax1 (dpg, abs(pkp1-p(i)))
  p(i) = pkp1
  p(2*ipt(1)+i) = p(i) - p(ipt(1)+i)
EndDo
c
c— Update saturation and save previous iterate values.
call SATW
Do i = 1, 2*ipt(2)
  satk(i) = sat(i)
EndDo
c
c— Update gas phase mass and molar density as a function of pressure.
n3 = ipt(41)
Do i = 1, ipt(1)
  den(i) = (patm+p(i)) / (r*temp(i))
  den(n3+i) = pmw(i) * den(i)
EndDo
c
c— Check convergence.
if (dpa/pa .le. t(13) .and. dpg/pg .le. t(13)) then
  if (ipt(28).gt.0) write (ipt(28), 200)
+ its, it, lctrl(14), t(9), t(9)/86400, d0, t(8)
  If (it .le. ipt(34)) then ! update time step adj. flag
    ipt(36) = 1
  Else
    ipt(36) = 0
  EndIf
c
c— Compute fluxes for mass balance
if (lprnt(6).or.lctrl(2)) then
  do 800 irow = 2, ipt(40), 2
    if (irow .le. nbw(1)) then
      jstr = nbw(1) + 2 - irow
      jend = nbw(2)
    else if (irow .ge. ipt(40)-nbw(1)) then
      jstr = 1
      jend = nbw(2) - nbw(1) + 2*ipt(1) - irow
    else
      jstr = 1
      jend = nbw(2)
    endif
    sum = 0.0d0
    do 805 j = jstr, jend
      805 sum = sum + amb(nbw(2)*(irow-1)+j)
+ * rhs(irow-nbw(1)-1+j)
    800 fmb(irow) = (sum-fmb(irow))
    do 820 irow = 1, ipt(40)-1, 2
      if (irow .le. nbw(1)) then
        jstr = nbw(1) + 2 - irow
        jend = nbw(2)
      else if (irow .ge. ipt(40)-nbw(1)) then
        jstr = 1
        jend = nbw(2) - nbw(1) + ipt(40) - irow
      else
        jstr = 1

```

```

        jend = nbw(2)
        endif
        sum = 0,d0
        do 825 j = jstrt,jend
825      sum = sum + amb(nbw(2)*(irow-1)+j)
        +      * rhs(irow-nbw(1)-1+j)
820      fmb(irow) = (sum - fmb(irow))
        endif
        return
        endif
c
c— Convergence not achieved. Iterate unless ipt(31) has been reached.
100 continue
c
c— Max iterations exceeded; increment flag for time step reduction.
it = it+1

```

```

ipt(36) = -1
If (ipt(28).gt.0) then
  write(ipt(28),*)>>> Time step reduction in flow it>=',ipt(31)
  write (ipt(28),500) its,t(9),t(8),it
EndIf
c
c— Formats.
200 format ('Flow sol converged at step =',i6,
+         ' iterations =',i3, ' lctrl(14) =',i3/
+         ' Time (s,d) = ',2e12.4,
+         ' time step (s) = ',e10.4)
500 format ('flow sol: t step=',i4, ' time=',e10.4, ' dt=',e10.4,
+ ' its =',i4)
        return
        end

```

Subroutine - grid.f

```

c
c
c GRID.U - Subroutine which generates a union jack based
c subdivision of a rectangular grid.
c On input this subroutine reads the number of rectangular
c spaces in the x-direction (nx) and the z-direction (nz).
c The spacing can be uniform or nonuniform in each
c direction.
c Input logical variables (lx) and (lz), if true,
c indicate if the spacing is uniform in the x and z
c directions respectively.
c If lx or lz are true, then a corresponding uniform
c spacing is input (udelx or udelz).
c If lx or lz are false, then nx and nz values of the
c spacing is input.
c On output, Grid.f gives the number of nodes itp(1), the
c number of elements ipt(0), the node coordinates xnode
c and znode, and the element incidence list nodel.
c
c
c-----
subroutine gridu
include 'dimen.inc'
common /cb1/ matel(nelmx),nodel(nel3),nodept(nnmx),nelpt(nel3),
+      matp(nn6)
common /cb1c/ xnode(nnmx),znode(nnmx),rbar(nelmx),area(nelmx)
common /cb95/ nbdL(nzmax),nbdR(nzmax),nbdT(nxmax),nbdB(nxmax),
+      nnhor,nnver
c
c— Dimension local arrays.
c
dimension delx(nxmax),delz(nzmax),imblk(nzmax)
c
c— read the number of rectangular spaces in the x and z directions
call commnt(11,21)
read(11,*) nx,nz
ipt(86) = nz + 1
ipt(87) = nx + 1
if (nx.le.0.or. nx.gt.nxmax) call ErrorMessage(15,0,ipt(29))
if (nz.le.0.or. nz.gt.nzmax) call ErrorMessage(16,0,ipt(29))
c
c— read horizontal spacing:
call commnt(11,21)
read(11,*) ldel,xzero ! is horizontal spacing uniform
if (ldel) then
  read(11,*) delx(1) ! uniform spacing in x-direction
  do 10 i = 2,nx
10    delx(i) = delx(1)
else
  read(11,*) delx(1)
  if (delx(1) .lt. 0.0d0) then !make unifrom grid linear in R.
    rmult = (-delx(1)/xzero) ** (1.0d0/dble(nx))
    rad = xzero
    do i = 1,nx

```

```

      radp1 = rad * rmult
      delx(i) = radp1 - rad
      rad = radp1
    EndDo
  else
    backspace 11
    read(11,*) (delx(i),i=1,nx) ! nonuniform x spacing
  endif
endif
do i = 1,nx
  If (delx(i) .le. 0.0d0) Call ErrorMessage(17,0,ipt(29))
EndDo
c
c— read vertical spacing
call commnt(11,21)
read(11,*) ldel,zzero ! is vertical spacing uniform
if (ldel) then
  read(11,*) delz(1) ! uniform spacing in z-direction
  do 11 j = 2,nz
11    delz(j) = delz(1)
  else
    read(11,*) (delz(j),j=1,nz) ! nonuniform z spacing
  endif
do j = 1,nz
  If (delz(j) .le. 0.0d0) Call ErrorMessage(18,0,ipt(29))
EndDo
c
c— Calculate the number of elements and nodes.
ipt(0) = 4 * nx * nz ! number of elements
ipt(1) = 2*nx*nz + nx + nz + 1 ! number of nodes
if (ipt(0) .gt. nelmx) Call ErrorMessage(19,0,ipt(29))
if (ipt(1) .gt. nnmx) Call ErrorMessage(20,0,ipt(29))
nnhor = nx + 1
nnver = nz + 1
c
c— Read the number of horizontally aligned material property blocks,
c followed by the material block number for all vertical spacings.
call commnt(11,21)
read(11,*) ipt(26)
If (ipt(26) .lt. 1) Then
  Call ErrorMessage(24,0,ipt(29))
Else If (ipt(26) .eq. 1) Then
  do i = 1,nz
    imblk(i) = 1
  EndDo
Else
  Read(11,*) (imblk(i),i=1,nz)
  do i = 1,nz
    if (imblk(i).lt.1 .or. imblk(i).gt.ipt(26))
+      Call ErrorMessage(24,0,ipt(29))
  EndDo
EndIf
c

```


c— Assign node coordinates. Number nodes along the shortest dimension.

```

if (nx .le. nz) then ! number nodes horizontally
  nd = 1
  z = zzero
  x = xzero
  xnode(nd) = x
  znode(nd) = z
  do 20 i = 1, nx
    nd = nd + 1
    x = x + delx(i)
    xnode(nd) = x
  20   znode(nd) = z
  do 21 j = 1, nz
    nd = nd + 1
    z = z + delz(j) * 0.50d0
    x = xzero + delx(1) * 0.50d0
    xnode(nd) = x
    znode(nd) = z
    if (nx .gt. 1) then
      do 22 i = 1, nx-1
        nd = nd + 1
        x = x + (delx(i)+delx(i+1)) * 0.50d0
        xnode(nd) = x
        znode(nd) = z
      22   endif
      nd = nd + 1
      x = xzero
      z = z + delz(j) * 0.50d0
      xnode(nd) = x
      znode(nd) = z
      do 21 i = 1, nx
        nd = nd + 1
        x = x + delx(i)
        xnode(nd) = x
        znode(nd) = z
      21   else ! number nodes vertically
        nd = 1
        z = zzero
        x = xzero
        xnode(nd) = x
        znode(nd) = z
        do 30 j = 1, nz
          nd = nd + 1
          z = z + delz(j)
          xnode(nd) = x
          znode(nd) = z
        30   do 31 i = 1, nx
          nd = nd + 1
          x = x + delx(i) * 0.50d0
          z = zzero + delz(1) * 0.50d0
          xnode(nd) = x
          znode(nd) = z
          if (nz .gt. 1) then
            do 32 j = 1, nz-1
              nd = nd + 1
              z = z + (delz(j)+delz(j+1)) * 0.50d0
              xnode(nd) = x
              znode(nd) = z
            32   endif
            nd = nd + 1
            x = x + delx(i) * 0.50d0
            z = zzero
            xnode(nd) = x
            znode(nd) = z
            do 31 j = 1, nz
              nd = nd + 1
              z = z + delz(j)
              xnode(nd) = x
              znode(nd) = z
            31   endif
          endif
        c
c— define the incidence lists
jel = 0
i3 = -3
if (nx .le. nz) then ! number elements horizontally
  do 40 j = 1, nz
    do 40 i = 1, nx
      ndul = (j-1)*(2*nx+1) + i

```

```

      ndur = ndul + 1
      ndmd = ndur + nx
      ndfl = ndmd + nx
      ndlr = ndll + 1
      If (i.eq.1) then !identify boundary nodes(left and right)
        nbdL(j) = ndul
        if (j.eq.nz) nbdL(j+1) = ndll
      Else if (i.eq.nx) then
        nbdR(j) = ndur
        if (j.eq.nz) nbdR(j+1) = ndlr
      Endif
      If (j.eq.1) then !identify boundary nodes(top and bottom)
        nbdT(i) = ndul
        if (i.eq.nx) nbdT(i+1) = ndur
      Else if (j.eq.nz) then
        nbdB(i) = ndll
        if (i.eq.nx) nbdB(i+1) = ndlr
      Endif
      jel = jel + 1
      i3 = i3 + 3
      nodel(i3+1) = ndul
      nodel(i3+2) = ndur
      nodel(i3+3) = ndmd
      matel(jel) = imblk(j)
      jel = jel + 1
      i3 = i3 + 3
      nodel(i3+1) = ndur
      nodel(i3+2) = ndlr
      nodel(i3+3) = ndmd
      matel(jel) = imblk(j)
      jel = jel + 1
      i3 = i3 + 3
      nodel(i3+1) = ndmd
      nodel(i3+2) = ndlr
      nodel(i3+3) = ndll
      matel(jel) = imblk(j)
      jel = jel + 1
      i3 = i3 + 3
      nodel(i3+1) = ndul
      nodel(i3+2) = ndmd
      nodel(i3+3) = ndll
      matel(jel) = imblk(j)
      continue
    40   else ! number elements vertically
      do 50 i = 1, nx
        do 50 j = 1, nz
          ndul = (i-1)*(2*nz+1) + j
          ndll = ndul + 1
          ndmd = ndll + nz
          ndur = ndmd + nz
          ndlr = ndur + 1
          If (i.eq.1) then !identify boundary nodes(left and right)
            nbdL(j) = ndul
            if (j.eq.nz) nbdL(j+1) = ndll
          Else if (i.eq.nx) then
            nbdR(j) = ndur
            if (j.eq.nz) nbdR(j+1) = ndlr
          Endif
          If (j.eq.1) then !identify boundary nodes(top and bottom)
            nbdT(i) = ndul
            if (i.eq.nx) nbdT(i+1) = ndur
          Else if (j.eq.nz) then
            nbdB(i) = ndll
            if (i.eq.nx) nbdB(i+1) = ndlr
          Endif
          jel = jel + 1
          i3 = i3 + 3
          nodel(i3+1) = ndul
          nodel(i3+2) = ndur
          nodel(i3+3) = ndmd
          matel(jel) = imblk(j)
          jel = jel + 1
          i3 = i3 + 3
          nodel(i3+1) = ndur
          nodel(i3+2) = ndlr
          nodel(i3+3) = ndmd
          matel(jel) = imblk(j)
          jel = jel + 1

```

```

i3 = i3 + 3
node1(i3+1) = ndmd
node1(i3+2) = ndlr
node1(i3+3) = ndll
matel(jel) = imblk(j)
jel = jel + 1
i3 = i3 + 3
node1(i3+1) = ndul
node1(i3+2) = ndmd
node1(i3+3) = ndll
matel(jel) = imblk(j)
50 continue
endif
return
end
C
C GRID.HB - Subroutine which generates a non-symmetric triangular
C grid system based subdivision of a rectangular grid
C (Similar to the grid used by howard).
C On input this subroutine reads the number of rectangular
C spaces in the x-direction (nx) and the z-direction (nz).
C The spacing can be uniform or nonuniform in each
C direction.
C Input logical variables (lx) and (lz), if true,
C indicate if the spacing is uniform in the x and z
C directions respectively.
C If lx or lz are true, then a corresponding uniform
C spacing is input (udelx or udelz).
C If lx or lz are false, then nx and nz values of the
C spacing is input.
C On output, Grid.f gives the number of nodes ipt(1), the
C number of elements ipt(0), the node coordinates xnode
C and znode, and the element incidence list node1.
C
C
C subroutine gridhb
C include 'dimen.inc'
C common /cb1/ matel(nelmx),node1(nel3),nodept(nnmx),nelpt(nel3),
C + matp(nn6)
C common /cb1c/ xnode(nnmx),znode(nnmx),rbar(nelmx),area(nelmx)
C common /cb95/ nbdL(nzmax),nbdR(nzmax),nbdT(nxmax),nbdB(nxmax),
C + nnhor,nnver
C
C Dimension local arrays.
C
C dimension delx(nxmax),delz(nzmax),imblk(nzmax)
C
C read the number of rectangular spaces in the x and z directions
C call commnt (11,21)
C read (11,*) nx,nz
C ipt(86) = nz + 1
C ipt(87) = nx + 1
C if (nx.le.0 .or. nx.gt.nxmax) call ErrorMessage (15,0,ipt(29))
C if (nz.le.0 .or. nz.gt.nzmax) call ErrorMessage (16,0,ipt(29))
C
C read horizontal spacing:
C call commnt (11,21)
C read (11,*) ldel,xzero ! is horizontal spacing uniform
C if (ldel) then
C read (11,*) delx(1) ! uniform spacing in x-direction
C do 10 i = 2,nx
10 delx(i) = delx(1)
C else
C read (11,*) delx(1)
C if (delx(1) .lt. 0.0d0) then !make uniform grid linear in ln r.
C rmult = (-delx(1)/xzero) ** (1.0d0/dble(nx))
C rad = xzero
C do i = 1,nx
C radp1 = rad * rmult
C delx(i) = radp1 - rad
C rad = radp1
C EndDo
C else
C backspace 11
C read (11,*) (delx(i),i=1,nx) ! nonuniform x spacing
C endif
C endif
C do i = 1,nx

```

```

C if (delx(i) .le. 0.0d0) Call ErrorMessage (17,0,ipt(29))
C EndDo
C
C read vertical spacing
C call commnt (11,21)
C read (11,*) ldel,zzero ! is vertical spacing uniform
C if (ldel) then
C read (11,*) delz(1) ! uniform spacing in z-direction
C do 11 j = 1,nz
11 delz(j) = delz(1)
C else
C read (11,*) (delz(j),j=1,nz) ! nonuniform z spacing in
C endif
C do j = 1,nz
C if (delz(j) .le. 0.0d0) Call ErrorMessage (18,0,ipt(29))
C EndDo
C
C Calculate the number of elements and nodes.
C ipt(0) = 2 * nx * nz ! number of elements
C ipt(1) = (nx+1) * (nz+1) ! number of nodes
C if (ipt(0) .gt. nelmx) Call ErrorMessage (19,0,ipt(29))
C if (ipt(1) .gt. nnmx) Call ErrorMessage (20,0,ipt(29))
C nnhor = nx + 1
C nnver = nz + 1
C
C Read the number of horizontally aligned material property blocks,
C followed by the material block number for all vertical spacings.
C call commnt (11,21)
C read (11,*) ipt(26)
C if (ipt(26) .lt. 1) Then
C Call ErrorMessage (24,0,ipt(29))
C Else if (ipt(26) .eq. 1) Then
C do i = 1,nz
C imblk(i) = 1
C EndDo
C Else
C Read (11,*) (imblk(i),i=1,nz)
C do i = 1,nz
C if (imblk(i) .lt. 1 .or. imblk(i) .gt. ipt(26))
C + Call ErrorMessage (24,0,ipt(29))
C EndDo
C EndIf
C
C Assign node coordinates. Number nodes along the shortest
C dimension.
C if (nx .le. nz) then ! number nodes horizontally
C nd = 1
C z = zzero
C x = xzero
C xnode(nd) = x
C znode(nd) = z
C do 20 i = 1,nx
C nd = nd + 1
C x = x + delx(i)
C xnode(nd) = x
C znode(nd) = z
20 do 21 j = 1,nz
C nd = nd + 1
C x = xzero
C z = z + delz(j)
C xnode(nd) = x
C znode(nd) = z
C do 21 i = 1,nx
C nd = nd + 1
C x = x + delx(i)
C xnode(nd) = x
C znode(nd) = z
21 else ! number nodes vertically
C nd = 1
C z = zzero
C x = xzero
C xnode(nd) = x
C znode(nd) = z
C do 30 j = 1,nz
C nd = nd + 1
C z = z + delz(j)
C xnode(nd) = x
C znode(nd) = z
30

```

```

do 31 i = 1,nx
  nd = nd + 1
  x = x + delx(i)
  z = zzero
  xnode(nd) = x
  znode(nd) = z
do 31 j = 1,nz
  nd = nd + 1
  z = z + delz(j)
  xnode(nd) = x
  znode(nd) = z
31 endif
c
c--- define the incidence lists
jel = 0
i3 = -3
if (nx .le. nz) then ! number elements horizontally
do 40 j = 1,nz
  if (2*(j/2) .eq. j) then
    idir = 1
  else
    idir = -1
  endif
do 40 i = 1,nx
  ndul = (j-1)*(nx+1) + i
  ndur = ndul + 1
  ndll = ndul + nx + 1
  ndlr = ndll + 1
  If (i.eq.1) then !identify boundary nodes(left and right)
    nbdL(j) = ndul
    if (j.eq.nz) nbdL(j+1) = ndll
  Else if (i.eq.nx) then
    nbdR(j) = ndur
    if (j.eq.nz) nbdR(j+1) = ndlr
  Endif
  If (j.eq.1) then !identify boundary nodes(top and bottom)
    nbdT(i) = ndul
    if (i.eq.nx) nbdT(i+1) = ndur
  Else if (j.eq.nz) then
    nbdB(i) = ndll
    if (i.eq.nx) nbdB(i+1) = ndlr
  Endif
  if (idir .lt. 0) then
    jel = jel + 1
    i3 = i3 + 3
    nodel(i3+1) = ndul
    nodel(i3+2) = ndlr
    nodel(i3+3) = ndll
    matel(jel) = imblk(j)
    jel = jel + 1
    i3 = i3 + 3
    nodel(i3+1) = ndur
    nodel(i3+2) = ndur
    nodel(i3+3) = ndll
    matel(jel) = imblk(j)
    idir = -idir
  else if (idir .gt. 0) then
    jel = jel + 1
    i3 = i3 + 3
    nodel(i3+1) = ndul
    nodel(i3+2) = ndur
    nodel(i3+3) = ndll
    matel(jel) = imblk(j)
    jel = jel + 1
    i3 = i3 + 3
    nodel(i3+1) = ndur
  endif
end
end
end

```

```

nodel(i3+2) = ndlr
nodel(i3+3) = ndll
matel(jel) = imblk(j)
idir = -idir
endif
40 continue
else ! number elements vertically
do 50 i = 1,nx
  if (2*(i/2) .eq. i) then
    idir = 1
  else
    idir = -1
  endif
do 50 j = 1,nz
  ndul = (i-1)*(nz+1) + j
  ndll = ndul + 1
  ndur = ndul + nz + 1
  ndlr = ndur + 1
  If (i.eq.1) then !identify boundary nodes(left and right)
    nbdL(j) = ndul
    if (j.eq.nz) nbdL(j+1) = ndll
  Else if (i.eq.nx) then
    nbdR(j) = ndur
    if (j.eq.nz) nbdR(j+1) = ndlr
  Endif
  If (j.eq.1) then !identify boundary nodes(top and bottom)
    nbdT(i) = ndul
    if (i.eq.nx) nbdT(i+1) = ndur
  Else if (j.eq.nz) then
    nbdB(i) = ndll
    if (i.eq.nx) nbdB(i+1) = ndlr
  Endif
  if (idir .lt. 0) then
    jel = jel + 1
    i3 = i3 + 3
    nodel(i3+1) = ndul
    nodel(i3+2) = ndur
    nodel(i3+3) = ndlr
    matel(jel) = imblk(j)
    jel = jel + 1
    i3 = i3 + 3
    nodel(i3+1) = ndul
    nodel(i3+2) = ndlr
    nodel(i3+3) = ndll
    matel(jel) = imblk(j)
    idir = -idir
  else if (idir .gt. 0) then
    jel = jel + 1
    i3 = i3 + 3
    nodel(i3+1) = ndur
    nodel(i3+2) = ndur
    nodel(i3+3) = ndll
    matel(jel) = imblk(j)
    jel = jel + 1
    i3 = i3 + 3
    nodel(i3+1) = ndur
    nodel(i3+2) = ndlr
    nodel(i3+3) = ndll
    matel(jel) = imblk(j)
    idir = -idir
  endif
50 continue
endif
return
end

```

Subroutine - input1.f

```

C-----
C INPUT1.f - main input routine. Reads model control information,
C fluid and soil parameters, and grid information.
C
C Control Flags computed internally in routine:
C
C lctrl(13) - logical variable denoting compositional dependence
C of the gas phase viscosity
C lctrl(13) = .true. - gas phase viscosity is
C dependent on composition
C lctrl(13) = .false. - gas phase viscosity is not
C dependent on composition
C-----
subroutine INPUT1
include 'dimen.inc'
character*20 infile(4),outpre,outfile(8+ncmp)
character*10 cname(ncmp)
character*3 cmb(10)
common /cb1/ matel(nelmx),node1(nel3),nodept(nnmxx),nelpt(nel3),
+ matpt(nn6)
common /cb1c/ xnode(nnmxx),znode(nnmxx),rbar(nelmx),area(nelmx)
common /cb1d/ gama(nel3),beta(nel3)
common /cb5a/ bphi(nmbk),bpermh(nmbk),bpermv(nmbk)
common /cb5b/ bvgn(nmbk),bvga(nmbk),bvgn(nmbk),bsrv(nmbk)
common /cb5c/ bfoc(nmbk)
common /cb6c/ temp(nnmxx)
common /cb6d/ dtemp(nzmax6),idepth(nnmxx)
common /cb7/ cvvis(ncmp),gamma(ncsqd)
common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
+ chen(ncmp),casol(ncmp),cmdif(ncmp2)
common /cb8/ vis(nnmxx),pmw(nn3)
common /cb41b/ nbw(0:2),ia
common /cb60/ khal(ncmp),fuse(ncmp2),umax(ncmp),xyield(ncmp),
+ kinhib(ncmp)
common /cb63/ kex(ncmp5),kmax(ncmp5)
common /cb64/ bok(nbcmp),bom(nbcmp),krtcd(ncmp)
common /cb64b/ bsden(nmbk)
common /cb70/ d(nmd),tor(nelmx),bdist(nmbk),bdisl(nmbk)
common /cb90/ infile,outpre,outfile
common /cb91/ cname
common /cb95/ nbdL(nzmax),nbdR(nzmax),nbdT(nxmax),nbdB(nxmax),
+ nnhor,nnver
C
C----- Dimension local arrays.
C
C dimension mp(4),zdepth(nzmax)
C
C===== READ INPUT/OUTPUT FILES AND OPTIONS =====
C
C----- Read the name of input file 2 (ICs and BCs) and open as unit 13;
C----- define the error and warning message input data file as unit 14.
call commnt(11,21)
read(11,*) infile(2),infile(3)
open(13,file=infile(2),status='unknown')
open(14,file=infile(3),status='unknown')
C
C----- Read the prefix name of all output files; determine output file
C----- names; and open the main output file.
call commnt(11,21)
read(11,*) outpre
ii = index(outpre,' ') - 1
outfile(1) = outpre(1:ii) // '.out' ! main output file
outfile(2) = outpre(1:ii) // '.err' ! error messages
outfile(3) = outpre(1:ii) // '.cnv' ! performance output
outfile(4) = outpre(1:ii) // '.con' ! contour data
outfile(5) = outpre(1:ii) // '.mb' ! mass balance checks
outfile(6) = outpre(1:ii) // '.plt' ! time series plot data
outfile(7) = outpre(1:ii) // '.rst' ! restart file
open(21,file=outfile(1),status='unknown')
C
C----- Read the device specifications for error messages and
C----- performance output.
C----- 0 is no output; 6 is the screen;
C----- 21 is the .out file; 22 is the .err file (only for error messages)
C----- , 23 is the .cnv file (only for performance output).
call commnt(11,21)

```

```

read(11,*) ipt(29), ipt(28)
C
C----- Open outfile(2) if ipt(29) = 22. Error message/runtime information.
if (ipt(29) .eq. 22) then
open(22,file=outfile(2),status='unknown')
else if (ipt(29) .eq. 6) then
outfile(2) = 'To Screen'
else if (ipt(29) .eq. 0) then
outfile(2) = 'None Opened'
else if (ipt(29) .eq. 21) then
outfile(2) = outfile(1)
else
call ErrorMessage(62,0,6)
endif
C
C----- Open outfile(3) if ipt(28) = 23. Performance and iteration output.
if (ipt(28) .eq. 23) then
open(23,file=outfile(3),status='unknown')
else if (ipt(28) .eq. 6) then
outfile(3) = 'To Screen'
else if (ipt(28) .eq. 0) then
outfile(3) = 'None Opened'
else if (ipt(28) .eq. 21) then
outfile(3) = outfile(1)
else
call ErrorMessage(61,0,ipt(29))
endif
C
C----- Read a logical parameter indicating if contour plot data should be
C----- printed to the file 'outpre.con'. Open the file, if yes.
call commnt(11,21)
read(11,*) lctrl(23)
if (lctrl(23)) then
open(24,file=outfile(4),status='unknown')
else
outfile(4) = 'None Opened'
endif
C
C----- Read A logical parameter indicating if mass balance checks
C----- should be computed and printed to the file 'outpre.mb'. Open the
C----- file, if yes.
call commnt(11,21)
read(11,*) lprnt(6)
if (lprnt(6)) then
open(25,file=outfile(5),status='unknown')
else
outfile(5) = 'No Mass Balance'
endif
C
C----- Read the uniform material balance print interval, either
C----- by the number of time steps or a specified time increment. Also
C----- read a logical variable controlling the type of material balance
C----- output (report style if true).
if (lprnt(6)) then
read(11,*) lprnt(25),lprnt(27)
if (lprnt(25)) then
read(11,*) ipt(83)
else
read(11,*) t(27)
endif
endif
C
C----- Read a logical parameter indicating if time series plot data
C----- should be printed to the file 'outpre.plt'. Open the file, if yes.
call commnt(11,21)
read(11,*) lctrl(15)
if (lctrl(15)) then
open(26,file=outfile(6),status='unknown')
else
outfile(6) = 'None Opened'
endif
C
C----- Read the uniform time series print interval, either by
C----- the number of time steps or a specified time increment. Also read
C----- a logical variable controlling the form in which the component
C----- information is outputted (t=mole fraction, f=concentration).
if (lctrl(15)) then
read(11,*) lprnt(26),lprnt(28)

```

```

    if(lprnt(26)) then
      read(11,*) ipt(84)
    else
      read(11,*) t(28)
    end if
  end if
c
c— Read a logical parameter indicating if restart data
c— should be printed to the file 'outpre.rst'. Open the file.
call commnt(11,21)
read (11,*) lctrl(5)
if (lctrl(5)) then
  open (27,file=outfile(7),status='unknown')
else
  outfile(7) = 'No Restart Available'
endif
c
c— Write to the main output file: banner, title cards, I/O file names.
write (21,500)
call commnt (11,21)
backspace 11
write (21,501) (infile(i),i=1,3),(outfile(i),i=1,7)
c
c— Read and write the uniform print interval, either by the number of
c— time steps or a specified time increment.
call commnt(11,21)
read (11,*) lprnt(0)
if (lprnt(0)) then
  read (11,*) ipt(25)
  write (21,529) ipt(25)
else
  read (11,*) t(12)
  write (21,530) t(12)
endif
c
c— Write the uniform material balance print interval and the type of
c— material balance.
if(lprnt(6)) then
  if(lprnt(27)) then
    write(21,553)
  else
    write(21,554)
  end if
  if(lprnt(25)) then
    write(21,549) ipt(83)
  else
    write(21,550) t(27)
  end if
end if
c
c— Write the uniform time series print interval and the form in which
c— the component information is outputted (t=mole fraction,
c— f=concentration).
if(lctrl(15)) then
  if(lprnt(28)) then
    write(21,557)
  else
    write(21,558)
  end if
  if(lprnt(26)) then
    write(21,551) ipt(84)
  else
    write(21,552) t(28)
  end if
end if
c
c===== BLOCK B - GENERAL MODEL CONTROL OPTIONS =====
c
  if (ipt(29).ne.0) write (ipt(29),*)
  +   'Reading Model Control information'
c
c— Read the coordinate system: 0=xz; 1=rz.
c— Read the horizontal and vertical gravity components.
c— Error check: ipt(27) (0-1)
call commnt(11,21)
read (11,*) ipt(27),t(21),t(22)
if (ipt(27).lt.0 .or. ipt(27).gt.1) call ErrMessage (1,0,ipt(29))
c

```

```

c— Read a logical variables defining which balance equations are
c— solved:
c— lctrl(1) => solve flow equations
c— lctrl(2) => solve transport equations
c— lctrl(24) => solve napl equations
c— lctrl(25) => solve solid phase equations
c— lctrl(3) => solve biophase equations
  call commnt(11,21)
  read (11,*) lctrl(1),lctrl(2),lctrl(24),lctrl(25),lctrl(3)
  if(lctrl(2)) then
    lctrl(22) = .true.
  else
    lctrl(22) = .false.
  end if
c
c— Error Check: lctrl(24),lctrl(25),lctrl(3)=true only if lctrl(2)
c— = true.
  if (lctrl(24) .and. .not. lctrl(2)) Call ErrMessage (76,0,ipt(29))
  if (lctrl(25) .and. .not. lctrl(2)) Call ErrMessage (75,0,ipt(29))
  if (lctrl(3) .and. .not. lctrl(2)) Call ErrMessage (42,0,ipt(29))
c
c— Read mass lumping options in the solution of the flow eqs
c— (lctrl(7)) and the transport eqs (lctrl(8)).
  call commnt(11,21)
  read (11,*) lctrl(7),lctrl(8)
c
c— Read an integer parameter denoting the number of time steps to be
c— skipped between solving for the flow equations.
  call commnt(11,21)
  read (11,*) ipt(85)
c
c— Read in coupling term between flow and transport. lctrl(14)
c— indicates if mass exchange terms are to be included in the
c— solution of the flow eqs.
  call commnt(11,21)
  read (11,*) lctrl(14)
  if (lctrl(1) .and. .not. lctrl(2)) lctrl(14)=.false.
c
c— Lelnum indicates if element dimensionless numbers are to be
c— calculated for the transport solution and written to ipt(28).
  call commnt(11,21)
  read (11,*) lctrl(4)
c
c— Write the general model control information.
  write (21,502) ipt(27),t(21),t(22),lctrl(1),ipt(85),lctrl(2)
  +   ,lctrl(24),lctrl(25),lctrl(3),lctrl(7),lctrl(8)
  +   ,lctrl(14),lctrl(4)
c
c===== TIME STEP/ITERATION CONTROL OPTIONS =====
c
  if (ipt(29).ne.0) write (ipt(29),*)
  +   'Reading Time Step/Iteration Control information'
c
c— Read the initial and final simulation time (sec).
c— Error check: t(2)>t(1); t(1) and t(2) .ge. 0.
  call commnt(11,21)
  read (11,*) t(1),t(2)
  if (t(1).lt.zero) call ErrMessage (2,0,ipt(29))
  if (t(1).ge.t(2)) call ErrMessage (3,0,ipt(29))
c
c— Read the time weighting factor.
c— Error check: 0.ge.weight.le.1
  call commnt(11,21)
  read (11,*) t(10)
  if (t(10).lt.zero.or.t(10).gt.rone)
  +   call ErrMessage (13,0,ipt(29))
c
c— Read the maximum number of iterations.
c— Error check:ipt(30) .gt. 0.
  call commnt(11,21)
  read (11,*) ipt(30)
  if (ipt(30).le.0) call ErrMessage (4,0,ipt(29))
c
c— Read the convergence tolerance for the flow, transport, NAPL
c— saturation and immobile transport eqs.
c— Error check: tol>0.
  call commnt(11,21)
  read (11,*) t(13),t(14),t(15),t(16)

```

```

if ((t(13).lt.zer0 .or. t(14).lt.zer0 .or. t(15).lt.zer0
+ .or. t(16).lt.zer0) call ErrMessage (9,0,ipt(29))
c
c— Read the initial, minimum and maximum time step size (s).
c— Error check: dtmin<dtmax; dtmin>0; dtmin.le.dinitial.le.dtmax
call commnt(11,21)
read (11,*) t(3),t(4),t(5)
if (t(4).le.zer0) call ErrMessage (5,0,ipt(29))
if (t(4).gt.t(5)) call ErrMessage (6,0,ipt(29))
if (t(3).lt.t(4) .or. t(3).gt.t(5)) call ErrMessage (7,0,ipt(29))
c
c— Read the maximum number of iterations for the flow, transport, NAPL
c— saturation, and immobile eqs.
call commnt(11,21)
read (11,*) ipt(31),ipt(32),ipt(33)
c
c— Read the minimum number of iterations for the flow, and transport
c— eqs. Error check: itmin<itmax.
call commnt(11,21)
read (11,*) ipt(34),ipt(35)
if (ipt(34).gt.ipt(31) .or. ipt(35).gt.ipt(32) .or.
+ ipt(35).gt.ipt(33)) call ErrMessage (8,0,ipt(29))
c
c— Read the empirical time step amplification and reduction factors.
c— Error check: t(6).ge.1; t(7).le.1.
call commnt(11,21)
read (11,*) t(6),t(7)
if (t(6).lt.rone) call ErrMessage (10,0,ipt(29))
if (t(7).gt.rone) call ErrMessage (11,0,ipt(29))
c
c— Write the time step control information.
write (21,503) t(1),t(2),t(10),ipt(30),t(13),t(14),t(15),t(16),
+ t(3),t(4),t(5),ipt(31),ipt(32),ipt(33),
+ ipt(34),ipt(35),t(6),t(7)
c
c— Initialize the time step.
t(8) = t(3)
c
c===== GRID PARAMETERS AND OPTIONS =====
c
if (ipt(29).ne.0) write (ipt(29),*)
+ 'Reading Grid Parameters and Options'
c
c— Read a logical variable indicating if grid info should be printed
c— to the main output file.
call commnt(11,21)
read (11,*) lprnt(1)
c
c— Generate the grid?
c 0 = don't generate the grid;
c 1 = generate a union jack grid;
c 2 = generate a herring bone grid.
c— Error check: igrid (0-2)
c— Error check: number of nodes and elements.
Call commnt(11,21)
Read (11,*) igrid
If (igrd.lt.0 .or. igrd.gt.2) call ErrMessage (14,0,ipt(29))
If (igrd .eq. 1) Then
if (ipt(29).ne.0) Write (ipt(29),*)
+ 'Generating a uniform union jack grid'
Call gridu
Else If (igrd .eq. 2) Then
if (ipt(29).ne.0) Write (ipt(29),*)
+ 'Generating a uniform herring bone grid'
Call gridhb
EndIf
c
c— Input the grid.
If (igrd .eq. 0) Then
c
c— Read the number of elements, number of nodes, and the number of
c— material property blocks.
c— Error check: number of nodes, elements, material property blocks.
Call commnt(11,21)
Read (11,*) ipt(0),ipt(1),ipt(26)
write (6,*) ipt(0),ipt(1),ipt(26)
if (ipt(0) .gt. nelmx) Call ErrMessage (19,0,ipt(29))
if (ipt(1) .gt. nmx) Call ErrMessage (20,0,ipt(29))

```

```

if (ipt(26) .gt. nmbk) Call ErrMessage (21,0,ipt(29))
C
C— Read the nodal incidence list and material property block
C— for each element. The element node incidence list consists of
C— the arbitrary global element number followed by that element's
C— three global node numbers. Each element has its own line. The
C— element node numbers start at an arbitrary node. If the z
C— coordinate is positive downwards proceed in the clockwise
C— direction, otherwise proceed in the counterclockwise direction.
C— If there is only one material property block for the entire
C— domain, the material property input assignment for each element
C— may be omitted. The minimum material property block is a two
c— element quadrilateral.
c— Error check: node and material block numbers are within the
c— defined range.
call commnt(11,21)
if (ipt(26).eq.1) then
do 100 i = 1, ipt(0)
read (11,*) ii, nodel(ii*3-2), nodel(ii*3-1), nodel(ii*3)
100 matel(ii) = 1
else
do 110 i = 1, ipt(0)
110 read (11,*) ii, nodel(ii*3-2), nodel(ii*3-1), nodel(ii*3)
+ , matel(ii)
endif
Do 120 i = 1, ipt(0)
i3 = 3*i
If (matel(i).lt.1 .or. matel(i).gt.ipt(26))
+ Call ErrMessage (24,0,ipt(29))
+ If (nodel(i3-2).lt.1 .or. nodel(i3-2).gt.ipt(1) .or.
+ nodel(i3-1).lt.1 .or. nodel(i3-1).gt.ipt(1) .or.
+ nodel(i3).lt.1 .or. nodel(i3).gt.ipt(1))
+ Call ErrMessage (25,0,ipt(29))
120 continue
c
c— Read the nodal coordinates.
call commnt(11,21)
do 130 i = 1, ipt(1)
130 read (11,*) j, xnode(j), znode(j)
EndIf
C
C— Compute the pointers for the nodal storage vectors. Multiple
C— entries are needed for each node whose contiguous elements have
C— different material properties. Pressure is always continuous,
C— while as an example saturation is not when adjacent elements
C— have different saturation/pressure relationships. The number
C— of entries at a given node is equal to the number of different
C— contiguous material property sets. This routine is configured to
C— allow a maximum of four different material property sets to be
C— contiguous at a given node.
C
if (ipt(29).ne.0) write (ipt(29),*)
+ 'Computing pointers for stacked storage'
do 140 i=1, ipt(1)+1
nodept(i)=i
140 continue
do 150 i=1, 4*ipt(1)
matpt(i)=0
150 continue
do 160 i=1, 3*ipt(0)
nelpt(i)=0
160 continue
C
C— First, determine the number of different material property
C— sets that are contiguous at each node. Search the entire
C— node incidence list node by node, storing the material property
C— identifiers in the local array mp.
C
do 170 i=1, ipt(1)
C
C— Initialize local counters to zero before each element search.
C
i4max=0
do 175 j=1, 4
mp(j)=0
175 continue
do 180 ii=1, ipt(0)
do 180 i3=2, 0, -1

```

```

C
C— If the node incidence list for element ii contains node i, check
C— if matel(ii) is contained in one of the four entries of mp for
C— node i.
C
      if(nodel(3*ii-i3).eq.i) then
        do 185 i4=1,4
C
C— If matel(ii) is already an entry in mp, create the corresponding
C— entry in nelpt and go to the next element.
C
          if(matel(ii).eq.mp(i4)) then
            nelpt(3*ii-i3)=i4-1
            go to 180
C
C— Store a new value of matel(ii) in the first zero entry of mp.
C— Keep track of the number of nonzero entries in mp for node i in
C— in isum. Create the corresponding entry in nelpt and continue on
C— to the next element.
C
            else if(mp(i4).eq.0) then
              mp(i4)=matel(ii)
              nelpt(3*ii-i3)=i4-1
              if(i4.gt.i4max+1) i4max=i4-1
              go to 180
            end if
C
C— If all entries of mp for node i are nonzero and matel(ii) is
C— not contained in mp, too many contiguous material properties
C— have been defined. Write an error message and stop.
C
            if(i4.eq.4) then
              write(ipt(29),*) 'ATTENTION: EXCESS MATERIAL',
                'PROPERTY at Node Number', i
              Call ErrMessage (22,0,ipt(29))
            end if
185      continue
        end if
180      continue
C
C— Use i4max and mp to construct nodept and matpt. The value of
C— nodept for node i+1 is the value of nodept for node i plus the
C— number of stacked entries for node i (i4max). Matpt is the nonzero
C— values of mp in order. nodept has dimension ipt(1)+1 to allow
C— for determination of the number of stacked variables at the last
C— global node.
      nodept(i+1) = nodept(i)+i4max+1
      do 170 i5=0,i4max
        matpt(nodept(i)+i5)=mp(i5+1)
170      continue
C
C— Define the dimension of stacked nodal storage.
C— Error check: number of nodal variables in stacked storage.
      ipt(2) = nodept(ipt(1)+1)-1
      if (ipt(2) .gt. nnstk) Call ErrMessage (23,0,ipt(29))
C
C— Determine the full bandwidth of the global matrix; one each for
C— the transport solution and the flow solution.
      nbw(0) = 0
      do 190 i = 1,ipt(0)
        i3 = (i-1)*3
190      nbw(0) = max(nbw(0), iabs(nodel(i3+2)-nodel(i3+1)),
          + iabs(nodel(i3+3)-nodel(i3+2)),
          + iabs(nodel(i3+3)-nodel(i3+1)))
        nbw(1) = 2 * nbw(0) + 1
        nbw(2) = 2 * nbw(1) + 1
C
C— Compute elemental areas
      call ATRI
C
C— Compute beta and gamma coefficients for computation of integration
C— Also compute radial centroid of each element for axisymmetric
C— coordinates. This is set to one if the xz coordinates are used.
      do 200 i = 1,ipt(0)
        i3 = 3*(i-1)
        beta(i3+1) = znodel(nodel(i3+2)) - znodel(nodel(i3+3))
        beta(i3+2) = znodel(nodel(i3+3)) - znodel(nodel(i3+1))
        beta(i3+3) = znodel(nodel(i3+1)) - znodel(nodel(i3+2))

```

```

      gama(i3+1) = xnode(nodel(i3+3)) - xnode(nodel(i3+2))
      gama(i3+2) = xnode(nodel(i3+1)) - xnode(nodel(i3+3))
      gama(i3+3) = xnode(nodel(i3+2)) - xnode(nodel(i3+1))
200      continue
C
C— Output basic grid information.
      write (21,504) igrd,nelmx,ipt(0),nnmx,ipt(1),nmbk,ipt(26),nnstk,
        + ipt(2),nbw(1),nbw(2)
C
C— Output the boundary nodes for the case when the grid is generated.
      If (igrd.ne.0) then
        write (21,559)
        write (21,560) 'Left boundary', (nbdL(i),i=1,nnver)
        write (21,560) 'Right boundary', (nbdR(i),i=1,nnver)
        write (21,560) 'Top boundary', (nbdT(i),i=1,nnhor)
        write (21,560) 'Bottom boundary', (nbdB(i),i=1,nnhor)
      Endif
C
C— Output nodal coordinates, and incidence list.
      If (lprnt(1)) Then
        If (ipt(27) .eq. 0) Then
          Write (21,505)
        Else
          Write (21,506)
        Endif
        Do 210 i = 1,ipt(1)
210      Write (21,507) i,xnode(i),znodel(i)
          Write (21,508)
          Do 220 i = 1,ipt(0)
            i3 = 3*i
220      Write (21,509) i,nodel(i3-2),nodel(i3-1),nodel(i3),
              + matel(i),area(i)
          Endif
C
C— Generate grid based pointers.
      ipt(40) = ipt(1) * 2
      ipt(41) = ipt(1) * 3
      ipt(42) = ipt(1) * 4
      ipt(43) = ipt(1) * 5
      ipt(44) = ipt(1) * 6
      ipt(45) = ipt(1) * 7
      ipt(46) = ipt(1) * 8
      ipt(47) = ipt(1) * 9
      ipt(48) = ipt(1) * 10
      ipt(49) = ipt(2) * 2
      ipt(50) = ipt(2) * 3
      ipt(51) = ipt(2) * 4
      ipt(52) = ipt(2) * 5
      ipt(53) = ipt(2) * 6
      ipt(54) = ipt(2) * 7
      ipt(55) = ipt(2) * 8
      ipt(56) = ipt(2) * 9
      ipt(57) = ipt(2) * 10
      ipt(67) = ipt(0) * 2
      ipt(68) = ipt(0) * 3
C
C===== COMPONENT CHEMICAL PROPERTIES =====
C
      if (ipt(29).ne.0) write (ipt(29),*)
        + 'Reading Component Chemical Properties'
C
C— Read the number of organic components.
C— Error check: ipt(15) must be nonnegative.
C— Error check: If NAPL (lctrl(24)=t), sorption (lctrl(25)=t), or
C— biodegradation (lctrl(3)=t) is considered ipt(15)>0,
      call commt (11,21)
      read (11,*) ipt(15)
      if (ipt(15) .lt. 0) Call ErrMessage (26,0,ipt(29))
      if (ipt(15) .gt. ncmp) Call ErrMessage (43,0,ipt(29))
      if (ipt(15) .eq. 0 .and. lctrl(24)) Call ErrMessage (80,0,ipt(29))
      if (ipt(15) .eq. 0 .and. lctrl(25)) Call ErrMessage (87,0,ipt(29))
      if (ipt(15) .eq. 0 .and. lctrl(3)) Call ErrMessage (88,0,ipt(29))
C
C— Read the organic component chemical properties.
C— Input units assumed: molecular weight (g/mole)
C— vapor pressure = atm
C— vapor viscosity = centipoise
C— liquid density = g/l

```

```

c-- vapor diffusivity = cm^2/s
c-- liquid diffusivity = cm^2/s
c-- henry's constant = atm l/g
c-- aqueous solubility = g/l
c-- Error check: values must be nonnegative except vapor pressure
c-- and solubility.
c-- call commnt(11,21)
  if (ipt(15).gt.0) Then
    Do 230 i = 1,ipt(15)
      read (11,*) ic,cname(ic),cmw(ic),cvp(ic),cvvis(ic)
      + ,cden(ic),cmdif(2*ic-1),cmdif(2*ic),chen(ic)
      + ,casol(ic)
      if (ic.ne.i) Call ErrorMessage(44,0,ipt(29))
      if (cmw(ic).lt.zer0) Call ErrorMessage(89,0,ipt(29))
      if (cvvis(ic).lt.zer0) Call ErrorMessage(90,0,ipt(29))
      if (cden(ic).lt.zer0) Call ErrorMessage(91,0,ipt(29))
      if (cmdif(2*ic-1).lt.zer0) Call ErrorMessage(92,0,ipt(29))
      if (cmdif(2*ic).lt.zer0) Call ErrorMessage(92,0,ipt(29))
      if (chen(ic).lt.zer0) Call ErrorMessage(93,0,ipt(29))
230 continue
c
c-- If lprnt(27) is false, open an additional material balance output
c-- file for each component.
  data cmb(1),cmb(2),cmb(3),cmb(4),cmb(5),cmb(6),cmb(7),cmb(8)
  + ,cmb(9),cmb(10),'1','2','3','4','5','6','7','8','9','10'/
  if(ncmp.gt.10) Call ErrorMessage(126,0,ipt(29))
  call commnt(11,21)
  if(lprnt(6).and.not.lprnt(27)) then
    ii = index(outpre,' ')-1
    do 225 i = 1, ipt(15)
      iii = index(cmb(i),' ')-1
      outfile(8+i) = outpre(1:ii)//'.mb'//cmb(i)(1:iii)
225 open (28+i,file=outfile(8+i),status='unknown')
      write(21,555)
      write(21,556) (cname(i),28+i,outfile(8+i),i=1,ipt(15))
    end if
  Endif
c
c-- Read chemical property data for water, oxygen, and nitrogen
c-- Error check: values must be nonnegative except vapor pressure
c-- and solubility.
  if (ipt(15)+3.gt.ncmp) Call ErrorMessage(43,0,ipt(29))
  call commnt(11,21)
  Do 240 i = ipt(15)+1,ipt(15)+3
    read (11,*) ic,cname(i),cmw(i),cvp(i),cvvis(i),cden(i),
    + ,cmdif(2*i-1),cmdif(2*i),chen(i),casol(i)
    if (ic.ne.i) Call ErrorMessage(44,0,ipt(29))
    if (cmw(i).lt.zer0) Call ErrorMessage(89,0,ipt(29))
    if (cvvis(i).lt.zer0) Call ErrorMessage(90,0,ipt(29))
    if (cden(i).lt.zer0) Call ErrorMessage(91,0,ipt(29))
    if (cmdif(2*ic-1).lt.zer0) Call ErrorMessage(92,0,ipt(29))
    if (cmdif(2*ic).lt.zer0) Call ErrorMessage(92,0,ipt(29))
    if (chen(i).lt.zer0) Call ErrorMessage(93,0,ipt(29))
240 continue
c
c-- Error check: oxygen must be present in the aqueous phase when
c-- biodegradation is considered.
  if (lctrl(3).and.casol(ipt(15)+2).lt.0.d0)
    + Call ErrorMessage(79,0,ipt(29))
c
c-- Nutrient information:
c-- Read a logical variable indicating if a nutrient
c-- is to be modeled.
  call commnt(11,21)
  read (11,*) lctrl(9)
c
c-- Error check: lctrl(3) = true for nutrient to be considered.
  if (lctrl(9).and.not.lctrl(3)) Call ErrorMessage(77,0,ipt(29))
c
c-- If nutrient is modeled, then read the nutrient chemical properties.
c-- Error check: values must be nonnegative except vapor pressure.
  istop = ipt(15) + 3
  if (lctrl(9)) Then
    istop = istop + 1
    if (istop.gt.ncmp) Call ErrorMessage(43,0,ipt(29))
    call commnt(11,21)
    read (11,*) ic,cname(istop),cmw(istop),cvp(istop),
    + ,cvvis(istop),cden(istop),cmdif(2*istop-1),

```

```

+ ,cmdif(2*istop),chen(istop),casol(istop)
  if (ic.ne.istop) Call ErrorMessage(44,0,ipt(29))
  if (ic.ne.i) Call ErrorMessage(44,0,ipt(29))
  if (cmw(ic).lt.zer0) Call ErrorMessage(89,0,ipt(29))
  if (cvvis(ic).lt.zer0) Call ErrorMessage(90,0,ipt(29))
  if (cden(ic).lt.zer0) Call ErrorMessage(91,0,ipt(29))
  if (cmdif(2*ic-1).lt.zer0) Call ErrorMessage(92,0,ipt(29))
  if (cmdif(2*ic).lt.zer0) Call ErrorMessage(92,0,ipt(29))
  if (chen(ic).lt.zer0) Call ErrorMessage(93,0,ipt(29))
c
c-- Error check: casol(nutrient) must be positive if lctrl(9) = true.
  if (casol(istop).lt.zer0) Call ErrorMessage(78,0,ipt(29))
  Endif
  ipt(65) = istop
  ipt(66) = istop * ipt(1)
c
c-- Write component data.
  Write (21,510) ipt(15)
  Do 250 i = 1,istop
    Write (21,511) cname(i),i,cmw(i),cvp(i),cvvis(i),
    + ,cden(i),cmdif(2*i-1),cmdif(2*i),chen(i),casol(i)
250 continue
c
c-- Adjust units.
c-- Convert aqueous solubilities to mole ratios.
c-- Convert all others to SI units.
  cmwaq = cmw(ipt(15)+1)
  do 260 i=1,istop
    if(cvp(i).gt.rone) then
      cvp(i) = chen(i) * casol(i)
    else
      cvp(i) = cvp(i) * patm ! vapor pressure = Pa
    end if
    casol(i) = casol(i)*cmwaq/(cmw(i)*cden(ipt(15)+1))
    cmw(i) = cmw(i) * 1.0d-3 ! molecular weight = kg/mole
    cvvis(i) = cvvis(i) * 1.0d-3 ! viscosity = Pa s
    cmdif(2*i-1) = cmdif(2*i-1) * 1.0d-4 ! diffusivity = m^2/s
    cmdif(2*i) = cmdif(2*i) * 1.0d-4
260 chen(i) = chen(i) * patm ! henry's constant = Pa m^3/kg
c
===== MASS EXCHANGE INFORMATION =====
c
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Mass Transfer Information'
c
c-- Read the mass exchange coefficients for the various components
c-- between contiguous phases. If the mass exchange coefficient
c-- for a given component and phase pair is zero, that component
c-- does not partition between that phase pair. Exchange coefficients
c-- are ordered aqueous/gas, aqueous/NAPL, gas/NAPL, aqueous/biophase,
c-- and aqueous/solid.
c-- Error Check: Component number must be between 1 and istop.
c
  write (21,512)
  do 270 i=1,istop
    call commnt(11,21)
    read (11,*) ic,kex(5*(ic-1)+1),kex(5*(ic-1)+2)
    + ,kex(5*(ic-1)+3),kex(5*(ic-1)+4),kex(5*(ic-1)+5)
    read (11,*) ic,kmax(5*(ic-1)+1),kmax(5*(ic-1)+2)
    + ,kmax(5*(ic-1)+3),kmax(5*(ic-1)+4),kmax(5*(ic-1)+5)
    if (ic.lt.1.or.ic.gt.istop) Call ErrorMessage(45,0,ipt(29))
270 write (21,513) cname(ic),kex(5*(ic-1)+1),kex(5*(ic-1)+2)
    + ,kex(5*(ic-1)+3),kex(5*(ic-1)+4),kex(5*(ic-1)+5)
    write (21,582)
    do 275 i=1,istop
      write (21,513) cname(i),kmax(5*(i-1)+1),kmax(5*(i-1)+2)
      + ,kmax(5*(i-1)+3),kmax(5*(i-1)+4),kmax(5*(i-1)+5)
    do 275 ii = 1, 5
275 if(kmax(5*(i-1)+ii).lt.0.05d0.or.kmax(5*(i-1)+ii).gt.rone)
    + Call ErrorMessage(134,0,ipt(29))
c
c-- Create the pointer for phase compositions. First consider
c-- the gas phase. Components are not included if their vapor
c-- pressures are negative. ipt(3) is the number of components in
c-- the gas phase and ipt(13) is the number of organic components in
c-- the gas phase.
  ipt(3)=0
  ipt(13)=0

```



```

do 280 i=1,istop
  if(cvp(i).ge.zer0) then
    ipt(3)=ipt(3)+1
    icp(ipt(3))=i
  end if
  if(i.eq.ipt(15)) ipt(13)=ipt(3)
280 continue
c
c— Error Check: if present gas phase must contain nitrogen.
  if (ipt(3).gt.0 .and. cvp(ipt(15)+3).lt.zer0)
+ Call ErrorMessage (69,0,ipt(29))
C
C— Next consider the aqueous phase. Components are not included if
C— their aqueous solubilities are negative. ipt(4) is the number of
C— components in the aqueous phase and ipt(14) is the number of
C— organic components in the aqueous phase.
  ipt(4)=0
  ipt(14)=0
  do 290 i=1,istop
    if(casol(i).ge.zer0) then
      ipt(4)=ipt(4)+1
      icp(ipt(3)+ipt(4))=i
    end if
    if(i.eq.ipt(15)) ipt(14)=ipt(4)
290 continue
c
c— Error Check: if present aqueous phase must contain water,
  if (ipt(4).gt.0 .and. casol(ipt(15)+1).lt.zer0)
+ Call ErrorMessage (70,0,ipt(29))
C
C— Next consider the organic phase if lctrl(24) = true. All organic
C— components are included in the NAPL when present.
  ipt(5) = 0
  if(lctrl(24)) then
    ipt(5)=ipt(15)
    do 300 i=1,ipt(5)
      icp(ipt(3)+ipt(4)+i)=i
300 continue
  EndIf
C
C— Next consider the solid phase. Sorption is considered for all
C— organic component in the aqueous phase with nonzero aqueous/solid
C— exchange coefficients. ipt(6) is the number of components in the
C— solid phase. ipt(16) is the number of organic components in the
C— solid phase.
  ipt(6)=0
  ipt(16)=0
  If (lctrl(25)) Then
    do 310 i=1,ipt(15)
      if(kex(5*(i-1)+5).ne.zer0) then
        ipt(6)=ipt(6)+1
        icp(ipt(3)+ipt(4)+ipt(5)+i)=i
      end if
      if(i.eq.ipt(15)) ipt(16)=ipt(6)
310 continue
  EndIf
C
C— Define pointers:
C— ipt(8) - start of gas phase component section - 1
C— ipt(9) - start of aqueous phase component section - 1
C— ipt(10) - start of organic phase component section - 1
C— ipt(11) - start of solid phase component section - 1
C— ipt(12) - start of bio-phase component section - 1
  ipt(8) = 0
  ipt(9) = ipt(8) + ipt(3)*ipt(1)
  ipt(10) = ipt(9) + ipt(4)*ipt(1)
  ipt(11) = ipt(10) + ipt(5)*ipt(1)
  ipt(12) = ipt(11) + ipt(6)*ipt(1)
  ipt(58) = ipt(3) + ipt(4)
  ipt(59) = ipt(3) + ipt(4) + ipt(5)
  ipt(60) = ipt(3) + ipt(4) + ipt(5) + ipt(6)
c
c===== MATERIAL PROPERTY BLOCK DATA =====
c
  if (ipt(29).ne.0) write (ipt(29),*)
+ 'Reading Material Property Block Data'
c
c— Read soil physical properties:

```

```

c— porosity (-); horizontal and vertical permeability (m**2);
c— bulk density (gm/l);
c— organic carbon fractional content(-)
c— Error Check: block number=(0,ipt(26)); porosity=(0,1);
c— permeability>0; density>0; foc=(0,1);
  call commnt (11,21)
  Do 340 i = 1,ipt(26)
    Read (11,*) ii,bphi(ii),bpermh(ii),bpermv(ii),bsden(ii),
+ b foc(ii)
    If (ii.lt.1 .or. ii.gt.ipt(26)) Call ErrorMessage (27,0,ipt(29))
    If (bphi(ii).lt.zer0 .or. bphi(ii).gt.rone)
+ Call ErrorMessage (28,0,ipt(29))
    If (bpermh(ii).lt.zer0 .or. bpermv(ii).lt.zer0)
+ Call ErrorMessage (29,0,ipt(29))
    If (bsden(ii).lt.zer0) Call ErrorMessage (30,0,ipt(29))
    If (b foc(ii).lt.zer0 .or. b foc(ii).gt.rone)
+ Call ErrorMessage (31,0,ipt(29))
340 continue
c
c— Read the water retention parameters:
c— residual water saturation(-); van Genuchten n value;
c— van Genuchten alpha value
c— Define the van Genuchten m value.
c— Error Check: block number=(0,ipt(26)); residual saturation=(0,1);
c— n value>0; alpha value>0.
  call commnt (11,21)
  Do 350 i = 1,ipt(26)
    Read (11,*) ii,bsrw(ii),bvgn(ii),bvga(ii)
    If (ii.lt.1 .or. ii.gt.ipt(26)) Call ErrorMessage (27,0,ipt(29))
    If (bsrw(ii).lt.zer0 .or. bsrw(ii).gt.rone)
+ Call ErrorMessage (33,0,ipt(29))
    If (bvgn(ii).lt.zer0) Call ErrorMessage (34,0,ipt(29))
    If (bvga(ii).lt.zer0) Call ErrorMessage (35,0,ipt(29))
    bvgn(ii) = rone - rone / bvgn(ii)
350 continue
c
c— Read the dispersion parameters:
c— longitudinal and transverse dispersivity (m).
c— Error Check: all parameters > 0.
  call commnt (11,21)
  Do 360 i = 1,ipt(26)
    Read (11,*) ii,bdisl(ii),bdist(ii)
    If (ii.lt.1 .or. ii.gt.ipt(26)) Call ErrorMessage (27,0,ipt(29))
    If (bdisl(ii).lt.zer0 .or. bdist(ii).lt.zer0)
+ Call ErrorMessage (36,0,ipt(29))
360 continue
c
c— Write the material properties for each block.
  write (21,514)
  Do 370 i = 1,ipt(26)
    Write (21,515) i,bphi(i),bpermh(i),bpermv(i),bsden(i),b foc(i),
+ bsrw(i),bvgn(i),bvga(i),
+ bdisl(i),bdist(i)
370 continue
c
c===== DISPERSION TENSOR =====
c
c
c— If lctrl(21) is true the hydrodynamic dispersion tensor is
c— computed internally for all components. If lctrl(21) is false
c— read hydrodynamic dispersion tensor information directly. The
c— hydrodynamic dispersion tensor is read for compound ic first
c— for the gas phase and then for the aqueous phase on separate
c— lines. Hydrodynamic dispersion tensors must be entered for all
c— components present when lctrl(21) is false. Units (m**2/sec).
  if (ipt(29).ne.0) write (ipt(29),*)
+ 'Reading Dispersion Parameters'
  call commnt(11,21)
  read (11,*) lctrl(21)
  write (21,580) lctrl(21)
  call commnt(11,21)
  if(.not.lctrl(21)) then
    write (21,581)
    do 380 i = 1, istop
      call commnt(11,21)
      read (11,*) ic, d(8*ic-7),d(8*ic-6),d(8*ic-5),d(8*ic-4)
      read (11,*) ic, d(8*ic-3),d(8*ic-2),d(8*ic-1),d(8*ic)
      write (21,585) cname(ic),'gas ',d(8*ic-7),d(8*ic-6)

```

```

+      ,d(8*ic-5),d(8*ic-4)
380 write (21,585) cname(ic),'aqueous',d(8*ic-3),d(8*ic-2)
+      ,d(8*ic-1),d(8*ic)
end if
c
c— Conversions:
c— convert solid phase density from gm/cc media to gm/L media;
c— reset bdisl(i) as the difference between the longitudinal and
c— transverse dispersivity;
c— redefine 'bpermh' as an anisotropy factor: kh = kh / kv;
c— multiply matrix compressibility by a constant 1/2; determine if the
c— entire domain is incompressible.
Do 390 i = 1, ipt(26)
  bsden(i) = bsden(i) * 1.0d3
  bdisl(i) = bdisl(i)-bdisl(i)
  bpermh(i) = bpermh(i) / bpermv(i)
390 continue
c
c===== SORPTION PARAMETERS =====
c
if (ipt(29).ne.0) write (ipt(29),*)
+   'Reading Sorption Parameters'
write(21,589)
c
c— Read data for rate limited sorption.
If (lctrl(25)) Then
c
c— Field 1 - Model type:
c— Read a logical variable indicating if a one or two compartment
c— sorption model is used.
c— Error check: for two compartment case make sure ipt(26)=1 and
c— ipt(15)=1.
call commnt (11,21)
read (11,*) lctrl(19)
if (lctrl(19).and.ipt(26).ne.1) Call ErrMessage (38,0,ipt(29))
if (lctrl(19).and.ipt(15).ne.1) Call ErrMessage (38,0,ipt(29))
c
c— Read two compartment model data:
c— The two compartment model has a slow and a fast compartment.
c— Both compartments are modeled with the Freundlich equation.
c— Four parameters must be input:
c— (1) a multiplier to convert the slow compartment bok to
c— the fast compartment bok;
c— (2) a multiplier to convert the slow compartment bom to
c— the fast compartment bom;
c— (3) a multiplier to convert the slow compartment kex to
c— the fast compartment kex;
c— (4) the fraction of solid phase density in the fast compartment.
c— Error Check: Kf and n=1/m > 0; slow/fast fraction = (0,1)
If (lctrl(19)) Then
  call commnt(11,21)
  read (11,*) xbok, xbom, xkex, xden
  if (xbok.lt. zero) Call ErrMessage (39,0,ipt(29))
  if (xbom.lt. zero) Call ErrMessage (40,0,ipt(29))
  if (xden.lt.zero .or. xden.gt.rone)
+   Call ErrMessage (41,0,ipt(29))
  ipt(6) = ipt(6) + 1
  ipt(16) = ipt(16) + 1
  ipt(60) = ipt(60) + 1
  ipt(12) = ipt(12) + ipt(1)
  icp(ipt(3)+ipt(4)+ipt(5)+ipt(6)) =
+   icp(ipt(3)+ipt(4)+ipt(5)+1)
EndIf
c
c— Read sorption parameters data:
c— Read for each material property block enter:
c— (1) the kf parameter values for each organic component in
c— order from 1 to the number of components:
c— units (micro gm)/(gm solid) with aqueous concentration
c— in mg / l;
c— (2) the n parameter values order in the same way.
c— Error Check: Kf and n=1/m > 0
call commnt(11,21)
do 400 i = 0, ipt(26)-1
  read (11,*) ii,(bok((ii-1)*ipt(15)+j),j=1,ipt(15))
  read (11,*) ii,(bom((ii-1)*ipt(15)+j),j=1,ipt(15))
400 continue
do 401 i = 0, ipt(26)-1

```

```

do 401 j = 1, ipt(15)
  if (bok(i*ipt(15)+j) .lt. zero)
+   Call ErrMessage (39,0,ipt(29))
401 if (bom(i*ipt(15)+j) .lt. zero)
+   Call ErrMessage (40,0,ipt(29))
c
c— Write rate-limited sorption parameters:
write (21,516) ! write sorption data
Do 410 i = 0, ipt(26)-1
  write (21,517) i+1
  write (21,518) (cname(j),bok(i*ipt(15)+j),cname(j),
+   bom(i*ipt(15)+j),j=1,ipt(15))
410 continue
If (lctrl(19)) Then ! write two compartment data
  write (21,519) xbok, xbom, xden, xkex
Endif
c
c— Conversions:
c— Convert units for sorption parameters. This assumes the input
c— values for Kf or Koc are micrograms/gram (with aqueous
c— concentration in mg/l) and bom is 1/m (unitless).
c— The converted Kf or Koc is in grams organic / gram solid.
do 420 i = 0, ipt(26)-1
  do 420 j = 1, ipt(15)
    if (bok(i*ipt(15)+j).ge.zero) then
      bok(i*ipt(15)+j) = bok(i*ipt(15)+j) * 1.0d-6
      bom(i*ipt(15)+j) = rone / bom(i*ipt(15)+j)
    else if (bok(i*ipt(15)+j).lt.zero) then
      bok(i*ipt(15)+j) = -bok(i*ipt(15)+j)*1.0d-6*bfoc(i+1)
      bom(i*ipt(15)+j) = rone
    end if
420 continue
c
c— Convert xbom to be consistent with bom.
if (lctrl(19)) xbom = rone/xbom
c
c— Initial the retardation factors to one when a separate solid
c— phase is considered.
do 404 i = 1, istop
404 krtcd(i) = rone
c
c— Read equilibrium sorption parameters (retardation factors).
Else
  lctrl(19) = .false.
c
c— Read retardation factors if desired. Retardation factors
c— can only be used when nonequilibrium sorption is not
c— being considered. Retardation factors are input as
c— component number as input above and then the retardation
c— factor. Retardation can be considered for oxygen or nutrient.
c
call commnt(11,21)
read(11,*) lretrd
call commnt(11,21)
if (.not.lretrd) then
  write(21,590)
  do 405 i = 1, istop
405 krtcd(i) = rone
  else
  write(21,588)
  do 406 i = 1, istop
    read(11,*) ic, krtcd(ic)
    write(21,574) cname(ic),krtcd(ic)
    if (krtcd(ic).lt.rone)
+   Call ErrMessage (128,1,ipt(29))
406 continue
  end if
Endif
c
c===== BIOLOGICAL PARAMETERS =====
c
c— This block read only if biotransformation eqs are solved.
If (lctrl(3)) Then
  if (ipt(29).ne.0) write (ipt(29),*)
+   'Reading Biological Parameters'
c
c— Read an integer indicating the number of organic components in
c— the biophase. The biophase always contains oxygen and nutrient

```

```

c— if present.
c— Error Check: lctrl(3) must be true if ipt(17)>0.
  call commnt(11,21)
  read (11,*) ipt(17)
  if (.not.lctrl(3) .and. ipt(17).gt.0)
+   Call ErrorMessage (81,0,ipt(29))
  if (ipt(17) .gt. ipt(15)) Call ErrorMessage (130,1,ipt(29))
C
C— Biological parameters.
C— ipt(7) is the number of components in the bio-phase. The bio-phase
C— always includes oxygen and nutrient if present.
  if (ipt(17).gt.0) then
    if (lctrl(9)) then
      ipt(7) = ipt(17) + 2
      icp(ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(17)+2) = ipt(15) + 4
    else
      ipt(7) = ipt(17) + 1
    end if
    icp(ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(17)+1) = ipt(15) + 2
  c
c— Add the biomass as a component to the biophase.
c— Initial cname with "biomass".
  ipt(7) = ipt(7) + 1
  icp(ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(7)) = istop+1
  cname(istop+1) = ' biomass '
  else if (ipt(17).eq.0) then
    ipt(7) = ipt(17)
  end if
c
c— Generate pointer.
  ipt(61) = ipt(3) + ipt(4) + ipt(5) + ipt(6) + ipt(7)
c
c— Read 2 logical biodegradation control switches:
c— lctrl(17) indicates if a steady state biomass is to be used;
c— lctrl(16) indicates biodegradation equations are modeled as a
c— sink term in the aqueous transport equations; otherwise their
c— modeled as rate-limited exchange to a separate biophase.
  call commnt(11,21)
  read (11,*) lctrl(17),lctrl(16)
c
c— Determine the type of kinetics.
c— 1 - standard monod kinetics
c— 2 - monod kinetics with substrate inhibition
c— 3 - monod kinetics with lumped substrate inhibition
c— 4 - monod kinetics with saturation dependency
c— 5 - monod kinetics with saturation dependency and
c— substrate inhibition
  if (lctrl(3)) then
    call commnt(11,21)
    read (11,*) ipt(39)
  c
c— Error Check: Current version restriction.
  if (ipt(39).gt.2) call ErrorMessage (68,1,ipt(29))
c
c— Read Monod parameters for each component of the biophase:
c— (1) component number
c— (2) electron acceptor use coefficient gm O2/gm substrate)
c— (3) nutrient use coefficient (gm n/gm substrate)
c— (4) maximum substrate use rate (gm substrate/gm biomass*sec)
c— (5) half saturation constant (gm substrate/l)
c— (6) yield coefficient (gm biomass/gm substrate)
c— (7) inhibition constant (unitless) expressed as a fraction of
c— the aqueous solubility. For substrate and nutrient this
c— turns off metabolism when the concentration is above this
c— threshold concentration and for electron acceptor this
c— turns off metabolism when the concentration is below this
c— threshold concentration. In both cases hyperbolic
c— functions are used.
c— Error Check: Component number must be defined; parameters values
c— must be nonnegative.
  ibpt = ipt(3)+ipt(4)+ipt(5)+ipt(6)
  do 430 i=1,ipt(7)-1
    call commnt(11,21)
    read (11,*) ic,fuse(ic),
+     fuse(ic+istop),umax(ic),
+     khalf(ic),xyield(ic),
+     kinhib(ic)
    icp(ibpt+i) = ic

```

```

    if (icp(ibpt+i).lt.0 .or. icp(ibpt+i).gt.istop)
+     Call ErrorMessage (45,0,ipt(29))
    if (fuse(ic).lt.zer0)
+     Call ErrorMessage (94,0,ipt(29))
    if (fuse(ic+istop).lt.zer0)
+     Call ErrorMessage (95,0,ipt(29))
    if (umax(ic).lt.zer0)
+     Call ErrorMessage (96,0,ipt(29))
    if (khalf(ic).lt.zer0)
+     Call ErrorMessage (97,0,ipt(29))
    if (xyield(ic).lt.zer0)
+     Call ErrorMessage (98,0,ipt(29))
    if (kinhib(ic).lt.zer0)
+     Call ErrorMessage (99,0,ipt(29))
430 continue
c
c— Read five more bio parameters:
c— (1) decay coefficient (1/sec)
c— (2) minimum biomass (gm/l media)
c— (3) maximum biomass (gm/l media)
c— (4) initial uniform biomass (gm/l media)
c— (5) delay period for initiation of bioreactions (sec)
c— Error Check: parameters must be nonnegative, initial biomass must
c— be between minimum and maximum biomass.
  call commnt(11,21)
  read (11,*) kd,xbmin,xbmax,xinit,t(11)
  if (kd.lt.zer0) Call ErrorMessage (99,0,ipt(29))
  do 441 i = 1, ipt(17)
441  if (9.990d0*kd.gt.umax(i)) Call ErrorMessage (37,1,ipt(29))
    if (xbmin.lt.zer0) Call ErrorMessage (100,0,ipt(29))
    if (xbmax.lt.zer0) Call ErrorMessage (101,0,ipt(29))
    if (xinit.lt.zer0) Call ErrorMessage (102,0,ipt(29))
    if (t(11).lt.zer0) Call ErrorMessage (103,0,ipt(29))
    if (xinit.lt.xbmin .or. xinit.gt.xbmax)
+     Call ErrorMessage (104,0,ipt(29))
  c
c— Write biological parameters.
  write (21,520) ipt(17),lctrl(17),lctrl(16),ipt(39),xinit
+  ,xbmin,xbmax,kd,t(11)
  Do 440 i = 1, ipt(7)-1
    write (21,521) cname(icp(ibpt+i)),fuse(icp(ibpt+i)),
+     fuse(icp(ibpt+i)+istop),umax(icp(ibpt+i)),
+     khalf(icp(ibpt+i)),xyield(icp(ibpt+i)),
+     kinhib(icp(ibpt+i))
440 continue
  EndIf
c
c— Bio eqs are not solved. Set control switches.
  Else
    ipt(17) = 0
    ipt(7) = 0
    lctrl(16) = .false.
    ipt(61) = ipt(60)
  EndIf
c
===== PHASE PARAMETERS AND COMPOSITION =====
c
  if (ipt(29).ne.0) write (ipt(29),*) 'Reading Phase Parameters'
c
c— Read water phase parameters:
c— (1) water phase viscosity (cPoise)
  call commnt(11,21)
  read (11,*) wvis
c
c— Determine if slip flow is modeled and read the Klinkenberg coeff
  call commnt(11,21)
  read (11,*) lctrl(20),b
  b = b * patm ! convert from atm to Pa
c
c— Write phase parameters and compositions.
  write(21,522) wvis,lctrl(20),b
  write(21,523) 'Gas', ipt(3)
  If (ipt(3).gt.0) write(21,524) (cname(icp(i)),i=1,ipt(3))
  write(21,523) 'Aqueous', ipt(4)
  If (ipt(4).gt.0) write(21,524) (cname(icp(i+ipt(3))),i=1,ipt(4))
  write(21,523) 'Organic Liquid', ipt(5)
  If (ipt(5).gt.0) write(21,524)
+ (cname(icp(ipt(3)+ipt(4)+i)),i=1,ipt(5))

```

```

write(21,523) 'Solid', ipt(6)
If (ipt(6).gt.0) write(21,524)
+ (cname(icmp(ipt(3)+ipt(4)+ipt(5)+i)),i=1,ipt(6))
write(21,523) 'Microbial', ipt(7)
If (ipt(7).gt.0) write(21,524)
+ (cname(icmp(ipt(3)+ipt(4)+ipt(5)+ipt(6)+i)),i=1,ipt(7))
c
c— Convert viscosity from cPoise to Pa-s
wvis = ronc / (wvis * 1.0d-3)
c
===== TEMPERATURE PARAMETERS =====
c
if (ipt(29).ne.0) write (ipt(29),*)
+ 'Reading Temperature Parameters'
c
c— Read a logical variable indicating if the temp dist is uniform.
call commnt(11,21)
read (11,*) lctrl(10)
write (21,525) lctrl(10)
C
C— Read and write the uniform temperature distribution in
centigrade.
call commnt(11,21)
If (lctrl(10)) then
read (11,*) ctemp
write (21,526) ctemp
do 450 i=1,ipt(1)
c
c— Express temperatures in absolute.
temp(i) = ctemp + tabs
450 continue
c
c— Read nonuniform vertical temperature profile (one temperature for
c— each node along the vertical boundary).
c— Nonuniform temperature can only be defined in association with a
c— generated rectangular grid. Error check on this.
else
if (igrd .lt. 0) Call ErrorMessage (132,0,ipt(29))
write (21,527)
do 460 i = 1,ipt(86)
read (11,*) depthnd, tnode
if(igrd.eq.1.and.i.gt.1) tnode = (tnode+tnode2)/2.0d0
tnode2 = tnode
do 460 ii = 1,ipt(87)
if(igrd.eq.2) then
if(ipt(87).le.ipt(86)) then
itnode = ii + (i-1)*ipt(87)
else
itnode = i + (ii-1)*ipt(86)
end if
temp(itnode) = tnode
idepth(itnode) = i
if (depthnd .ne. znode(itnode))
+ Call ErrorMessage (133,0,ipt(29))
zdepth(i) = znode(itnode)
else if(igrd.eq.1) then
if(ipt(87).le.ipt(86)) then
itnode = ii + (i-1)*(2*ipt(87)-1)
itnod2 = itnode - ipt(87) + 1
else
itnode = i + (ii-1)*(2*ipt(86)-1)
itnod2 = itnode + ipt(86) - 1
end if
temp(itnode) = tnode
idepth(itnode) = 2*i-1
if (depthnd .ne. znode(itnode))
+ Call ErrorMessage (133,0,ipt(29))
zdepth(2*i-1) = znode(itnode)
if(i.gt.1.or.ii.lt.ipt(87)) then
temp(itnod2) = tnode
idepth(itnod2) = 2*i
zdepth(2*i-3) = (znode(itnode)+znode(itnod3))/2.0d0
end if
itnod3 = itnode
end if
460 continue
c
c— Express temperatures in absolute.

```

```

do 470, i = 1, ipt(1)
470 temp(i) = temp(i) + 273.150d0
write (21,528) (i, temp(i), i = 1, ipt(1))
c
c— Read vectors of temperature dependent parameters. Values are
c— needed for each node at the vertical boundary starting at the
c— surface. Intermediate values are linearly interpolated when the
c— union jack grid is selected. Values are needed for the first
c— five parameters below for every component. Give the five sets
c— for a given component and then go to the next component. Use
c— the original component ordering as in the component information
c— section. After all the component values give the kd values.
c
c— Temperature dependent parameters are as follows:
c— (1) component vapor pressure (atm)
c— (2) component vapor viscosity (cPoise)
c— (3) component Henry's law constant (atm l/gm)
c— (4) component aqueous solubility (gm/l)
c— (5) component maximum utilization rate
c— (gm substrate / gm biomass sec )
c— (6) biomass decay rate (1/sec)
c
c— Read component vapor pressures.
if(igrd.eq.2) then
ipt(88) = ipt(86)
else if (igrd.eq.1) then
ipt(88) = 2*ipt(86)-1
endif
ipt(89) = 5 * ipt(88)
ii = 0
c
c— Loop over the number of components.
do 477 ic = 1, istop
write(21,570) cname(ic)
c
c— Read component vapor pressure.
call commnt(11,21)
read(11,*) (dtemp(i+ii),i=1,ipt(86))
do 471 i = ipt(86),1,-1 ! read nodal vapor pressure
if(igrd.eq.2) then
dtemp(i+ii) = dtemp(i+ii)*patm - cvp(ic)
else if (igrd.eq.1) then
dtemp(2*i-1+ii) = dtemp(i+ii)*patm - cvp(ic)
if(i.gt.1) dtemp(2*i-2+ii)
+ = patm*(dtemp(i+ii)+dtemp(i-1+ii))/2.0d0-cvp(ic)
end if
471 continue
if(igrd.eq.2) then
ii=ipt(86)+i
else if (igrd.eq.1) then
ii = 2*ipt(86)-1+ii
endif
c
c— Read component vapor viscosity.
call commnt(11,21)
read(11,*) (dtemp(i+ii),i=1,ipt(86))
do 472 i = ipt(86),1,-1
if(igrd.eq.2) then
dtemp(i+ii) = dtemp(i+ii)*1.0d-3 - cvvis(ic)
else if (igrd.eq.1) then
dtemp(2*i-1+ii) = dtemp(i+ii)*1.0d-3 - cvvis(ic)
if(i.gt.1) dtemp(2*i-2+ii) = 1.0d-3*
+ (dtemp(i+ii)+dtemp(i-1+ii))/2.0d0 - cvvis(ic)
end if
472 continue
if(igrd.eq.2) then
ii=ipt(86)+i
else if (igrd.eq.1) then
ii = 2*ipt(86)-1+ii
endif
c
c— Read component Henry's law coefficients.
call commnt(11,21)
read(11,*) (dtemp(i+ii),i=1,ipt(86))
do 473 i = ipt(86),1,-1
if(igrd.eq.2) then
dtemp(i+ii) = patm*dtemp(i+ii) - chen(ic)
else if (igrd.eq.1) then

```

```

    dtemp(2*i-1+ii) = patm*dtemp(i+ii) - chen(ic)
  + if(i.gt.1) dtemp(2*i-2+ii) = patm*
    (dtemp(i+ii)+dtemp(i-1+ii))/2.0d0 - chen(ic)
  + end if
473 continue
  if(igrd.eq.2) then
    ii=ipt(86)+ii
  else if (igrd.eq.1) then
    ii = 2*ipt(86)-1+ii
  endif
c
c— Read component aqueous solubilities.
  call commnt(11,21)
  read(11,*) (dtemp(i+ii),i=1,ipt(86))
  cmwq = cmw(ipt(15)+1)/(cmw(ic)*cden(ipt(15)+1))
  do 474 i = ipt(86),1,-1
    if(igrd.eq.2) then
      dtemp(i+ii) = cmwq*dtemp(i+ii) - casol(ic)
    else if (igrd.eq.1) then
      dtemp(2*i-1+ii) = cmwq*dtemp(i+ii) - casol(ic)
    if(i.gt.1) dtemp(2*i-2+ii) = cmwq*
      (dtemp(i+ii)+dtemp(i-1+ii))/2.0d0 - casol(ic)
  + end if
474 continue
  if(igrd.eq.2) then
    ii=ipt(86)+ii
  else if (igrd.eq.1) then
    ii = 2*ipt(86)-1+ii
  endif
c
c— Read component maximum substrate utilization rate.
  call commnt(11,21)
  read(11,*) (dtemp(i+ii),i=1,ipt(86))
  do 475 i = ipt(86),1,-1
    if(igrd.eq.2) then
      dtemp(i+ii) = dtemp(i+ii) - umax(ic)
    else if (igrd.eq.1) then
      dtemp(2*i-1+ii) = dtemp(i+ii) - umax(ic)
    if(i.gt.1) dtemp(2*i-2+ii)
  + = (dtemp(i+ii)+dtemp(i-1+ii))/2.0d0 - umax(ic)
  + end if
475 continue
  if(igrd.eq.2) then
    ii=ipt(86)+ii
  else if (igrd.eq.1) then
    ii = 2*ipt(86)-1+ii
  endif
  do 477 ip = 1, ipt(86)
    write(21,571) zdepth(ip)
  + (dtemp(ip+ipp*ipt(88)+(ic-1)*ipt(89)),ipp=0,4)
477 continue
c
c— Read biomass decay rate.
  call commnt(11,21)
  write(21,572)
  read(11,*) (dtemp(i+ii),i=1,ipt(86))
  do 476 i = ipt(86),1,-1
    if(igrd.eq.2) then
      dtemp(i+ii) = dtemp(i+ii) - kd
    else if (igrd.eq.1) then
      dtemp(2*i-1+ii) = dtemp(i+ii) - kd
    if(i.gt.1) dtemp(2*i-2+ii)
  + = (dtemp(i+ii)+dtemp(i-1+ii))/2.0d0 - kd
  + end if
476 continue
  write(21,573) (zdepth(ip),dtemp(ip+istop*ipt(89)),ip=1,ipt(86))
  endif
c
c— Compute the gamma factor for computing mixture vapor viscosities.
c— Initialize gas phase viscosity if it is a constant.
  lctrl(13) = .false.
  if (ipt(3) .gt. 1) then
    lctrl(13) = .true.
  do 320 node = 1, ipt(1)
  do 320 i = 1,ipt(3)
    ii = icp(i)
    if(lctrl(10)) then
      cvvist = cvvis(ii)

```

```

  else
    cvvist = cvvis(ii)
  + dtemp(ipt(88)+(ii-1)*ipt(89)+idepth(node))
  end if
  i3 = (i-1) * ipt(3) + (node-1)*ipt(3)*ipt(3)
  do 320 j = 1,ipt(3)
    jj = icp(j)
    if(lctrl(10)) then
      cvvist2 = cvvis(jj)
    else
      cvvist2 = cvvis(jj)
  + dtemp(ipt(88)+(jj-1)*ipt(89)+idepth(node))
  end if
  gamma(i3+j) = ((rone + dsqrt(cvvist/cvvist2) *
  + (cmw(jj)/cmw(ii))**0.250d0) ** 2) /
  + dsqrt(8.0d0 * (rone + (cmw(jj)/cmw(ii))))
320 continue
  else ! vapor viscosity is a constant
  do 330 i = 1,ipt(1)
    if(lctrl(10)) then
      cvvist = cvvis(icp(1))
    else
      cvvist = cvvis(icp(1))
  + dtemp(icp(1)-1)*ipt(88)+idepth(i))
  end if
  vis(i) = rone / cvvist
330 continue
  endif
c
c===== OUTPUT CONTROL PARAMETERS =====
c
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Output Control Parameters'
c
c— Read a logical variable indicating if initial conditions
c— should be printed.
  call commnt(11,21)
  read (11,*) lprnt(3)
c
c— Initial print counters.
  ipt(69) = 0
  ipt(70) = 0
  ipt(71) = 0
  ipt(72) = 0
  ipt(73) = 0
c
c— Read and write print switches for specific variables:
  call commnt(11,21)
  iend = ipt(3) + ipt(4) + ipt(5) + ipt(6) + ipt(7)
  do 480 i = 1, 7
480 read(11,*) lprnt(i+7),lcon(i)
  read(11,*) lprnt(15),lcon(8)
  if(lprnt(15).or.lcon(8))
  + read(11,*) ipt(69),(icp(iend+i),i=1,ipt(69))
  read(11,*) lprnt(16),lcon(9)
  if(lprnt(16).or.lcon(9))
  + read(11,*) ipt(70),(icp(iend+ipt(69)+i),i=1,ipt(70))
  read(11,*) lprnt(17),lcon(10)
  if(lprnt(17).or.lcon(10))
  + read(11,*) ipt(71),(icp(iend+ipt(69)+ipt(70)+i),i=1,ipt(71))
  read(11,*) lprnt(18),lcon(11)
  if(lprnt(18).or.lcon(11)) read(11,*) ipt(72)
  + ,(icp(iend+ipt(69)+ipt(70)+ipt(71)+i),i=1,ipt(72))
  read(11,*) lprnt(19),lcon(12)
  if(lprnt(19).or.lcon(12)) read(11,*) ipt(73)
  + ,(icp(iend+ipt(69)+ipt(70)+ipt(71)+ipt(72)+i),i=1,ipt(73))
  read(11,*) lprnt(29),lcon(18) ! element TPH coccentration
  do 485 i = 13, 17
485 read(11,*) lprnt(i+7),lcon(i)
c
c— Setup pointers.
  ipt(81) = 0
  ipt(82) = 0
  iend2 = ipt(69) + ipt(70) + ipt(71) + ipt(72) + ipt(73)
  ipt(74) = iend2
  call commnt(11,21)
  read(11,*) lplt(1)
  if(lplt(1)) read(11,*) ipt(81),(icp(iend+2*iend2+i),i=1,2*ipt(81))

```

```

call commn(11,21)
read(11,*) lplt(2)
if(lplt(2)) read(11,*) ipt(82)
+ ,(icp(iend+2*iend2+2*ipt(81)+i),i=1,2*ipt(82))
c
c- Error check: time series plot is restricted to six components.
if(ipt(81)+ipt(82).gt.6) call ErrMessage (124,1,ipt(29))
c
c- Error check: only print out information for phases that are present.
if ((lprnt(9).or.lprnt(11).or.lprnt(12).or.lprnt(15).or.lprnt(20)
+ .or.lprnt(23).or.lcon(2).or.lcon(4).or.lcon(5).or.lcon(8)
+ .or.lcon(13)).and.ipt(3).eq.0)
+ call ErrMessage (71,1,ipt(29))
if ((lprnt(10).or.lprnt(11).or.lprnt(13).or.lprnt(16).or.lprnt(21)
+ .or.lprnt(24).or.lcon(3).or.lcon(4).or.lcon(6).or.lcon(9)
+ .or.lcon(14)).and.ipt(4).eq.0)
+ call ErrMessage (72,1,ipt(29))
if ((lprnt(14).or.lprnt(17).or.lprnt(22).or.lcon(7).or.lcon(10)
+ .or.lcon(15)).and.ipt(5).eq.0)
+ call ErrMessage (107,1,ipt(29))
if (lprnt(18).and.ipt(6).eq.0) call ErrMessage (108,1,ipt(29))
if (lprnt(19).and.ipt(7).eq.0) call ErrMessage (109,1,ipt(29))
c
c- Error check: only print out information for more components than
c- in a given phase.
c
c if (ipt(69).gt.ipt(3).or.ipt(81).gt.ipt(3))
if (ipt(69).gt.ipt(3))
+ call ErrMessage (110,1,ipt(29))
if (ipt(70).gt.ipt(4))
+ call ErrMessage (111,1,ipt(29))
if (ipt(71).gt.ipt(5)) call ErrMessage (112,1,ipt(29))
if (ipt(72).gt.ipt(6)) call ErrMessage (113,1,ipt(29))
if (ipt(73).gt.ipt(7)) call ErrMessage (114,1,ipt(29))
write(21,531) (lprnt(i),i=8,14)
write(21,532) lprnt(15)
if(lprnt(15)) write(21,545) (cname(icp(iend+i)),i=1,ipt(69))
write(21,533) lprnt(16)
if(lprnt(16)) write(21,545)
+ (cname(icp(iend+ipt(69)+i)),i=1,ipt(70))
write(21,534) lprnt(17)
if(lprnt(17)) write(21,545)
+ (cname(icp(iend+ipt(69)+ipt(70)+i)),i=1,ipt(71))
write(21,535) lprnt(18)
if(lprnt(18)) write(21,545)
+ (cname(icp(iend+ipt(69)+ipt(70)+ipt(71)+i)),i=1,ipt(72))
write(21,536) lprnt(19).lprnt(29)
if(lprnt(19)) write(21,545) (cname(
+ icp(iend+ipt(69)+ipt(70)+ipt(71)+ipt(72)+i)),i=1,ipt(73))
write(21,537) (lprnt(i),i=20,24)
write(21,538) (lcon(i),i=1,7)
write(21,539) lcon(8)
if(lcon(8)) write(21,545) (cname(icp(iend+i)),i=1,ipt(69))
write(21,540) lcon(9)
if(lcon(9)) write(21,545)
+ (cname(icp(iend+ipt(69)+i)),i=1,ipt(70))
write(21,541) lcon(10)
if(lcon(10)) write(21,545)
+ (cname(icp(iend+ipt(69)+ipt(70)+i)),i=1,ipt(71))
write(21,542) lcon(11)
if(lcon(11)) write(21,545)
+ (cname(icp(iend+ipt(69)+ipt(70)+ipt(71)+i)),i=1,ipt(72))
write(21,543) lcon(12).lcon(18)
if(lcon(12)) write(21,545) (cname(
+ icp(iend+ipt(69)+ipt(70)+ipt(71)+ipt(72)+i)),i=1,ipt(73))
write(21,544) (lcon(i),i=13,17)
write(21,546) lplt(1)
if(lplt(1)) then
do 490 i = 1, 2*ipt(81),2
c
c- Error check: node numbers for time series input must be within
c- domain.
if(icp(iend+2*iend2+i+1).lt.0.or.icp(iend+2*iend2+i+1)
+ .gt.ipt(1)) call ErrMessage (125,1,ipt(29))
490 write(21,547) cname(icp(iend+2*iend2+i)),
+ icp(iend+2*iend2+i+1)
end if
write(21,548) lplt(2)
if(lplt(2)) then

```

```

do 495 i = 1, 2*ipt(82),2
c
c- Error check: node numbers for time series input must be within
c- domain.
if(icp(iend+2*iend2+2*ipt(81)+i+1).lt.0
+ .or.icp(iend+2*iend2+2*ipt(81)+i+1).gt.ipt(1))
+ call ErrMessage (125,1,ipt(29))
495 write(21,547) cname(icp(iend+2*iend2+2*ipt(81)+i))
+ ,icp(iend+2*iend2+2*ipt(81)+i+1)
end if
iadd = 1
iadd2 = 1
if(ipt(69).gt.0) then
ii = 0
do 600 i = 1,ipt(3)
if((icp(i).eq.icp(iend+iadd)).and.(ii.le.ipt(69))) then
ii = ii + 1
icp(iend+iend2+iadd) = (i-1)*ipt(1)
iadd = iadd + 1
end if
600 continue
end if
if(ipt(81).gt.0) then
ii = 0
do 601 i = 1,ipt(3)
if((icp(i).eq.icp(iend+2*iend2+iadd2))
+ .and.(ii.le.ipt(81))) then
ii = ii + 1
icp(iend+2*iend2+iadd2+1)
= (i-1)*ipt(1)+icp(iend+2*iend2+iadd2+1)
iadd2 = iadd2 + 2
end if
601 continue
end if
if(ipt(70).gt.0) then
ii = 0
do 605 i = 1+ipt(3),ipt(3)+ipt(4)
if((icp(i).eq.icp(iend+iadd)).and.
+ (ii-ipt(3).le.ipt(70))) then
ii = ii + 1
icp(iend+iend2+iadd) = (i-1)*ipt(1)
iadd = iadd + 1
end if
605 continue
end if
if(ipt(82).gt.0) then
ii = 0
do 606 i = 1+ipt(3),ipt(3)+ipt(4)
if((icp(i).eq.icp(iend+2*iend2+iadd2))
+ .and.(ii-ipt(3).le.ipt(82))) then
ii = ii + 1
icp(iend+2*iend2+iadd2+1)
= (i-1)*ipt(1)+icp(iend+2*iend2+iadd2+1)
iadd2 = iadd2 + 2
end if
606 continue
end if
if(ipt(71).gt.0) then
ii = 0
do 610 i = 1+ipt(3)+ipt(4),ipt(3)+ipt(4)+ipt(5)
if((icp(i).eq.icp(iend+iadd)).and.
+ (ii-ipt(3)-ipt(4).le.ipt(71))) then
ii = ii + 1
icp(iend+iend2+iadd) = (i-1)*ipt(1)
iadd = iadd + 1
end if
610 continue
end if
if(ipt(72).gt.0) then
ii = 0
do 615 i = 1+ipt(3)+ipt(4)+ipt(5),ipt(3)+ipt(4)+ipt(5)+ipt(6)
if((icp(i).eq.icp(iend+iadd)).and.
+ (ii-ipt(3)-ipt(4)-ipt(5).le.ipt(72))) then
ii = ii + 1
icp(iend+iend2+iadd) = (i-1)*ipt(1)
iadd = iadd + 1
end if
615 continue

```

```

end if
if(ipt(73).gt.0) then
  ii = 0
  do 620 i = 1+ipt(3)+ipt(4)+ipt(5)+ipt(6),
+   ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(7)
+   if((icp(i).eq.icp(iend+iadd)).and.
+   ((i-ipt(3)-ipt(4)-ipt(5)-ipt(6)).le.ipt(73))) then
    ii = ii + 1
    icp(iend+iend2+iadd) = (i-1)*ipt(1)
    iadd = iadd + 1
  end if
620 continue
end if
c
c— Error check: contour file must be opened if any Icon is true.
do 625 i = 1, 17
625 if (Icon(i).and..not.lctrl(23)) call ErrMessage (106,0,ipt(29))
c
c— Error check: time series file must be opened if any lplt is true.
do 630 i = 1, 2
630 if (lplt(i).and..not.lctrl(15)) call ErrMessage (123,0,ipt(29))
return
c
c— formats
500 format (76('**')/76('**')/
+ ' MISER'/
+ /
+ ' Michigan Soil vapor Extraction Remediation model'
+ //
+ ' Project directed by: L.M. Abriola'
+ //
+ ' Written by: J.R. Lang and K.M. Rathfelder'
+ //
+ ' Beta Version 1.0; May 1, 1996'
+ //
+ 76('**')/76('**'))
501 format ('INPUT AND OUTPUT FILES ',53('=')/
+ ' Main input file; control and field data: (unit 11) = ',a20/
+ ' Input file #2; ICs and BCs: (unit 13) = ',a20/
+ ' Error and warning message input data: (unit 14) = ',a20/
+ ' Main output file: (unit 21) = ',a20/
+ ' Error messages: (unit 22) = ',a20/
+ ' Performance Output: (unit 23) = ',a20/
+ ' Contour plot data: (unit 24) = ',a20/
+ ' Material balance information: (unit 25) = ',a20/
+ ' Time series plot data: (unit 26) = ',a20/
+ ' Restart data: (unit 27) = ',a20)
502 format ('GENERAL MODEL CONTROL OPTIONS ',46('=')/
+ ' Domain configuration (0=xz,1=rz): ipt(27) = ',i5/
+ ' Gravitational constant (m/s^2)-horizontal:t(21) = ',e12.4/
+ ' Gravitational constant (m/s^2)-vertical: t(22) = ',e12.4/
+ ' Solve transient phase balance: lctrl(1) = ',i5/
+ ' Skip ipt(85) time steps for phase blnce: ipt(85) = ',i5/
+ ' Solve transient component balance: lctrl(2) = ',i5/
+ ' Solve NAPL equations: lctrl(24) = ',i5/
+ ' Solve solid phase equations: lctrl(25) = ',i5/
+ ' Solve biophase equations: lctrl(3) = ',i5/
+ ' Lump the phase balance mass matrix: lctrl(7) = ',i5/
+ ' Lump the component balance mass matrix: lctrl(8) = ',i5/
+ ' Include phase mass exchange in flow eqs: lctrl(14) = ',i5/
+ ' Calculate element dimensionless numbers: lctrl(4) = ',i5)
503 format ('TIME STEP/ITERATION CONTROL OPTIONS ',40('=')/
+ ' Initial simulation time (sec): t(1) = ',e12.4/
+ ' Final simulation time (sec): t(2) = ',e12.4/
+ ' Time weighting factor: t(10) = ',e12.4/
+ ' Maximum number of time steps: ipt(30) = ',i12 /
+ ' Convergence tolerance for pressure: t(13) = ',e12.4/
+ ' Convergence tolerance for concentration: t(14) = ',e12.4/
+ ' Convergence tolerance for NAPL saturation:t(15) = ',e12.4/
+ ' Convergence tolerance for immobile phases:t(16) = ',e12.4/
+ ' Initial time step: t(3) = ',e12.4/
+ ' Minimum time step: t(4) = ',e12.4/
+ ' Maximum time step: t(5) = ',e12.4/
+ ' Maximum phase balance iterations: ipt(31) = ',i5 /
+ ' Maximum component balance iterations: ipt(32) = ',i5 /
+ ' Maximum NAPL saturation iterations: ipt(33) = ',i5 /
+ ' Minimum phase balance iterations: ipt(34) = ',i5 /
+ ' Minimum component balance iterations: ipt(35) = ',i5 /
+ ' Time step multiplier for amplification: t(6) = ',e12.4/
+ ' Time step multiplier for reduction: t(7) = ',e12.4)
504 format('GRID INFORMATION ',59('=')/
+ ' Generate grid: igrid = ',i5 /
+ ' 0 = don't generate the grid;/
+ ' 1 = generate a union jack grid;/
+ ' 2 = generate a herring bone grid./
+ ' Maximum dimension for number of elements: nelmx = ',i5 /
+ ' Actual number of elements: ipt(0) = ',i5 /
+ ' Maximum dimension for number of nodes: nnmx = ',i5 /
+ ' Actual number of nodes: ipt(1) = ',i5 /
+ ' Maximum dimension for the number of /
+ ' material property blocks: nmblok = ',i5 /
+ ' Number of material property blocks: ipt(26) = ',i5 /
+ ' Maximum dimension for the number of /
+ ' nodal variables in stacked storg: nnstk = ',i5 /
+ ' Actual number of stacked nodal variables: ipt(2) = ',i5 /
+ ' Full bandwidth of transport matrix: nbw(1) = ',i5 /
+ ' Full bandwidth of flow matrix: nbw(2) = ',i5 )
505 format(
+ ' Nodal Coordinates in the cross-sectional (xz) domain:/'
+ ' Node Number X-Location Z-Location' )
506 format(
+ ' Nodal Coordinates in the radial-vertical (rz) domain:/'
+ ' Node Number R-Location Z-Location' )
507 format(6x,i5,9x,e14.5,6x,e14.5)
508 format(
+ ' Element Information:/'
+ ' Element Number Nodal Incidence List ',
+ ' Material Block Area (m^2)')
509 format(6x,i5,12x,3i5,8x,i5,6x,e14.4)
510 format('COMPONENT CHEMICAL PROPERTY DATA ',43('=')/
+ ' Number of organic components: ipt(15) = ',i5 )
511 format(
+ ' Component chemical property data for: ',a/
+ ' Component number: = ',i5/
+ ' Molecular weight (g/mole): = ',e12.4/
+ ' Vapor pressure (atm): = ',e12.4/
+ ' Vapor viscosity (cPoise): = ',e12.4/
+ ' Liquid density (g/l): = ',e12.4/
+ ' Gas diffusivity (cm^2/s): = ',e12.4/
+ ' Aqueous diffusivity (cm^2/s): = ',e12.4/
+ ' Henry's Law constant (atm l/g): = ',e12.4/
+ ' Aqueous solubility (g/l): = ',e12.4)
512 format('MASS EXCHANGE COEFFICIENTS ',49('=')/2x,
+ ' Component Aqueous/gas Aqueous/NAPL Gas/NAPL'
+ ' Bio/Aqueous Solid/Aqueous')
513 format(2x,a10,5e12.4)
582 format('MINIMUM DEVIATION FROM EQUILIBRIUM ',41('=')/2x,
+ ' Component Aqueous/gas Aqueous/NAPL Gas/NAPL'
+ ' Bio/Aqueous Solid/Aqueous')
514 format('MATERIAL BLOCK PROPERTY DATA ',47('='))
515 format(
+ ' Material Block Number: ',i5/
+ ' Porosity: = ',e12.4/
+ ' Horizontal permeability (m^2): = ',e12.4/
+ ' Vertical permeability (m^2): = ',e12.4/
+ ' Bulk soil density (gm/cm^3): = ',e12.4/
+ ' Organic carbon content: = ',e12.4/
+ ' Residual water saturation: = ',e12.4/
+ ' van Genuchten n for air/water retention data: = ',e12.4/
+ ' van Genuchten alpha value (1/Pa): = ',e12.4/
+ ' longitudinal dispersivity (m): = ',e12.4/
+ ' transverse dispersivity (m): = ',e12.4)
580 format('HYDRODYNAMIC DISPERSION TENSOR INFORMA-
TION ',33('=')/
+ ' Compute hydrodynamic dispersion tensor: lctrl(21) = ',i5)
581 format(
+ ' Input values of hydrodynamic dispersion:/'
+ ' component phase',7x,'d11',9x,'d12',9x,'d21',9x,'d22')
585 format(a12,a11,4e12.4)
589 format('SORPTION PARAMETERS ',56('='))
516 format(
+ ' Isotherm parameters')
517 format(
+ ' Material Block Number ',i5)
518 format(
+ ' Kf (micrograms/gram-solid) for: ',a10, = ',e12.4/

```

```

+ ' m parameter (unitless) for: 'a10,' =',e12.4)
519 format(
+ ' Two Compartment Model is in use'
+ ' Slow to fast compartment Kf multiplier: =',e12.4/
+ ' Slow to fast compartment m multiplier: =',e12.4/
+ ' Fast compartment fraction of solid phase: =',e12.4/
+ ' Slow to fast compartment exchange multiplier: =',e12.4)
588 format(
+ ' Retardation is considered for the following components'
590 format(
+ ' Sorption is not considered for this simulation')
574 format(
+ ' Retardation factor for: a10 ; =',e12.4)
520 format('/ BIOLOGICAL PARAMETERS ',54('='))
+ ' Number of degradable organic components: ipt(17) =',i5/
+ ' Steady state biomass is assumed: lctrl(17) =',i5/
+ ' Biophase mass transfer is at equilibrium: lctrl(16) =',i5/
+ ' Degradation kinetics: ipt(39) =',i5/
+ ' 1 = standard Monod kinetics'
+ ' 2 = Monod kinetics with substrate inhibition'
+ ' 3 = Monod kinetics with lumped substrate inhibition'
+ ' 4 = Monod kinetics with saturation dependency'
+ ' 5 = monod kinetics with saturation dependency and'
+ ' substrate inhibition'
+ ' Initial uniform biomass (g/l): xinit =',e12.4/
+ ' Minimum biomass (g/l): xbmin =',e12.4/
+ ' Maximum biomass (g/l): xbmax =',e12.4/
+ ' Decay coefficient (1/sec): kd =',e12.4/
+ ' Delay period for bioreaction (sec): t(11) =',e12.4)
521 format(
+ ' Component biological parameter data for: ',a/
+ ' Oxygen Use coefficient (gm O2/gm substrate): =',e12.4/
+ ' Nutrient Use coefficient (gm n/gm substrate): =',e12.4/
+ ' Maximum substrate use rate (gm/gm cell sec): =',e12.4/
+ ' Half saturation constant (gm substrate/l): =',e12.4/
+ ' Yield coefficient (gm cell/gm substrate): =',e12.4/
+ ' Inhibition multiplier: =',e12.4)
522 format('/ PHASE PARAMETERS AND COMPOSITION ',43('='))
+ ' Water phase viscosity (cPoise): wvis =',e12.4/
+ ' Include Klinkenberg effect: lctrl(20) =',i5/
+ ' Klinkenberg parameter (compute if b=<0): b =',e12.4)
523 format(2x,a,' Phase has ',i2,' components listed below:')
524 format(6x,7a10)
525 format('/ TEMPERATURE DISTRIBUTION ',51('='))
+ ' Constant temperature distribution: lctrl(10) =',i5)
526 format(
+ ' Uniform temperature (degree C): ctemp =',e12.4)
527 format(
+ ' Temperature distribution (node number; temperature)')
528 format(4(i5,' ',e11.5,1x))
570 format('/ Temperature dependent parameters for: ',a10/
+ ' depth',7x,'cvp',10x,'cvvis',8x,'chen',9x,'casol',8x,'umax')
571 format(6(e12.5,1x))
572 format(/
+ ' Temperature dependent decay coefficient (depth; kd)')
573 format(3(e12.5,' ',e12.5,1x))
529 format('/ PRINTING INTERVAL VARIABLES ',48('='))
+ ' The print interval is set by time steps: ipt(25) =',i5)
530 format('/ PRINTING INTERVAL VARIABLES ',48('='))
+ ' The print interval is set by time units: t(12) =',e12.4)
531 format('/ PRINTING CONTROL VARIABLES ',49('='))
+ ' Generate output for the selected items'
+ ' Output in molar form: lprnt(8) =',i5 /
+ ' Gas phase pressure: lprnt(9) =',i5 /
+ ' Aqueous phase pressure: lprnt(10) =',i5 /
+ ' Gas/aqueous capillary pressure: lprnt(11) =',i5 /
+ ' Gas phase density: lprnt(12) =',i5 /
+ ' Aqueous phase density: lprnt(13) =',i5 /
+ ' NAPL density: lprnt(14) =',i5 )
532 format (
+ ' Gas phase components: lprnt(15) =',i5 )

```

```

533 format (
+ ' Aqueous phase components: lprnt(16) =',i5 )
534 format (
+ ' NAPL components: lprnt(17) =',i5 )
535 format (
+ ' Solid phase loadings: lprnt(18) =',i5 )
536 format (
+ ' Bio-phase components: lprnt(19) =',i5 /
+ ' Total organic soil concentration lprnt(29) =',i5 )
537 format (
+ ' Gas phase saturation: lprnt(20) =',i5 /
+ ' Aqueous phase saturation: lprnt(21) =',i5 /
+ ' NAPL saturation: lprnt(22) =',i5 /
+ ' Gas phase Darcy velocity: lprnt(23) =',i5 /
+ ' Aqueous phase Darcy velocity: lprnt(24) =',i5 )
538 format (' Generate contour files for the selected items' /
+ ' Contour in molar form: lcon(1) =',i5 /
+ ' Gas phase pressure: lcon(2) =',i5 /
+ ' Aqueous phase pressure: lcon(3) =',i5 /
+ ' Gas/aqueous capillary pressure: lcon(4) =',i5 /
+ ' Gas phase density: lcon(5) =',i5 /
+ ' Aqueous phase density: lcon(6) =',i5 /
+ ' NAPL density: lcon(7) =',i5 )
539 format (
+ ' Gas phase components: lcon(8) =',i5 )
540 format (
+ ' Aqueous phase components: lcon(9) =',i5 )
541 format (
+ ' NAPL components: lcon(10) =',i5 )
542 format (
+ ' Solid phase loadings: lcon(11) =',i5 )
543 format (
+ ' Bio-phase components: lcon(12) =',i5 /
+ ' Total organic soil concentration lcon(18) =',i5 )
544 format (
+ ' Gas phase saturation: lcon(13) =',i5 /
+ ' Aqueous phase saturation: lcon(14) =',i5 /
+ ' NAPL saturation: lcon(15) =',i5 /
+ ' Gas phase Darcy velocity: lcon(16) =',i5 /
+ ' Aqueous phase Darcy velocity: lcon(17) =',i5 )
545 format(5x,7a10)
546 format(' Generate time series files for the selected items' /
+ ' Gas phase components: lplt(1) =',i5 )
547 format(
+ ' ',a10,' at node number',i10 )
548 format(
+ ' Aqueous phase components: lplt(2) =',i5 )
549 format (
+ ' Mass balance print interval in time steps: ipt(83) =',i5)
550 format (
+ ' Mass balance print interval in time units: t(27) =',e12.4)
551 format (
+ ' Time series print interval in time steps: ipt(84) =',i5)
552 format (
+ ' Time series print interval in time units: t(28) =',e12.4)
553 format (
+ ' Material balance output in report form')
554 format (
+ ' Material balance output in time series form')
555 format (' OUTPUT FILES FOR COMPONENT MASS BALANCE ',37('='))
556 format(
+ ' ',a10,' (unit',i2,') =',a20)
557 format (
+ ' Time series output in mole fractions')
558 format (
+ ' Time series output in concentrations')
559 format(
+ ' Boundary nodes of the generated grid:')
560 format(5x,a/50(10x,10i6/))
end

```


Subroutine - input2.f

```

C-----
C
C INPUT2.f - Subroutine which reads the initial and boundary
C conditions from device 13 and writes to device 21.
C This routine is written so that data2 can serve as
C a restart file generated by an option in data1.
C-----
subroutine INPUT2
include 'dimen.inc'
character*20 infile(4),outpre,outfile(8+ncmp)
character*10 cname(ncmp)
common /cb1/ matel(nelmx),node1(nel3),nodept(nnmx),nelpt(nel3),
+   matpt(nn6)
common /cb1c/ xnode(nnmx),znode(nnmx),rbar(nelmx),area(nelmx)
common /cb2/ p(nn3)
common /cb2c/ q(nel4)
common /cb2d/ pmw0(nnmx),den0(nnmx)
common /cb3/ sat(nnstk3)
common /cb5a/ bphi(nmblk),bpermh(nmblk),bpermv(nmblk)
common /cb5b/ bvga(nmblk),bvga(nmblk),bvgn(nmblk),bsrw(nmblk)
common /cb6b/ por(nelmx),srw(nnstk)
common /cb6c/ temp(nnmx)
common /cb6d/ dtemp(nzmax6),idepth(nnmx)
common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
+   chen(ncmp),casol(ncmp),cmdif(ncmp2)
common /cb8/ vis(nnmx),pmw(nn3)
common /cb9/ xmf(nmf)
common /cb10/ den(nn6)
common /cb11/ pex(nns10),rxnp(nn2)
common /cb30/ ibc(nnmx)
common /cb31/ source(nn2)
common /cb64/ bok(nbcmp),bom(nbcmp),krtid(ncmp)
common /cb84/ ibcxmf(nmbc),bcxmf(nmbc),dfxmf(nmbc)
common /cb86/ str1(ncmpp5),str0(ncmpp5),cmf(ncmpp5),csink(ncmpp5)
+   ,cwsink(ncmpp5),csflux(ncmpp5),cmass1(ncmpp5),cmass0(ncmpp5)
+   ,cphex(ncmpp5),crsink(ncmpp5),tmass1,tmass0
common /cb90/ infile,outpre,outfile
common /cb91/ cname

C-----
C Dimension local arrays.
C
C dimension omfel(ncmp),soel(nelmx),iel(nelmx)
C
C-----
C Zero vectors and define local pointers.
C
if (ipt(29).ne.0) write (ipt(29),*)
+   'Reading from Input File ',infile(2)
lcsat=.false.
lprnt(3)=.true.
do 80 i= 1,(ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(7))*ipt(1)
80 xmf(i)=zer0
do 81 i= 1,(ipt(3)+ipt(4))*ipt(1)
   ibcxmf(i)=0
   dfxmf(i)=zer0
81 bcxmf(i)=zer0
do 85 i= 1,3*ipt(2)
   sat(i) = zer0
85 continue
ipt02=2*ipt(0)
ipt03=3*ipt(0)
ipt12=2*ipt(1)
ipt13=3*ipt(1)
ipt22=2*ipt(2)

c
c===== RESTART INFORMATION =====
c
if (ipt(29).ne.0) write (ipt(29),*)
+   'Reading Restart Information'

c
c----- Read two logical variables. Lctrl(26) indicates if this run
C----- is a restart and lctrl(32) indicates whether the run is a
C----- continuation of the previous run (lctrl(32)=t) or
C----- is a new run using the previous run as initial conditions

```

```

C----- (setting lctrl(32)=f resets the cumulative mass balance).
C
lctrl(32) = .false.
call commnt(13,21)
read(13,*) lctrl(26), lctrl(32)
if (lctrl(26)) then
  call commnt(13,21)
  read(13,*) infile(4)
  open (28,file=infile(4),status='unknown')
else
  infile(4) = 'none opened'
endif
write(21,550) lctrl(26),infile(4),lctrl(32)

c
c===== INITIAL PRESSURE INFORMATION =====
c
if (ipt(29).ne.0) write (ipt(29),*)
+   'Reading Initial Pressure Information'

c----- Read an integer control variable (ipt(75)) indicating how the
c----- initial pressure distribution is to be input:
c   1 = Compute initial pressure distribution assuming Pg = 1 atm
c       and Pa is hydrostatic referenced to atmospheric pressure
c       at the water table. The water table is assumed to be flat;
c   2 = Input gas and aqueous pressures at all nodes;
c   3 = Assume gas and aqueous pressures are atmospheric and input
c       the Darcy velocity and saturations for both the aqueous
c       and gas phases. This option can only be used if the
c       phase mass balance equations are not solved.
c Note: ipt(75)=3 is not a general option but is retained for
c       for column simulations.
c----- Error Check: ipt(75)=(1-3);
c----- Write header information.
write (21,499)
call commnt(13,21)
read (13,*) ipt(75)
if (ipt(75).lt.1 .or. ipt(75).gt.3) call ErrorMessage (48,0,ipt(29))
write (21,500) ipt(75)

c
c----- If ipt(75) = 1, then read the depth to the water table (m).
c----- Compute the initial pressure distribution.
c----- All pressures are gauge pressures in Pascals.
c----- Write to output file.
if (ipt(75) .eq. 1) Then
  if (ipt(29).ne.0) write (ipt(29),*)
  +   'Computing Initial Pressure Distribution'
  call commnt(13,21)
  read(13,*) wtdpth

c
c----- Estimate a gas phase density. Do not consider NAPL effects.
Do 86 i = 1,ipt(1)
  gmw = zer0
  vpn = rone
  do 87 ic = ipt(15)+1, ipt(15)+2
    if(cvp(ic).gt.zer0) then
      if(lctrl(10)) then
        cvpt = cvp(ic)
      else
        cvpt = cvp(ic) + dtemp((ic-1)*ipt(89)+idepth(i))
      end if
      gmw = gmw + cvpt*cmw(ic)/patm
      vpn = vpn - cvpt/patm
    end if
  end if
87 continue
  gmw = gmw + vpn * cmw(ipt(15)+3)
  gmnden = gmw * patm / r
  gden = gmnden / temp(i)

C
C----- Use Hubbert's Potential to set initial gas phase pressure.
C
pwtble = patm * (dexp(wtdpth*gden*t(22)/patm)-rone)
if(znode(i).ge.wtdpth) then
  p(ipt(1)+i) = pwtble
  +   + cden(ipt(15)+1) * t(22) * (znode(i)-wtdpth)

```

```

        p(i) = p(ipt(1)+i)
        else if(znode(i).lt.wtdpth) then
            p(i) = patm * (dexp(znode(i)*gden*t(22)/patm)-rone)
            p(ipt(1)+i) = cden(ipt(15)+1) * t(22) * (znode(i)-wtdpth)
+           + pwtble
        +
        end if
86    continue
        write (21,501) wtdpth
    Endif

c
c— Otherwise if ipt(75) = 2, then read in initial water and gas phase
c— at all nodes.
c— One node per line, followed by water and air pressure.
c— Output all user input pressures (Pa gauge).
    If (ipt(75).eq. 2) Then
        call commnt(13,21)
        Read (13,*) nd
        Backspace 13
        If (nd.lt. 0) Then
            Read (13,*) nd, paq,pgas
            Do i = 1,ipt(1)
                p(i) = pgas
                p(ipt(1)+i) = paq
            EndDo
        Else
            Do i = 1,ipt(1)
                Read (13,*) nd,p(ipt(1)+nd),p(nd)
            EndDo
        Endif
        write (21,502)
        write (21,503) (nd,p(ipt(1)+nd),p(nd),nd=1,ipt(1))
    Endif

c
c— For ipt(75)=3, set pressures to atmospheric conditions
    If (ipt(75).eq. 3) Then
        Do i = 1,ipt(1)
            p(i) = zcr0
            p(ipt(1)+i) = zcr0
        EndDo
    Endif

c
c— Compute initial capillary pressures.
    Do i = 1,ipt(1)
        p(ipt(12)+i) = p(i) - p(ipt(1)+i)
    EndDo

c
c===== INITIAL VELOCITY INFORMATION =====
c
    if (ipt(29).ne.0) write (ipt(29),*)
+       'Reading Darcy Velocity Computation Information'
c
c— Element or nodal Darcy velocities. Read a logical variable
c— indicating if velocities are discretized on an elemental or nodal
c— basis.
    call commnt (13,21)
    read (13,*) lctrl(18)
    write (21,504) lctrl(18)

c
c— Darcy velocity computation approach:
c— If lctrl(1) is true then Darcy velocities are determined from
c— transient pressure distributions. Write message.
    If (lctrl(1)) Then
        write (21,505)
    Endif

c
c— If lctrl(1) is false then the Darcy velocity distribution is
c— constant. Read a logical variable indicating if steady state Darcy
c— velocities are computed from the pressure field.
    lessv = .false.
    If (.not. lctrl(1)) Then
        call commnt (13,21)
        read (13,*) lessv
    Endif

c
c— Compute the SS Darcy velocities from input pressures when
c— lctrl(1)=F and lessv = true
    If (.not.lctrl(1) .and. lessv) Then
        write (21,506)

```

```

        call VEL
    Endif

c
c— Read in SS Darcy velocities when lctrl(1)=F and lessv=false.
c— First determine if Darcy velocity components are uniform. If yes,
c— then read only the 4 components, otherwise read the 4 components
c— for all nodes or elements depending on the selected computation
c— method.
    lcv = .false.
    If (.not.lctrl(1) .and. .not.lessv) Then
        write (21,507)
        call commnt (13,21)
        read (13,*) lcv
        call commnt (13,21)
        If (lcv) Then
            read (13,*) qgx,qgz,qax,qaz
            If (lctrl(18)) Then
                Do i = 1,ipt(1)
                    q(i) = qgx
                    q(i+ipt(1)) = qgz
                    q(i+ipt(2)) = qax
                    q(i+ipt(3)) = qaz
                EndDo
            Else
                Do i = 1,ipt(0)
                    q(i) = qgx
                    q(i+ipt(0)) = qgz
                    q(i+ipt(02)) = qax
                    q(i+ipt(03)) = qaz
                EndDo
            Endif
        Endif
        Else
            If (lctrl(18)) Then
                Do i = 1,ipt(1)
                    read (13,*) i,q(i),q(i+ipt(1)),q(i+ipt(2)),q(i+ipt(3))
                EndDo
            Else
                Do i = 1,ipt(0)
                    read (13,*) i,q(i),q(i+ipt(0)),q(i+ipt(02)),q(i+ipt(03))
                EndDo
            Endif
        Endif
    Endif

c
c— Write the steady state velocity components (m/sec).
    If (.not. lctrl(1)) Then
        If (lcv) Then
            write (21,508) qgx,qgz,qax,qaz
        Else
            If (lctrl(18)) Then
                write (21,509)
                write (21,511) (i,q(i),q(i+ipt(1)),q(i+ipt(2)),
+                 q(i+ipt(3)),i=1,ipt(1))
            Else
                write (21,510)
                write (21,511) (i,q(i),q(i+ipt(0)),q(i+ipt(02)),
+                 q(i+ipt(03)),i=1,ipt(0))
            Endif
        Endif
    Endif

c
c— Read mobile phase saturations if ipt(75)=3.
c— Note: This is not a general option but is retained for column
c— simulations. Saturations are never stacked for this case.
c— lcsat indicates uniform mobile phase saturations (true).
    If (.not.lctrl(1) .and. ipt(75).eq.3) Then
        call commnt (13,21)
        read(13,*) lcsat
        call commnt (13,21)
        If (lcsat) Then
            read(13,*) sat(1),sat(1+ipt(1))
            Do i = 2,ipt(1)
                sat(i) = sat(1)
                sat(i+ipt(1)) = sat(1+ipt(1))
            EndDo
        Else
            Do i = 1, ipt(1)
                read(13,*) ii,sat(ii),sat(ii+ipt(1))

```

```

EndDo
EndIf
EndIf
c
===== NAPL SATURATION AND COMPOSITION =====
c
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Initial NAPL Saturation and Composition Data'
c
c— Read the number of elements containing NAPL.
c— A number less than zero indicates that a NAPL saturation is
c— uniform and contained in all elements
c— Error check: inoel<=ipt(0)
c— Print header
  call commnt (13,21)
  read (13,*) inoel
  If (inoel .gt. ipt(0)) call ErrorMessage (49,0,ipt(29))
  write (21,512)
c
c— If no NAPL present, print message.
  If (inoel .eq. 0) write (21,513)
c
c— Error check: lctrl(24) must be true for inoel nonzero and vice
c— versa.
  if(.not.lctrl(24) .and. inoel.ne.0) call ErrorMessage (82,0,ipt(29))
  if(lctrl(24) .and. inoel.eq.0) call ErrorMessage (105,0,ipt(29))
c
c— If inoel<0 then read and distribute uniform NAPL sat and mole
c— fractions at all nodes.
  If (inoel .lt. 0) Then
    call commnt (13,21)
    read (13,*) soel(1),(omfel(j),j=1,ipt(15))
    If (soel(1).lt.zero .or. soel(1).gt.rone)
  + call ErrorMessage (51,0,ipt(29))
    sum = zer0
    Do ii = 1, ipt(15)
      sum = sum + omfel(ii)
    EndDo
    If (sum .ne. rone) call ErrorMessage (52,0,ipt(29))
    write (21,514)
    write (21,515) soel(1),(omfel(j),j=1,ipt(15))
    Do 146 i = 1, ipt(1)
      sat(ipt(2)*2+i) = soel(1)
      Do 146 ii = 1, ipt(15)
        iim1=(ii-1)*ipt(1)
        xmf(ipt(10)+iim1+i) = omfel(ii)
146 continue
  EndIf
c
c— If the number of elements with NAPL is greater than zero,
c— then for each element provide the following information:
c— (1) element number
c— (2) NAPL saturation
c— (3) mole fraction of each organic component. There must be
c— ipt(15) mole fractions specified for each element, and they
c— must sum to 1.
c— Error Checks: element number=(1,ipt(0)); So=(0-1); sum xi = 1.
c— Distribute the element saturation to each node in the element.
  If (inoel .gt. 0) Then
    call commnt (13,21)
    write (21,516) inoel
    write (21,529) (cname(icp(i+ipt(3)+ipt(4))),i=1,ipt(5))
    Do 130 i = 1, inoel
      read (13,*) iel(i)
      If (iel(i).lt.1 .or. iel(i).gt.ipt(0))
  + call ErrorMessage (50,0,ipt(29))
      backspace 13
      read (13,*) iel(i),soel(i),(omfel(j),j=1,ipt(15))
      If (soel(i).lt.zero .or. soel(i).gt.rone)
  + call ErrorMessage (51,0,ipt(29))
      sum = zer0
      Do ii = 1, ipt(15)
        sum = sum + omfel(ii)
      EndDo
      If (sum .ne. rone) call ErrorMessage (52,0,ipt(29))
      write (21,517) iel(i),soel(i),(omfel(j),j=1,ipt(15))
      i3=iel(i)*3
      iloc1 = ipt22 + nodept(nodel(i3-2)) + nelpt(i3-2)

```

```

      iloc2 = ipt22 + nodept(nodel(i3-1)) + nelpt(i3-1)
      iloc3 = ipt22 + nodept(nodel(i3)) + nelpt(i3)
      sat(iloc1) = 10.0d0 + sat(iloc1) + soel(i)
      sat(iloc2) = 10.0d0 + sat(iloc2) + soel(i)
      sat(iloc3) = 10.0d0 + sat(iloc3) + soel(i)
      Do ii=1, ipt(15)
        iim1=(ii-1)*ipt(1)
        iloc1 = ipt(10) + iim1 + nodel(i3-2)
        iloc2 = ipt(10) + iim1 + nodel(i3-1)
        iloc3 = ipt(10) + iim1 + nodel(i3)
        xmf(iloc1) = 10.0d0 + xmf(iloc1) + omfel(ii)
        xmf(iloc2) = 10.0d0 + xmf(iloc2) + omfel(ii)
        xmf(iloc3) = 10.0d0 + xmf(iloc3) + omfel(ii)
      EndDo
130 continue
c
c— Now adjust for the multiple entries in sat and xmf for nodes where
c— adjacent elements have nonzero organic saturations. Average
c— organic saturations over a given material
c— property block and mole fractions over the entire domain.
  Do i = 1, ipt(2)
    If (sat(2*ipt(2)+i).ge.10.0d0) then
      sat(2*ipt(2)+i) = sat(2*ipt(2)+i)/10.0d0
      xx=dint(sat(2*ipt(2)+i))
      sat(2*ipt(2)+i) = 10.0d0*(sat(2*ipt(2)+i)-xx)/xx
    End If
  EndDo
  do 145 i = 1, ipt(1)
    do 145 ii=1, ipt(15)
      iim1=(ii-1)*ipt(1)
      if(xmf(ipt(10)+iim1+i).ge.10.0d0) then
        xmf(ipt(10)+iim1+i)=xmf(ipt(10)+iim1+i)/10.0d0
        xx=dint(xmf(ipt(10)+iim1+i))
        xmf(ipt(10)+iim1+i)
  + =10.0d0*(xmf(ipt(10)+iim1+i)-xx)/xx
      end if
145 continue
  EndIf
c
c— Initialize porosity and residual saturation vectors.
  do 170 i = 1, ipt(2)
170 srw(i) = bsrw(matpt(i))
  do 171 i = 1, ipt(0)
171 por(i) = bphi(matel(i))
c
c— Compute initial gas and aqueous phase saturations.
  If (ipt(75) .ne. 3) Then
    call SATW
  EndIf
  If (ipt(75) .eq. 3) Then
    do 147 i = 1, ipt(2)
      if(ipt(3).gt.0) then
        if(sat(i).gt.sat(i+2*ipt(2))) then
          sat(i) = sat(i)-sat(i+2*ipt(2))
        else
          write(21,*) 'Respecify gas phase saturation'
          stop
          end if
        else
          if(sat(i+ipt(2)).gt.sat(i+2*ipt(2))) then
            sat(i+ipt(2)) = sat(i+ipt(2))-sat(i+2*ipt(2))
          else
            write(21,*) 'Respecify aqueous phase saturation'
            stop
            end if
          end if
147 continue
    EndIf
c
===== BLOCK Q: OXYGEN AND NUTRIENT INITIAL CONDI-
===== TIONS =====
c
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Gas, Aqueous, and Biophase Initial Conditions'
c
c— Read a logical variable indicating if the gas phase initial
c— conditions are uniform. Skip this input if oxygen is not
c— present in the gas phase.

```

```

write (21,518)
call commnt (13,21)
if (cvp(ipt(15)+2).ge.zer0) then
call commnt (13,21)
read (13,*) lunfx
c
c— Read the gas phase initial conditions for oxygen and nutrient.
c— Initial conditions are read as partial pressures (i.e. mole
c— fractions). Nutrient can only be present if oxygen is present.
If (lunfx) Then
call commnt (13,21)
if(cvp(ipt(15)+4).ge.zer0.and.lctrl(9)) then
read (13,*) xog, xng
write (21,519) xog
write (21,543) xng
c
c— Error Check: mole frac=(0-1)
If (xog.lt.zer0 .or. xog.gt.rone)
+ call ErrMessage (53,0,ipt(29))
+ If (xng.lt.zer0 .or. xng.gt.rone)
+ call ErrMessage (73,0,ipt(29))
else
read (13,*) xog
write (21,519) xog
c
c— Error Check: mole frac=(0-1)
If (xog.lt.zer0 .or. xog.gt.rone)
+ call ErrMessage (53,0,ipt(29))
end if
C
C— Assign gas phase initial conditions to the xmf vector.
if (cvp(ipt(15)+4).ge.zer0.and.lctrl(9)) then
ipto = ipt(3)-3
iptgo2=ipt(8)+ipto*ipt(1)
if (cvp(ipt(15)+4).ge. zer0) then
iptn = ipt(3)-1
iptgn2=ipt(8)+iptn*ipt(1)
do 155 i=1,ipt(1)
155 xmf(i+iptgn2)=xng
end if
else
ipto = ipt(3)-2
iptgo2=ipt(8)+ipto*ipt(1)
end if
do 156 i=1,ipt(1)
156 xmf(i+iptgo2)=xog
else
c
c— Read a vector of nonuniform initial gas phase mole fractions.
call commnt (13,21)
if (cvp(ipt(15)+4).ge.zer0.and.lctrl(9)) then
iptgo = ipt(3)-3
iptgo2=ipt(8)+iptgo*ipt(1)
if (cvp(ipt(15)+4).ge. zer0) then
iptn = ipt(3)-1
iptgn2=ipt(8)+iptn*ipt(1)
write (21,547)
do 157 ii=1,ipt(1)
read(13,*) i, xmf(i+iptgo2), xmf(i+iptgn2)
c
c— Error Check: mole frac=(0-1)
If (xmf(i+iptgo2).lt.zer0 .or. xmf(i+iptgo2)
+ .gt.rone) call ErrMessage (53,0,ipt(29))
+ If (xmf(i+iptgn2).lt.zer0 .or. xmf(i+iptgn2)
+ .gt.rone) call ErrMessage (73,0,ipt(29))
157 continue
else
write (21,520)
do 158 ii=1,ipt(1)
read(13,*) i, xmf(i+iptgo2)
c
c— Error Check: mole frac=(0-1)
If (xmf(i+iptgo2).lt.zer0 .or. xmf(i+iptgo2)
+ .gt.rone) call ErrMessage (53,0,ipt(29))
158 continue
end if
else
iptgo = ipt(3)-2

```

```

iptgo2=ipt(8)+iptgo*ipt(1)
write (21,520)
do 159 ii=1,ipt(1)
read(13,*) i, xmf(i+iptgo2)
c
c— Error Check: mole frac=(0-1)
If (xmf(i+iptgo2).lt.zer0 .or. xmf(i+iptgo2).gt.rone)
+ call ErrMessage (53,0,ipt(29))
159 continue
end if
end if
end if
c
c— Read a logical variable indicating if the aqueous phase initial
c— conditions are uniform. Skip this input if oxygen is not present
c— in the aqueous phase.
call commnt (13,21)
if (casol(ipt(15)+2).ge.0) then
call commnt (13,21)
read (13,*) lunfx
c
c— Read the aqueous phase initial conditions for oxygen and nutrient.
c— Initial conditions are read as concentrations (i.e. g/L). Nutrient
c— can only be present if oxygen is present.
If (lunfx) Then
call commnt (13,21)
if(lctrl(9)) then
read (13,*) xoa, xna
c
c— Convert xna into a mole fraction.
xna = xna*cmw(ipt(15)+1)/(cmw(ipt(15)+4)*cden(ipt(15)+1))
write (21,543) xna
c
c— Error Check: mole frac=(0-1)
If (xna.lt.zer0 .or. xna.gt.rone)
+ call ErrMessage (74,0,ipt(29))
else
read (13,*) xoa
end if
c
c— Convert xoa into a mole fraction.
xoa = xoa*cmw(ipt(15)+1)/(cmw(ipt(15)+2)*cden(ipt(15)+1))
write (21,544) xoa
c
c— Error Check: mole frac=(0-1)
If (xoa.lt.zer0 .or. xoa.gt.rone)
+ call ErrMessage (54,0,ipt(29))
C
C— Assign aqueous phase initial conditions to the xmf vector.
iptao = ipt(14)+1
iptao2=ipt(9)+iptao*ipt(1)
if (lctrl(9)) then
iptan = ipt(4)-1
iptan2=ipt(9)+iptan*ipt(1)
do 160 i=1,ipt(1)
160 xmf(i+iptan2)=xna
end if
do 161 i=1,ipt(1)
C
C— Initialize biophase if present (lctrl(3)=t). Oxygen is
C— always a component of the biophase. Nutrient is present
C— in the biophase if included (lctrl(9)=t).
C
if (lctrl(3))
+ xmf(ipt(12)+ipt(17)*ipt(1)+i)=xoa
if (lctrl(9))
+ xmf(ipt(12)+(ipt(17)+1)*ipt(1)+i)=xna
161 xmf(i+iptao2)=xoa
else
c
c— Read a vector of nonuniform initial aqueous component
c— concentrations (gm/l).
call commnt (13,21)
iptao = ipt(14)+1
iptao2=ipt(9)+iptao*ipt(1)
if (casol(ipt(15)+4).ge. zer0 .and. lctrl(9) ) then
iptan = ipt(4)-1
iptan2=ipt(9)+iptan*ipt(1)

```

```

write (21,548)
do 162 ii=1,ipt(1)
  read(13,*) i, xoa, xna
c
c— Convert xoa into a mole fraction.
  xoa = xoa*cmw(ipt(15)+1)/(cmw(ipt(15)+2)
+
  *cden(ipt(15)+1))
c
c— Error Check: mole frac=(0-1)
  If (xoa.lt.zer0 .or. xoa.gt.rone)
+
  call ErrMessage (54,0,ipt(29))
c
c— Convert xna into a mole fraction.
  xna = xna*cmw(ipt(15)+1)/(cmw(ipt(15)+4)
+
  *cden(ipt(15)+1))
c
c— Error Check: mole frac=(0-1)
  If (xna.lt.zer0 .or. xna.gt.rone)
+
  call ErrMessage (74,0,ipt(29))
C
C— Initialize biophase if present (lctrl(3)=t). Oxygen is
C— always a component of the biophase. Nutrient is present
C— in the biophase if included (lctrl(9)=t).
C
  if (lctrl(3))
+
  xmf(ipt(12)+ipt(17)*ipt(1)+i)=xoa
  if (lctrl(9))
+
  xmf(ipt(12)+(ipt(17)+1)*ipt(1)+i)=xna
162  xmf(i+iptao2) = xoa
    xmf(i+iptan2) = xna
  else
  write (21,546)
  do 163 ii=1,ipt(1)
    read(13,*) i, xoa
c
c— Convert xoa into a mole fraction.
  xoa = xoa*cmw(ipt(15)+1)/(cmw(ipt(15)+2)
+
  *cden(ipt(15)+1))
c
c— Error Check: mole frac=(0-1)
  If (xoa.lt.zer0 .or. xoa.gt.rone)
+
  call ErrMessage (54,0,ipt(29))
C
C— Initialize biophase oxygen mole fraction if needed. Oxygen is
C— always a component of the biophase. If nutrient is present
C— (lctrl(9)=t) it must be included in the biophase.
C
  if (lctrl(3))
+
  xmf(ipt(12)+ipt(17)*ipt(1)+i)=xoa
163  xmf(i+iptao2) = xoa
  end if
end if
end if
C
C— Now assign gas phase nitrogen mole fractions for calculation
C— of initial gas phase density without organic content. This
C— neglects water vapor presence
C
  if (ipt(3).gt.0) then
  if (lctrl(9)) then
  in = ipt(3) - 2
  else
  in = ipt(3) - 1
  end if
  ix = in * ipt(1)
  do 165 i=1,ipt(1)
  xmf(ipt(8)+ix+i) = rone
  do 165 ii=ipt(13),ipt(3)
  if (icp(ii).ne.icp(in+1)) then
  xmf(ipt(8)+ix+i) = xmf(ipt(8)+ix+i)
+
  - xmf(ipt(8)+(ii-1)*ipt(1)+i)
  end if
165  continue
  end if
C
C— Now assign aqueous phase water mole fractions for calculation

```

```

C— of initial aqueous phase density without organic content.
C
  if (ipt(4).gt.0) then
  ix=ipt(14)*ipt(1)
  do 166 i=1,ipt(1)
  xmf(ipt(9)+ix+i) = rone
  do 166 ii=ipt(14),ipt(4)
  if (icp(ipt(3)+ii).ne.icp(ipt(3)+ipt(14)+1))
+
  xmf(ipt(9)+ix+i) = xmf(ipt(9)+ix+i)
+
  - xmf(ipt(9)+(ii-1)*ipt(1)+i)
166  continue
  end if
C
C— Calculate the gas phase initial phase molecular weight and the
C— aqueous phase initial density for velocity calculations.
C
  lkeep = lctrl(22)
  lctrl(22) = .false.
  call MOLEWT
  lctrl(22) = lkeep
  do 167 i = 1, ipt(1)
  pmw0(i) = pmw(i)
167  den0(i) = den(4*ipt(1)+i)
C
C— Next calculate the equilibrium distribution of the organic
C— components. This initial distribution occurs only in elements
C— containing NAPL. Organic components are partitioned into the gas,
C— aqueous, solid, and biophases.
  igas = 0
  iaq = igas + ipt(3)
  inapl = iaq + ipt(4)
  isol = inapl + ipt(5)
  ibio = isol + ipt(6)
  do 164 ii = 1, ipt(15)
  inapl = inapl + 1
  if (ipt(3).gt.0).and.(icp(igas+1).eq.ii) igas=igas+1
  if (ipt(4).gt.0).and.(icp(iaq+1).eq.ii) iaq=iaq+1
  if (ipt(6).gt.0).and.(icp(isol+1).eq.ii) isol=isol+1
  if (ipt(7).gt.0).and.(icp(ibio+1).eq.ii) ibio=ibio+1
  ngas = (igas-1) * ipt(1)
  naq = (iaq-1) * ipt(1)
  nnapl = (inapl-1) * ipt(1)
  nsol = (isol-1) * ipt(1)
  nbio = (ibio-1) * ipt(1)
C
C— Compute organic equilibrium mole fractions when organic phase
C— is present.
  Do 164 i = 1, ipt(1)
  if (xmf(nnapl+i).gt.zer0) then
C
C— Compute gas phase equilibrium organic mole fractions. Allow
C— for the possibility that there are fewer organic components
C— in the gas phase than in the organic phase.
  if (ipt(3).gt.0) then
  if (cvp(icp(igas)).gt.zer0) then
  if (lctrl(10)) then
  cvpt = cvp(icp(igas))
  casolt = casol(icp(iaq))
  else
  cvpt = cvp(icp(igas))
+
  + dtemp((icp(igas)-1)*ipt(89)+idepth(i))
  casolt = casol(icp(iaq)) + dtemp
+
  ((icp(iaq)-1)*ipt(89)+3*ipt(88)+idepth(i))
  end if
+
  if (icp(igas).eq.ii) xmf(ngas+i)
  =xmf(nnapl+i)*cvpt/patm
  end if
  end if
C
C— Compute aqueous phase equilibrium organic mole fractions. Adjust
C— for the possibility that there are fewer organic species in the
C— aqueous phase than in the organic phase
  if (ipt(4).gt.0) then
  if (icp(iaq).eq.ii) xmf(naq+i)
+
  =xmf(nnapl+i)*casolt
  end if
C
C— Set initial bio-phase organic component mole fractions equal to

```

```

C— the aqueous phase mole fractions if needed.
  if (lctrl(3)) then
    if (lcp(ibio).eq.1) xmf(nbio+i) = xmf(naq+i)
  end if
end if
164 continue
C
C— Initialize the biomass.
  if (lctrl(3)) then
    do 82 i=ipt(12)+(ipt(7)-1)*ipt(1)+1, ipt(12)+ipt(7)*ipt(1)
82   xmf(i)=xinit
    end if
  c
  ===== RESTART VARIABLE INPUT =====
  c
  C— If lctrl(26) is true the run is a restart. Read the appropriate
  c— restart information here. Restart information is consistent with
  c— specified control options in other sections of the input. For
  c— example restarts can be for either the flow or transport alone,
  c— or use the flow restart information to generate a steady state
  c— flow field.
  if (lctrl(26)) then
    if (ipt(29).ne.0) write (ipt(29),*)
    + 'Reading from Restart File ',infile(4)
    rewind(28)
    call commnt(28,21)
    read(28,*) ipt0r, ipt1r, ipt2r, ipt3r, ipt4r, ipt5r, ipt6r, ipt7r
  c
  C— Error Check simulation parameters from restart file.
  if (ipt0r.ne.ipt(0)) call ErrorMessage (115,0, ipt(29))
  if (ipt1r.ne.ipt(1)) call ErrorMessage (116,0, ipt(29))
  if (ipt2r.ne.ipt(2)) call ErrorMessage (117,0, ipt(29))
  if (ipt3r.ne.ipt(3)) call ErrorMessage (118,0, ipt(29))
  if (ipt4r.ne.ipt(4)) call ErrorMessage (119,0, ipt(29))
  if (ipt5r.ne.ipt(5)) call ErrorMessage (120,0, ipt(29))
  if (ipt6r.ne.ipt(6)) call ErrorMessage (121,0, ipt(29))
  if (ipt7r.ne.ipt(7)) call ErrorMessage (122,0, ipt(29))
  c
  C— Read the current time step size, the final or maximum time of the
  c— simulation, the current simulation time, and the current number of
  c— iterations.
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Restart Time Step, etc. '
  call commnt(28,21)
  read(28,601) t(8), t(2), t(9), ipt(76)
  c
  C— Read the mass balance information.
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Restart Mass Balance Information '
  call commnt(28,21)
  iendmb = ipt(65)+5
  read(28,602) (str1(i), str0(i), cmf(i), csink(i), cwsink(i)
  + , crsink(i), csflux(i), cphex(i), i=1, iendmb)
  call commnt(28,21)
  iendmb = ipt(65)*5
  read(28,602) (cmass1(i), cmass0(i), i=1, iendmb)
  call commnt(28,21)
  read(28,*) tmass1, tmass0
  c
  C— Read the pressure vectors.
  if (lctrl(1).or.ipt(75).eq.2) then
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Restart Pressures '
  call commnt(28,21)
  read(28,*) iend1
  read(28,602) (p(i), i=1, iend1)
  c
  C— Read the saturation vectors.
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Restart Saturations '
  call commnt(28,21)
  read(28,*) iend2
  read(28,602) (sat(i), i=1, iend2)
  if (ipt(75).eq.2) call VEL
  end if
  c
  C— Read the mole fraction vectors.
  if (lctrl(2)) then

```

```

  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Restart Mole Fractions '
  call commnt(28,21)
  read(28,*) iend3
  read(28,602) (xmf(i), i=1, iend3)
  c
  C— Read the exchange vectors.
  if (ipt(29).ne.0) write (ipt(29),*)
  + 'Reading Restart Exchange Vectors '
  call commnt(28,21)
  read(28,*) iend4
  read(28,602) (pex(i+ipt(52)), i=1, iend4)
  end if
end if
  c
  ===== BOUNDARY CONDITION INFORMATION =====
  c
  if (ipt(29).ne.0) write (ipt(29),*) 'Reading Boundary Conditions '
  c
  C— Read nodes with a constant gas phase pressure equal to
  c— the initial pressures.
  C— Error Check node numbers.
  call commnt (13,21)
  read (13,*) itype1
  If (itype1 .gt. 0) Then
  Read (13,*) (ibc(i), i=1, itype1)
  Do i = 1, itype1
  if (ibc(i).lt.0 .or. ibc(i).gt.ipt(1))
  + call ErrorMessage (55,0, ipt(29))
  EndDo
  EndIf
  c
  C— Read nodes with a constant gas phase pressure different from the
  c— initial conditions. Update capillary pressure at those nodes.
  C— Error Check node numbers. Pressure is in Pa gauge.
  call commnt (13,21)
  read (13,*) ipt(18)
  ipt(18) = ipt(18) + itype1
  If (ipt(18)-itype1 .gt. 0) Then
  Do i = itype1+1, ipt(18)
  read (13,*) (ibc(i), tpress
  if (ibc(i).lt.0 .or. ibc(i).gt.ipt(1))
  + call ErrorMessage (55,0, ipt(29))
  p(ibc(i)) = tpress
  p(2*ipt(1)+ibc(i)) = p(ibc(i))-p(ipt(1)+ibc(i))
  EndDo
  EndIf
  c
  C— Read nodes with a constant aqueous phase pressure equal to
  c— the initial pressures.
  call commnt (13,21)
  read (13,*) itype1
  If (itype1 .gt. 0) Then
  read (13,*) (ibc(ipt(18)+i), i=1, itype1)
  Do i = 1, itype1
  ii = ipt(18) + i
  if (ibc(ii).lt.0 .or. ibc(ii).gt.ipt(1))
  + call ErrorMessage (56,0, ipt(29))
  EndDo
  EndIf
  c
  C— Read nodes with a constant aqueous phase pressure different from
  c— the initial conditions. Update capillary pressure at those nodes.
  C— Pressure is in Pa gauge.
  call commnt (13,21)
  read (13,*) ipt(19)
  ipt(19) = ipt(19) + itype1
  If (ipt(19)-itype1 .gt. 0) Then
  Do i = itype1+1, ipt(19)
  ii = ipt(18) + i
  read (13,*) (ibc(ii), tpress
  if (ibc(ii).lt.0 .or. ibc(ii).gt.ipt(1))
  + call ErrorMessage (56,0, ipt(29))
  p(ipt(1)+ibc(ii)) = tpress
  p(2*ipt(1)+ibc(ii)) =
  + p(ibc(ii))-p(ipt(1)+ibc(ii))
  EndDo
  EndIf

```

```

c
c— Gas phase component boundary conditions.
c (1) Read the number of nodes for which gas phase component boundary
c conditions are specified.
c (2) For each such node, read:
c (2a) the node number
c (2b) an integer variable indicating the boundary condition
c type for all gas phase components at the node.
c 1 = constant mole fraction
c 2 = constant diffusive flux
c 3 = mixed type (contact with a known fluid).
c (2c) the boundary condition values for each component in the
c gas phase. The values are listed in sequential order
c corresponding to the component numbers. Only components
c that are present in the gas phase are listed. Component
c boundary conditions are not provided for components which
c are excluded from the gas phase (i.e. negative vapor
c pressure). Two values are needed for each boundary node.
c These values are used as needed to specify the boundary
c condition.
c 1 = specified gas phase mole fraction (partial pressure)
c in contacting fluid. The partial pressures must sum
c to one (used for first type boundary).
c 2 = user supplied value of Dm/length (m/sec).
c— Error check node numbers and BC type
c— Impose first type BCs
call commnt (13,21)
read (13,*) ipt(20)
If (ipt(20) .gt. 0) Then
  error = .false.
  Do i = 1, ipt(20)
    ii = ipt(18) + ipt(19)
    read (13,*) ibc(ii+i),ibcxmf(ibc(ii+i))
    + , (bcxmf(ibc*ipt(1)+ibc(ii+i))
    + , dfxmf(ibc*ipt(1)+ibc(ii+i)),icbc=0,ipt(3)-1)
    if (ibc(ii+i).lt.0 .or. ibc(ii+i).gt.1)
    + call ErrorMessage (57,0,ipt(29))
    if (ibcxmf(ibc(ii+i)).lt.1 .or. ibcxmf(ibc(ii+i)).gt.3)
    + call ErrorMessage (58,0,ipt(29))
    if (lctr(9)) then
      ixmfnc = ipt(3)-1
    else
      ixmfnc = ipt(3)
    end if
    ixmfnc = (ixmfnc-1)*ipt(1)+ibc(ii+i)
    sum = zero
    do icbc = 0, ipt(3)-1
      ixmf = icbc*ipt(1)+ibc(ii+i)
c
c— Error check: xmf = (0,1).
  if (bcxmf(ixmf).lt.zer0 .or. bcxmf(ixmf).gt.rone )
  + call ErrorMessage (83,0,ipt(29))
  if (ibcxmf(ibc(ii+i)).eq.1.and.lprnt(6) ) lerror = .true.
  if (ibcxmf(ibc(ii+i)).eq.1) xmf(ixmf) = bcxmf(ixmf)
  if (icbc+1.ne.ixmfnc) sum = sum + bcxmf(ixmf)
  end do
c
c— Calculate master component boundary condition in mole fractions.
c
  bcxmf(ixmfnc) = rone - sum
c
c— Error check: sum of specified boundary concentrations must
c equal 1.
  if (sum+bcxmf(ixmfnc).ne.rone) call ErrorMessage (85,0,ipt(29))
  EndDo
  if (lerror) call ErrorMessage (127,1,ipt(29))
  lerror = .false.
EndIf
c
c— Aqueous phase component boundary conditions.
c (1) Read the number of nodes for which aqueous phase component
c boundary conditions are specified.
c (2) For each such node, read:
c (2a) the node number
c (2b) an integer variable indicating the boundary condition
c type for all aqueous phase components at the node.
c 1 = constant mole fraction
c 2 = constant diffusive flux

```

```

c
c 3 = mixed type (contact with a known fluid).
c (2c) two boundary conditions values for each component in the
c aqueous phase. The values are listed in sequential order
c corresponding to the component numbers. Only components
c that are present in the aqueous phase are listed.
c Component boundary conditions are not provided for
c components which are excluded from the aqueous phase
c (i.e. negative solubility). Two values are needed for
c each boundary node component. These values are used as
c needed to specify the boundary condition.
c 1 = specified aqueous phase concentration (g/l)
c in contacting fluid (used for first type boundary).
c 2 = user supplied value of Dm/length (m/sec).
c— Error check node numbers and BC type
c— Impose first type BCs
call commnt (13,21)
read (13,*) ipt(21)
If (ipt(21) .gt. 0) Then
  error = .false.
  ii = ipt(18) + ipt(19) + ipt(20)
  Do i = 1, ipt(21)
    read (13,*) ibc(ii+i),ibcxmf(ipt(1)+ibc(ii+i))
    + , (bcxmf((ipt(3)+icbc)*ipt(1)+ibc(ii+i))
    + , dfxmf(icbc*ipt(1)+ibc(ii+i)),icbc=0,ipt(4)-1)
    if (ibc(ii+i).lt.0 .or. ibc(ii+i).gt.1)
    + call ErrorMessage (59,0,ipt(29))
    if (ibcxmf(ipt(1)+ibc(ii+i)).lt.1 .or.
    + ibcxmf(ipt(1)+ibc(ii+i)).gt.3)
    + call ErrorMessage (60,0,ipt(29))
    sum = zero
    do icbc = 0, ipt(4) - 1
      ixmf = (ipt(3)+icbc)*ipt(1)+ibc(ii+i)
c
c— Convert specified boundary concentrations into mole fractions.
  if (icbc+1.ne.ipt(14)+1) then
    bcxmf(ixmf) = bcxmf(ixmf) * cmw(ipt(15)+1)
    + / (cmw(icp(ipt(3)+icbc+1)) * cden(ipt(15)+1))
    sum = sum + bcxmf(ixmf)
c
c— Error check: xmf = (0,1).
  if (bcxmf(ixmf).lt.zer0 .or. bcxmf(ixmf).gt.rone )
  + call ErrorMessage (84,0,ipt(29))
c
c— Error warning: no mass balance for first type nodes.
  if (ibcxmf(ipt(1)+ibc(ii+i)).eq.1.and.lprnt(6) )
  + lerror = .true.
c
c— Impose first type boundary condition.
  if (ibcxmf(ipt(1)+ibc(ii+i)) .eq. 1)
  + xmf(ixmf) = bcxmf(ixmf)
  end if
  end do
c
c— Calculate master component boundary condition in mole fractions.
c
  ixmfw = (ipt(3)+ipt(14))*ipt(1)+ibc(ii+i)
  bcxmf(ixmfw) = rone - sum
  if (ibcxmf(ipt(1)+ibc(ii+i)) .eq. 1)
  + xmf(ixmfw) = rone - sum
  EndDo
  if (lerror) call ErrorMessage (127,1,ipt(29))
  lerror = .false.
EndIf
c
c— Read nodes with a constant gas phase specific discharge.
c— Units of source are m**3/sec.
c— Error check node numbers
call commnt (13,21)
read (13,*) ipt(22)
If (ipt(22) .gt. 0) Then
  iptbc = ipt(18) + ipt(19) + ipt(20) + ipt(21)
  Do i = 1, ipt(22)
    read (13,*) ibc(iptbc+i),source(i)
    if (ibc(iptbc+i).lt.0 .or. ibc(iptbc+i).gt.1)
    + call ErrorMessage (63,0,ipt(29))
  + EndDo
  EndIf
c

```

```

c— Read nodes with a constant aqueous phase specific discharge.
c— Units of source are m**3/sec.
call commnt (13,21)
read (13,*) ipt(23)
If (ipt(23) .gt. 0) Then
  iptbc = ipt(18) + ipt(19) + ipt(20) + ipt(21) + ipt(22)
  Do i = 1, ipt(23)
    read (13,*) ibc(iptbc+i), source(ipt(22)+i)
    if (ibc(iptbc+i) .lt. 0 .or. ibc(iptbc+i) .gt. ipt(1))
      + call ErrMessage (64,0, ipt(29))
  EndDo
EndIf

c
c— Write boundary condition information.
write (21,521)
write (21,522) ipt(18) ! constant gas phase pressures
If (ipt(18) .gt. 0) Then
  write (21,523)
  write (21,524) (ibc(i), p(ibc(i))), i=1, ipt(18))
EndIf
write (21,525) ipt(19) ! constant aqueous phase pressures
If (ipt(19) .gt. 0) Then
  write (21,526)
  Do i = 1, ipt(19)
    ii = ipt(18) + i
    write (21,524) ibc(ii), p(ipt(1)+ibc(ii))
  EndDo
EndIf
write (21,527) ipt(20) ! gas phase component BCs
If (ipt(20) .gt. 0) Then
  write (21,528)
  write (21,529) (cname(icp(i)), i=1, ipt(3))
  Do i = 1, ipt(20)
    ii = ipt(18) + ipt(19)
    If (ipt(3) .le. 2) Then
      write (21,531) ibc(ii+i), ibcxmf(ibc(ii+i)),
      + (bcxmf(ibc(ii+i))+icbc*ipt(1))
      + ,dfxmf(ibc(ii+i))+icbc*ipt(1)), icbc=0, ipt(3)-1)
    Else
      write (21,531) ibc(ii+i), ibcxmf(ibc(ii+i)),
      + (bcxmf(ibc(ii+i))+icbc*ipt(1))
      + ,dfxmf(ibc(ii+i))+icbc*ipt(1)), icbc=0, 1)
    Do j = 1, (ipt(3)-1)/2
      jj = min(ipt(3)-1, (j+1)*2-1)
      write (21,532)
      + (bcxmf(ibc(ii+i))+icbc*ipt(1))
      + ,dfxmf(ibc(ii+i))+icbc*ipt(1)), icbc=j*2, jj)
    EndDo
  EndIf
EndIf
write (21,533) ipt(21) ! aqueous phase component BCs
If (ipt(21) .gt. 0) Then
  write (21,534)
  write (21,529) (cname(icp(i+ipt(3))), i=1, ipt(4))
  Do i = 1, ipt(21)
    ii = ipt(18) + ipt(19) + ipt(20)
    If (ipt(4) .le. 2) Then
      write (21,531) ibc(ii+i), ibcxmf(ipt(1)+ibc(ii+i)),
      + (bcxmf(ibc(ii+i))+ipt(1)*(icbc+ipt(3)))
      + ,dfxmf(ibc(ii+i))+ipt(1)*(icbc+ipt(3)))
      + ,icbc=0, ipt(4)-1)
    Else
      write (21,531) ibc(ii+i), ibcxmf(ipt(1)+ibc(ii+i)),
      + (bcxmf(ibc(ii+i))+ipt(1)*(icbc+ipt(3)))
      + ,dfxmf(ibc(ii+i))+ipt(1)*(icbc+ipt(3)))
      + ,icbc=0, 1)
    Do j = 1, (ipt(4)-1)/2
      jj = min(ipt(4)-1, (j+1)*2-1)
      write (21,532)
      + (bcxmf(ibc(ii+i))+ipt(1)*(icbc+ipt(3)))
      + ,dfxmf(ibc(ii+i))+ipt(1)*(icbc+ipt(3)))
      + ,icbc=j*2, jj)
    EndDo
  EndIf
EndIf
write (21,536) ipt(22) ! constant gas phase flux

```

```

If (ipt(22) .gt. 0) Then
  iptbc = ipt(18) + ipt(19) + ipt(20) + ipt(21)
  write (21,537)
  Do i = 1, ipt(22)
    write (21,524) ibc(iptbc+i), source(i)
  EndDo
EndIf
write (21,538) ipt(23) ! constant aqueous phase flux
If (ipt(23) .gt. 0) Then
  iptbc = ipt(18) + ipt(19) + ipt(20) + ipt(21) + ipt(22)
  write (21,539)
  Do i = 1, ipt(23)
    write (21,524) ibc(iptbc+i), source(ipt(22)+i)
  EndDo
EndIf

c
c— If constant pressure nodes are present, then update saturations.
if ((ipt(18) .gt. 0 .or. ipt(19) .gt. 0) .and. lctrl(1)) then
  call SATW
endif

c
c===== INITIALIZE VARIABLES =====
c
c— Do not perform these initializations if the run is a restart.
if (.not. lctrl(26)) then
  C
  C— Now assign aqueous phase water mole fractions.
  C
  if (ipt(4) .gt. 0) then
    ix = ipt(14)*ipt(1)
    do 460 i=1, ipt(1)
      xmf(ipt(9)+ix+i) = rone
      do 460 ii=1, ipt(4)
        if (icp(ipt(3)+ii) .ne. icp(ipt(3)+ipt(14)+1))
          + xmf(ipt(9)+ix+i) = xmf(ipt(9)+ix+i)
          + - xmf(ipt(9)+(ii-1)*ipt(1)+i)
        460 continue
      C
      C— Now assign gas phase water mole fractions. No gas phase water
      C— vapor equilibrations are permissible when the aqueous phase is
      C— absent.
      C
      if (ipt(3) .gt. 0) then
        do 470 i=1, ipt(1)
          if (lctrl(10)) then
            cvpt = cvp(ipt(15)+1)
            else
              cvpt = cvp(ipt(15)+1)
              + dtemp(ipt(15))*ipt(89)+idepth(i)
            end if
          470 if (cvp(ipt(15)+1) .ge. zero)
            + xmf(ipt(13)*ipt(1)+i)
            + = xmf(ipt(9)+ix+i)*cvpt
            + /(temp(i)*cden(ipt(15)+1)*r*patm)
          end if
        end if
      C
      C— Finally assign gas phase nitrogen mole fractions.
      C
      if (ipt(3) .gt. 0) then
        if (lctrl(9)) then
          in = ipt(3) - 2
          else
            in = ipt(3) - 1
          end if
          ix = in * ipt(1)
          do 475 i=1, ipt(1)
            xmf(ipt(8)+ix+i) = rone
            do 475 ii=1, ipt(3)
              if (icp(ii) .ne. icp(in+1)) then
                xmf(ipt(8)+ix+i) = xmf(ipt(8)+ix+i)
                + - xmf(ipt(8)+(ii-1)*ipt(1)+i)
              end if
            475 continue
          end if
        end if
      C
      C— Compute the phase molecular weights and densities. Do not

```



```

C— compute the phase density derivative terms on the first time
C— step unless the run is a restart.
C
c if(.not.lctrl(26)) then
  lkeep = lctrl(22)
  lctrl(22) = .false.
  call MOLEWT
  lctrl(22) = lkeep
c else
c call MOLEWT
c end if
C
C— Compute solid phase equilibrium organic mole fractions when the
C— run is not a restart.
C
if(.not.lctrl(26)) then
  if(ipt(14).gt.0.and.ipt(16).gt.0) then
    iaq = ipt(3)
    isol = ipt(3) + ipt(4) + ipt(5)
    do 480 ii = 1, ipt(15)
      ngas = (igas-1) * ipt(1)
      inapl = inapl + 1
      if(icp(iaq+1).eq.ii) iaq=iaq+1
      if(icp(isol+1).eq.ii) isol=isol+1
      naq = (iaq-1) * ipt(1)
      nsol = (isol-1) * ipt(1)
      do 480 i = 1, ipt(1)
        if(icp(isol).eq.ii) then
          if(xmf(naq+i).gt.0.d0) then
            xmf(nsol+i) = bok(1)*(xmf(naq+i)*cmw(icp(isol))
+             * den(ipt(1)+i) * 1.0d3) ** (rone/bom(1))
          else
            xmf(nsol+i) = 0.d0
          end if
          if(lctrl(19)) then
            if(xmf(naq+i).gt.0.d0) then
              xmf(nsol+i+ipt(1)) = xbok*bok(1)*(xmf(naq+i)
+              *cmw(icp(isol))*den(ipt(1)+i)*1.0d3)
+              ** (rone/ (xbom * bom(1)))
            else
              xmf(nsol+i+ipt(1)) = 0.d0
            end if
          end if
        end if
      end if
480 continue
    end if
  end if
c
c===== WELL CONDITIONS =====
c
if (ipt(29).ne.0) write (ipt(29),*)
+ 'Reading Extraction / Injection Well Conditions'
c
C— Read logical variable indicating if a well is included.
call commnt (13,21)
read (13,*) lctrl(12) ! Is a well to be simulated?
write (21,540) lctrl(12)
c
C— Read the volumetric flux (standard cubic feet per minute).
If (lctrl(12)) then
  call commnt (13,21)
  read (13,*) qwell
c
C— Read the well radius (m); and the minimum and maximum node numbers
C— along the well screen.
C— Error Check: rwell > 0; all nodes along screen have same radius.
call commnt (13,21)
read (13,*) rwell,ii,jj
if (rwell .le. zero) call ErrorMessage (65,0,ipt(29))
if (xnode(ii).ne.rwell .or. xnode(jj).ne.rwell)
+ call ErrorMessage (66,0,ipt(29))
iptbc = ipt(18)+ipt(19)+ipt(20)+ipt(21)+ipt(22)+ipt(23)
ipt(24) = 1 ! number of nodes along well screen.
ibc(iptbc+ipt(24)) = ii ! nodes along well screen.
Do i = ii+1, jj-1
  If (xnode(i) .eq. rwell) Then
    ipt(24) = ipt(24) + 1
    ibc(iptbc+ipt(24)) = i
  EndIf
EndDo
ipt(24) = ipt(24) + 1
ibc(iptbc+ipt(24)) = jj ! bottom node along well screen
zwell = zero ! determine length of well screen
Do i = 1, ipt(24)-1
  zwell = zwell + znode(ibc(iptbc+i+1)) - znode(ibc(iptbc+i))
EndDo
Do 350 i = 1, ipt(24)-1 ! identify elements along the well screen
  nd1 = ibc(iptbc+i)
  nd2 = ibc(iptbc+i+1)
  Do j = 1, ipt(0)
    i3 = 3*(j-1)
    If ((nd1 .eq. nodel(i3+1) .or. nd1 .eq. nodel(i3+2) .or.
+     nd1 .eq. nodel(i3+3)) .and.
+     (nd2 .eq. nodel(i3+1) .or. nd2 .eq. nodel(i3+2) .or.
+     nd2 .eq. nodel(i3+3))) Then
      ibc(iptbc+ipt(24)+i) = j
      goto 350
    EndIf
  EndDo
350 Continue
c
C— Write well conditions.
write (21,541) qwell,rwell,zwell,ipt(24)
Do i = 1, ipt(24)-1
  write (21,542) ibc(iptbc+ipt(24)+i),ibc(iptbc+i),
+ ibc(iptbc+i+1)
EndDo
qwell = qwell / 2118.60d0 ! convert cfm to m^3/s
Endif
c
C— Generate boundary condition pointers.
ipt(62) = ipt(18) + ipt(19) + ipt(20) + ipt(21)
ipt(63) = ipt(62) + ipt(22)
ipt(64) = ipt(63) + ipt(23)
C
C===== VELOCITY BOUNDARY CONDITIONS =====
C
if (ipt(29).ne.0)
+ write (ipt(29),*) 'Reading Velocity Boundary Conditions'
C
C— Read a logical variable indicating if the bottom boundary is
C— impervious.
C
if(lctrl(18)) then
  call commnt (13,21)
  read (13,*) lctrl(28)
C
C— Read a logical variable indicating if the R.H.S. boundary is
C— impervious.
C
call commnt (13,21)
read (13,*) lctrl(29)
C
C— Read a logical variable indicating if the L.H.S. boundary is
C— impervious. Note that this boundary will be adjusted in the
C— presence of a well.
C
call commnt (13,21)
read (13,*) lctrl(30)
C
C— Read a logical variable indicating if the top boundary is
C— impervious.
C
call commnt (13,21)
read (13,*) lctrl(31)
write (21,555) lctrl(28),lctrl(29),lctrl(30),lctrl(31)
C
C— If lctrl(31) is true read the length of the cap (m). The cap is
C— assumed to extend from the well to the input value.
C
if(lctrl(31)) then
  call commnt (13,21)
  read (13,*) caplen
  write(21,556) caplen
end if
end if

```

```

Endif
EndDo
ipt(24) = ipt(24) + 1
ibc(iptbc+ipt(24)) = jj ! bottom node along well screen
zwell = zero ! determine length of well screen
Do i = 1, ipt(24)-1
  zwell = zwell + znode(ibc(iptbc+i+1)) - znode(ibc(iptbc+i))
EndDo
Do 350 i = 1, ipt(24)-1 ! identify elements along the well screen
  nd1 = ibc(iptbc+i)
  nd2 = ibc(iptbc+i+1)
  Do j = 1, ipt(0)
    i3 = 3*(j-1)
    If ((nd1 .eq. nodel(i3+1) .or. nd1 .eq. nodel(i3+2) .or.
+     nd1 .eq. nodel(i3+3)) .and.
+     (nd2 .eq. nodel(i3+1) .or. nd2 .eq. nodel(i3+2) .or.
+     nd2 .eq. nodel(i3+3))) Then
      ibc(iptbc+ipt(24)+i) = j
      goto 350
    EndIf
  EndDo
350 Continue
c
C— Write well conditions.
write (21,541) qwell,rwell,zwell,ipt(24)
Do i = 1, ipt(24)-1
  write (21,542) ibc(iptbc+ipt(24)+i),ibc(iptbc+i),
+ ibc(iptbc+i+1)
EndDo
qwell = qwell / 2118.60d0 ! convert cfm to m^3/s
Endif
c
C— Generate boundary condition pointers.
ipt(62) = ipt(18) + ipt(19) + ipt(20) + ipt(21)
ipt(63) = ipt(62) + ipt(22)
ipt(64) = ipt(63) + ipt(23)
C
C===== VELOCITY BOUNDARY CONDITIONS =====
C
if (ipt(29).ne.0)
+ write (ipt(29),*) 'Reading Velocity Boundary Conditions'
C
C— Read a logical variable indicating if the bottom boundary is
C— impervious.
C
if(lctrl(18)) then
  call commnt (13,21)
  read (13,*) lctrl(28)
C
C— Read a logical variable indicating if the R.H.S. boundary is
C— impervious.
C
call commnt (13,21)
read (13,*) lctrl(29)
C
C— Read a logical variable indicating if the L.H.S. boundary is
C— impervious. Note that this boundary will be adjusted in the
C— presence of a well.
C
call commnt (13,21)
read (13,*) lctrl(30)
C
C— Read a logical variable indicating if the top boundary is
C— impervious.
C
call commnt (13,21)
read (13,*) lctrl(31)
write (21,555) lctrl(28),lctrl(29),lctrl(30),lctrl(31)
C
C— If lctrl(31) is true read the length of the cap (m). The cap is
C— assumed to extend from the well to the input value.
C
if(lctrl(31)) then
  call commnt (13,21)
  read (13,*) caplen
  write(21,556) caplen
end if
end if

```

```

C
C===== PRINT INITIAL CONDITIONS =====
C
  If (lprnt(3)) Then
    if (ipt(29).ne.0) write (ipt(29),*)
    + 'Writing Initial Conditions'
    call prnt(0)
  Endif
c
c--- Formats
499 format ('INPUT2 ',69('='))
500 format ('INITIAL PRESSURE DISTRIBUTION ',46('='))
    + ' Pressures are defined as:      ipt(75) = ',i5/
    + ' 1 = compute assuming: Pg = 1atm' /
    + '                               Pa = hydrostatic;' /
    + ' 2 = input Pg and Pa at all nodes;')
501 format (
+ ' Depth to water table (m):      wtdpth = ',e12.4)
502 format (
+ ' Input initial gauge pressure (Pa) at all nodes:/'
+ ' Node      Aqueous Phase      Gas Phase')
503 format (3x,i6,4x,e12.4,4x,e12.4)
504 format ('DARCY VELOCITY COMPUTATION APPROACH',40('='))
    + ' Use nodal Darcy velocities if true:  lctrl(18) = ',i5)
505 format (
+ ' Time dependent Darcy velocity profiles:')
506 format (
+ ' Steady state Darcy velocity profile is generated from the '/'
+ ' user defined initial pressure distribution:')
507 format (
+ ' User defined steady state velocities:')
508 format (
+ ' Constant steady state Darcy velocity components/'
+ ' gas phase velocity, x(or r) component: vgx = ',e12.4/
+ ' gas phase velocity, z component:      vgz = ',e12.4/
+ ' aqueous phase velocity, x component   vax = ',e12.4/
+ ' aqueous phase velocity, z component:   vaz = ',e12.4)
509 format (
+ ' Gas Phase      Aqueous Phase/'
+ ' Node      x-dir  z-dir  x-dir  z-dir')
510 format (
+ ' Gas Phase      Aqueous Phase/'
+ ' Element x-dir  z-dir  x-dir  z-dir')
511 format (3x,i5,5x,4e12.4)
512 format ('INITIAL NAPL SATURATION AND COMPOSITION ',36('='))
513 format (
+ ' No NAPL is present')
514 format (
+ ' Uniform NAPL saturation and mole fractions in all elements,/'
+ ' NAPL sat Component mole fractions')
515 format (3x,6(e12.5,1x))
516 format (
+ ' Number of elements with NAPL present:  inoel = ',i5,/
+ ' Element NAPL sat Component mole fractions (1-ipt(15))')
517 format (2x,i6,2x,6(e12.5,1x))
518 format ('GAS, AQUEOUS, AND BIOPHASE PHASE INITIAL CONDI-
TIONS '
+ ',24('='))
519 format (
+ ' Uniform gas phase oxygen mole fraction:  xog = ',e12.4)
543 format (
+ ' Uniform gas phase nutrient mole fraction: xng = ',e12.4)
544 format (
+ ' Uniform aqueous phase oxygen mole fraction: xoa = ',e12.4)
545 format (
+ ' Uniform aqueous phase nutrient mole fraction:xna = ',e12.4)
546 format (
+ ' Non-uniform aqueous phase oxygen mole fraction')
547 format (
+ ' Non-uniform gas phase oxygen and nutrient mole fraction')
548 format (
+ ' Non-uniform aqueous phase oxygen and nutrient mole fraction')
520 format (
+ ' Non-uniform gas phase oxygen mole fraction')
521 format ('BOUNDARY CONDITIONS ',56('='))
522 format (
+ ' Number of nodes with constant gas pressure: ipt(18) = ',i5)
523 format (
+ ' Node      Constant gas phase pressure (Pa gauge)')
524 format (3x,i5,14x,e12.4)
525 format (
+ ' Number of nodes w/constant aqueous press.: ipt(19) = ',i5)
526 format (
+ ' Node      Constant aqueous phase pressure (Pa gauge)')
527 format (
+ ' Number of nodes w/gas phase component ' /
+ ' boundary conditions:      ipt(20) = ',i5)
528 format (' Gas phase BCs:' /
+ ' Node BC type BC values for gas phase components' /
+ ' in pairs (concentration:Dm/length)')
529 format (20x,a13,13x,a13)
531 format (3x,i5,6x,i1,4x,2(e12.4,':',e10.4,2x))
532 format (19x,2(e12.4,':',e10.4,2x))
533 format (
+ ' Number of nodes w/aqueous phase ' /
+ ' component boundary conditions:  ipt(21) = ',i5)
534 format (' Aqueous phase BC:' /
+ ' Node BC type BC values for aqueous phase components/'
+ ' in pairs (concentration:Dm/length)')
536 format (
+ ' Number of nodes w/constant gas phase flux:ipt(22) = ',i5)
537 format (
+ ' Node      Constant gas phase flux (scms)')
538 format (
+ ' Number of nodes w/constant aq. phase flux:ipt(23) = ',i5)
539 format (
+ ' Node      Constant aqueous phase flux (cms)')
540 format ('EXTRACTION / INJECTION WELL DATA ',43('='))
    + ' Include an extraction/injection well:  lctrl(12) = ',i5)
541 format(
+ ' Volumetric fluid extraction rate (scfm): qwell = ',e12.4/
+ ' Well radius (m):      rwell = ',e12.4/
+ ' Length of well screen (m):      zwell = ',e12.4/
+ ' Number of nodes along well screen:  ipt(24) = ',i5 /
+ ' Element and node numbers along well screen:/'
+ ' Element      Element nodes along well screen')
542 format (9x,i6,18x,2i8)
550 format ('RESTART INFORMATION ',56('='))
    + ' Run is a restart:      lctrl(26) = ',i5/
    + ' Restart data:      (unit 28) = ',a20/
    + ' Run is a continuation:  lctrl(32) = ',i5)
555 format ('VELOCITY BOUNDARY CONDITIONS',48('='))
    + ' Bottom boundary is impervious:  lctrl(28) = ',i5/
    + ' R.H.S. boundary is impervious:  lctrl(29) = ',i5/
    + ' L.H.S. boundary is impervious:  lctrl(30) = ',i5/
    + ' Note: L.H.S. is adjusted when a well is present/'
    + ' Top boundary is impervious:  lctrl(31) = ',i5)
556 format (
+ ' Top boundary is covered from x = 0 to:  caplen = ',e12.4)
601 format(3e15.8,i10)
602 format(5e15.8)
    return
    end

```

Subroutine - mobil.f

```

C-----
C
C mobil.f - This subroutine computes capacity coefficients, and
C aqueous and gas phase mobility terms in stacked
C storage.
C-----
subroutine mobil (iter)
include 'dimen.inc'
common /cb1/ matel(nelmx),node1(nel3),nodept(nnm),nelpt(nel3),
+ matpt(nn6)
common /cb2/ p(nn3)
common /cb2b/ pt(nn3)
common /cb3/ sat(nnstk3)
common /cb3b/ satt(nnstk3)
common /cb4/ satk(nnstk2),cc(nnstk)
common /cb5a/ bphi(nmbk),bpermh(nmbk),bpermv(nmbk)
common /cb5b/ bvgn(nmbk),bvga(nmbk),bvgm(nmbk),bsrw(nmbk)
common /cb6/ pmob(nnstk4)
common /cb6b/ por(nelmx),srw(nnstk)
common /cb8/ vis(nnm),pmw(nn3)
C-----
C Dimension local arrays.
C
dimension c(3)
data ccmin / 1.0d-7 /
C
C Branch computations for single and multiple material property
C blocks uniform material properties
if (ipt(26) .eq. 1) then ! uniform material properties.
C
C Update chord slope approx for capacity coeffs, no stacking.
do 10 i = 1,ipt(1)
if (iter .eq. 1) then
cc(i) = -ccmin
else
x = p(2*ipt(1)+i) - pt(2*ipt(1)+i)
if (x .ne. 0.0d0) then
cc(i) = (sat(ipt(2)+i) - satt(ipt(2)+i)) / x
if (dabs(cc(i)) .lt. ccmin) cc(i) = -ccmin
else
cc(i) = -ccmin
endif
endif
C
C Compute relative permeability.
c(1) = 1.0d0 / (1.0d0 - srw(i))
! van Genuchten function
c(2) = 1.0d0 / bvgm(1)
sweff = (sat(ipt(2)+i) - srw(i)) * c(1)
steff = (sat(ipt(2)+i)+sat(ipt(49)+i) - srw(i)) * c(1)
pmob(2*ipt(2)+i) = dsqrt(sweff) * (1.0d0 - (1.0d0 - ! kra
+ sweff**c(2)) ** bvgm(1)) ** 2
+ pmob(i) = dsqrt(1.0d0-steff) * (1.0d0 ! krg
+ -steff**c(2)) ** (2*bvgm(1))
C
C Compute mobilities. Note 'bpermh' is an anisotropy factor.
pmob(3*ipt(2)+i) = bpermv(1) ! aq. z-dir
+ * pmob(2*ipt(2)+i) * wvis
pmob(2*ipt(2)+i) = bpermh(1) * pmob(3*ipt(2)+i) ! aq. x-dir
if (lctrl(20)) then ! include Klinkenberg adjustment

```

```

pmob(ipt(2)+i) = bpermv(1) * pmob(i) * vis(i) *
+ (1.0d0 + b/(patm+p(i)))
else
pmob(ipt(2)+i) = bpermv(1) * pmob(i) * vis(i) ! gas z-dir
endif
pmob(i) = bpermh(1) * pmob(ipt(2)+i) ! gas x-dir
10 continue
C
C Multiple material property blocks.
else
C
C Update chord slope approx for capacity coeffs.
do 20 i = 1,ipt(1)
do 20 j = 0,nodept(i+1)-nodept(i)-1
jj = nodept(i) + j
if (iter .eq. 1) then
cc(jj) = -ccmin
else
x = p(2*ipt(1)+i) - pt(2*ipt(1)+i)
if (x .ne. 0.0d0) then
cc(jj) = (sat(ipt(2)+jj) - satt(ipt(2)+jj)) / x
if (dabs(cc(jj)) .lt. ccmin) cc(jj) = -ccmin
else
cc(jj) = -ccmin
endif
endif
C
C Compute relative permeability.
mprop = matpt(jj)
c(1) = 1.0d0 / (1.0d0 - srw(jj))
c(2) = 1.0d0 / bvgm(mprop)
sweff = (sat(ipt(2)+jj) - srw(jj)) * c(1)
steff = (sat(ipt(2)+jj)+sat(ipt(49)+jj) - srw(jj)) * c(1)
sweff = dmin1(sweff,1.0d0)
steff = dmin1(steff,1.0d0)
steff = dmax1(steff,xround)
sweff = dmax1(sweff,xround)
pmob(2*ipt(2)+jj) = dsqrt(sweff) ! kra
+ * (1.0d0 - (1.0d0 - sweff**c(2)) ** bvgm(mprop)) ** 2
+ pmob(jj) = dsqrt(1.0d0-steff) ! krg
+ * (1.0d0 - steff**c(2)) ** (2*bvgm(mprop))
C
C Compute mobilities. Note 'bpermh' is an anisotropy factor.
pmob(3*ipt(2)+jj) = bpermv(mprop) ! aq. z-dir
+ * pmob(2*ipt(2)+jj) * wvis
+ pmob(2*ipt(2)+jj) = bpermh(mprop) ! aq. x-dir
+ * pmob(3*ipt(2)+jj)
if (lctrl(20)) then ! include Klinkenberg adjustment
pmob(ipt(2)+jj) = bpermv(mprop) * pmob(jj) * vis(i) *
+ (1.0d0 + b/(patm+p(i)))
else
pmob(ipt(2)+jj) = bpermv(mprop) * pmob(jj) *
+ vis(i) ! gas z-dir
endif
pmob(jj) = bpermh(mprop) * pmob(ipt(2)+jj) ! gas x-dir
20 continue
endif
return
end

```

Subroutine - molewt.f

```

C-----
C
C MOLEWT.f - Subroutine which computes the fluid phase molecular
C weights, densities, and mass densities.
C
C Required Control Flags:
C
C lctrl(1) - logical variable controlling presence of flow

```

```

C
C solution
C lctrl(1) = .true. - compute flow solution
C lctrl(1) = .false. - skip flow solution
C lctrl(22) - logical variable controlling execution of
C solution for dden
C lctrl(22) = .true. - compute dden
C lctrl(22) = .false. - skip dden
C

```

```

C-----
subroutine MOLEWT
include 'dimen.inc'
C
C--- Declare and define common block variables.
C
common /cb2/ p(nn3)
common /cb6c/ temp(nnmx)
common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
+ chen(ncmp),casol(ncmp),cmdif(ncmp2)
common /cb8/ vis(nnmx),pmw(nn3)
common /cb9/ xmf(nmf)
common /cb10/ den(nn6)
common /cb10b/ dden(nn6),pmwt(nn3),dent(nn6)
i2=ipt(40)
i3=ipt(41)
C
C--- Initial the phase molecular weights and densities with zero.
C
do 90 i=1+ipt(1),ipt(41)
90 pmw(i)=zer0
do 95 i=1,ipt(44)
95 den(i)=zer0
C
C--- Compute the gas phase molecular weight and density
C--- and mass density.
C
if (ipt(3).gt.0) then
do 100 i = 1,ipt(1)
pmw(i) = zer0
do 105 ii = 1,ipt(3)
ic = icp(ii)
iim1=(ii-1)*ipt(1)
C
C--- If a mole fraction is negative, do not include that component
C--- in the phase molecular weight and adjust the mole fraction of
C--- the principal component, nitrogen, accordingly.
C
if (xmf(i+iim1).lt.zer0) then
if (lctrl(9)) then
ipt3 = ipt(3)
ic3 = icp(ipt3-1)
pmw(i) = pmw(i) - xmf(i+iim1)*cmw(ic3)
else
ipt3 = ipt(3)
ic3 = icp(ipt3)
pmw(i) = pmw(i) - xmf(i+iim1)*cmw(ic3)
end if
else
pmw(i) = pmw(i) + xmf(i+iim1)*cmw(ic)
end if
105 continue
106 den(i) = (p(i) + patm) / (r * temp(i))
den(i+i3) = den(i)*pmw(i)
C
C--- Compute the gas phase compositional molecular weight time

```

Subroutine - mpex.f

```

C-----
C
C MPEX.f - Subroutine which computes the mole exchange
C terms and the mass exchange terms for the flow
C routines.
C
C Required Control Flags:
C
C lctrl(16) - logical variable denoting method of including
C biological reaction
C lctrl(16) = .true. - include bioreaction in
C aqueous transport
C lctrl(16) = .false. - solve FEM solution for rate

```

```

C--- derivative.
C
if (lctrl(1).and.lctrl(22)) dden(ip)=(pmw(i)-pmwt(i))/t(8)
100 continue
end if
C
C--- Compute the aqueous phase molecular weight and density
C--- and mass density. Use Amagat's Law for the molar density.
C
if (ipt(4).gt.0) then
do 150 i = 1,ipt(1)
ip=i+ipt(1)
do 155 ii = 1,ipt(4)
iim1=(ii-1)*ipt(1)
ipc=i+ipt(9)+iim1
ipt3 = ipt(3)
ic=icp(ii+ipt3)
pmw(ip) = pmw(ip) + xmf(ipc)*cmw(ic)
155 den(ip) = den(ip) + xmf(ipc)*cmw(ic)/cden(ic)
den(ip) = rone/den(ip)
den(ip+i3) = den(ip)*pmw(ip)
C
C--- Compute the aqueous phase density derivatives due to
C--- compositional effects. This term is on a mass basis.
C
if (lctrl(1).and.lctrl(22))
+ dden(ip)=(den(ip+i3)-den(ip+i3))/t(8)
150 continue
end if
C
C--- Compute the organic phase molecular weight and density
C--- and mass density. Use Amagat's Law for the molar density.
C
if (ipt(5).gt.0) then
do 200 i = 1,ipt(1)
ip=i+i2
do 205 ii = 1,ipt(5)
iim1=(ii-1)*ipt(1)
ipc=i+ipt(10)+iim1
iip3 = ii+ipt(3)+ipt(4)
ic=icp(iip3)
pmw(ip) = pmw(ip) + xmf(ipc)*cmw(ic)
205 den(ip) = den(ip) + xmf(ipc)*cmw(ic)/cden(ic)
if (den(ip).gt.zer0) den(ip) = rone/den(ip)
den(ip+i3) = den(ip)*pmw(ip)
C
C--- Compute the organic phase mass density derivative. The organic
C--- phase is assumed to be incompressible and therefore any
C--- change in organic density is due to compositional effects.
C
if (lctrl(22)) dden(ip)=(den(ip+i3)-den(ip+i3))/t(8)
200 continue
end if
return
end

```

```

C
C limited biophase
C lctrl(3) - logical variable controlling inclusion of
C biodegradation
C lctrl(3) = .true. - include biodegradation
C lctrl(3) = .false. - neglect biodegradation
C lctrl(19) - logical variable controlling type of sorption
C lctrl(19) = .true. - two compartment sorption
C lctrl(19) = .false. - single compartment
C sorption
C-----
subroutine MPEX
include 'dimen.inc'

```

```

C
C— Declare and define common block variables.
C
common /cb1/ matel(nelmx),node1(nel3),nodept(nnmx),nelpt(nel3),
+   matpt(nn6)
common /cb1c/ xnodel(nnmx),znodel(nnmx),rbar(nelmx),area(nelmx)
common /cb2c/ q(nel4)
common /cb3/ sat(nnstk3)
common /cb3b/ satt(nnstk3)
common /cb6d/ dtemp(nzmax6),idepth(nnmx)
common /cb7b/ cmw(ncmp),cwp(ncmp),cden(ncmp),
+   chen(ncmp),casol(ncmp),cmdif(ncmp2)
common /cb9/ xmf(nmf)
common /cb9b/ xmf(nmf)
common /cb10/ den(nn6)
common /cb11/ pex(nns10),rxnp(nn2)
common /cb62/ rxn(nmf),cex(nmfs)
common /cb62b/ rhsex(nmfs)
common /cb63/ kex(ncmp5),kmax(ncmp5)

common /cb64/ bok(nbcmp),bom(nbcmp),krtcd(ncmp)

C
C— Dimension local arrays.
C
dimension qaveg(nnmx),qavea(nnmx),xl(nnmx)

C
C— Klen is the dimension used to calculate the exchange coefficient
C— bounds. Ordinarily this is not used and the exchange coefficients
C— are bounded using the element dimension. However for column
C— experiments, the column dimension may be appropriate and should
C— be entered here in meters. Otherwise set klen to a negative
C— number.
C
data klen / -0.10d0 /
data limag, limao, limgo, limab, limas /
+ .true., .true., .true., .true., .false. /

C
C— This statement sets upper limits on exchange coefficients.
C— Limits are expressed in the fraction of equilibrium it is
C— possible to approach over one element. Limits are in order:
C— gas/aqueous, aqueous/NAPL, gas/NAPL, aqueous/biophase,
C— and aqueous/solid.
C
do 1 i = 1, ipt(1)
  xl(i) = zero
  qaveg(i) = rone
  qavea(i) = rone
1 continue
do 2 i = 1, ipt(0)
  i3=i*3
  n3=ndel(i3)
  n2=ndel(i3-1)
  n1=ndel(i3-2)
  darea = dsqrt( 2.0d0 * area(i) )
  if(klen.lt.zero) then
    xl(n1) = dmax1( xl(n1), darea )
    xl(n2) = dmax1( xl(n2), darea )
    xl(n3) = dmax1( xl(n3), darea )
  else
    xl(n1) = klen
    xl(n2) = klen
    xl(n3) = klen
  end if
  if(ictrl(18)) then
    qelg1 = dsqrt( q(n1)**2 + q(n1+ipt(1))**2 ) / darea
    qelg2 = dsqrt( q(n2)**2 + q(n2+ipt(1))**2 ) / darea
    qelg3 = dsqrt( q(n3)**2 + q(n3+ipt(1))**2 ) / darea
    qela1 = dsqrt( q(n1+ipt(40))**2 + q(n1+ipt(41))**2 ) / darea
    qela2 = dsqrt( q(n2+ipt(40))**2 + q(n2+ipt(41))**2 ) / darea
    qela3 = dsqrt( q(n3+ipt(40))**2 + q(n3+ipt(41))**2 ) / darea
    qaveg(n1) = dmin1( qaveg(n1), qelg1 )
    qavea(n1) = dmin1( qavea(n1), qela1 )
    qaveg(n2) = dmin1( qaveg(n2), qelg2 )
    qavea(n2) = dmin1( qavea(n2), qela2 )
    qaveg(n3) = dmin1( qaveg(n3), qelg3 )
    qavea(n3) = dmin1( qavea(n3), qela3 )
  else
    qelg = dsqrt( q(i)**2 + q(i+ipt(0))**2 ) / darea

```

```

    qela = dsqrt( q(i+ipt(67))**2 + q(i+ipt(68))**2 ) / darea
    qaveg(n1) = dmin1( qaveg(n1), qelg )
    qavea(n1) = dmin1( qavea(n1), qela )
    qaveg(n2) = dmin1( qaveg(n2), qelg )
    qavea(n2) = dmin1( qavea(n2), qela )
    qaveg(n3) = dmin1( qaveg(n3), qelg )
    qavea(n3) = dmin1( qavea(n3), qela )
  end if
2 continue
C
C— Compute the phase mass exchange. Initially iterate over just
C— the mobile phases and then add NAPL, solid, and biophases.
C— Aqueous/gas exchange is controlled by the aqueous phase
C— mole fractions. NAPL/gas and NAPL/aqueous exchange are controlled
C— by the mole fractions in the respective mobile phases.
C— Solid/aqueous mass exchange is governed by the solid phase
C— loading. Aqueous/ biophase mass exchange is governed by the
C— biophase mole fractions. For now this routine is not stacked.
C— Stacking will arise when the exchange coefficients are correlated
C— with velocities or material properties.
C
ipt2x2 = ipt(49)
ipt2x4 = ipt(51)
ipt2x5 = ipt(52)
ipt2x6 = ipt(53)
ipt2x7 = ipt(54)
ipt2x8 = ipt(55)
ipt2x9 = ipt(56)
ipt2 = ipt(2)
ipt1 = ipt(1)

C
C— Zero the exchange vectors.
C
do 90 i = 1, 10*ipt2
  90 pex(i) = zero
do 91 i = 1, ipt(61)*ipt2
  91 cex(i) = zero
  91 rhsex(i) = zero
C
C— Begin iterations over the various components. First the organic
C— components.
C
icg = 0
ica = 0
icn = 0
ics = 0
icb = 0
do 100 iexc = 1, ipt(15)
  ic = icp(iexc)
  if(icp(icg+1).eq.ic) then
    icg = icg + 1
    igc = icp(icg)
  end if
  if(icp(ipt(3)+ica+1).eq.ic) then
    ica = ica + 1
    iac = icp(ipt(3)+ica)
  end if
  if(icp(ipt(3)+ipt(4)+icn+1).eq.ic) then
    icn = icn + 1
  end if
  if(icp(ipt(3)+ipt(4)+ipt(5)+ics+1).eq.ic) then
    ics = ics + 1
    isc = icp(ipt(3)+ipt(4)+ipt(5)+ics)
  end if
  if(icp(ipt(3)+ipt(4)+ipt(5)+ipt(6)+icb+1).eq.ic) then
    icb = icb + 1
  end if
  ig = (icg-1) * ipt(1)
  ia = ipt(9) + (ica-1) * ipt(1)
  in = ipt(10) + (icn-1) * ipt(1)
  is = ipt(11) + (ics-1) * ipt(1)
  ib = ipt(12) + (icb-1) * ipt(1)
  igs = (icg-1) * ipt2
  ias = (ipt(3) + ica - 1) * ipt2
  ins = (ipt(3) + ipt(4) + icn - 1) * ipt2
  iss = (ipt(3) + ipt(4) + ipt(5) + ics - 1) * ipt2
  ibs = (ipt(3) + ipt(4) + ipt(5) + ipt(6) + icb - 1) * ipt2

```

```

C— Iterate over the nodes. Initial temperature dependent parameters.
C
do 100 i = 1, ipt(1)
  if((lctrj(10)) then
    cvpt = cvpt(ic)
    casolt = casolt(ic)
  else
    itemp = (ic-1)*ipt(89)+idepth(i)
    cvpt = cvpt(ic) + dtemp(itemp)
    itemp = jtemp+3*ipt(88)
    casolt = casolt(ic) + dtemp(itemp)
  end if
  do 100 ii = 0,nodept(i+1)-nodept(i)-1
    istk = ii + nodept(i)
  C
  C— Initialize saturations. Control NAPL exchange with NAPL
  C— saturations from the previous time step.
  C
  sgcont = sat(istk)
  snt = satt(ipt2x2+istk)
  C
  C— Calculate terms when NAPL is present.
  C
  if((snt.gt.zer0).and.(xmf(i+in).gt.zer0)) then
  C
  C— Calculate gas phase exchange when the gas phase is present.
  C
  if(ipt(3).gt.0.and.igc.eq.ic) then
  C
  C— PUT USER DEFINED GAS/NAPL MASS TRANSFER COEFFI-
  C— CIENT FUNCTION HERE.
  C— ASSIGN IT TO KGO TO UTILIZE BUILT-IN CONTROL.
  C
  if(limgo) then
    kgo = kex(5*(ic-1)+3)
  C
  C— Gas/NAPL control on mass transfer coefficient with the greater of
  C— the gas phase advective velocity or diffusional velocity.
  C
  kgo = dmin1( kgo, -( dmax1( qaveg(i),
  +   cmdif(2*ic-1)/xl(i)**2 ) * dlog(
  +   kmax(ic+2*ipt(65))) ) )
  C
  C— PUT USER DEFINED GAS/NAPL MASS TRANSFER COEFFI-
  C— CIENT FUNCTION HERE.
  C— ASSIGN IT TO KGO TO BYPASS BUILT-IN LIMIT.
  C
  else
    kgo = kex(5*(ic-1)+3)
  end if
  C
  C— Calculate constants.
  C
  sdgas = den(i)
  keq = cvpt/patm
  cexgo = sdgas * kgo
  xmfj = xmf(i+ig)
  xmfj = xmf(i+in)
  C
  C— Prevent exchange from the gas phase into the NAPL for
  C— the single organic component case. Prevent exchange if
  C— the gas saturation is below sgtest.
  C
  if((sgcont.lt.sgtest).or.
  +   (xmfj.gt.keq*xmfj.and.ipt(15).eq.1))
  +   cexgo = zer0
  C
  pexgo = cexgo * (keq*xmfj - xmfj)
  C
  C— Adjust the exchange coefficient if the predicted exchange will
  C— exceed the amount of organic component available from the NAPL.
  C
  xflux1 = pexgo*(8)
  xflux2 = den(ipt(40)+i)*snt*xmf(i+in)
  if(xflux1.gt.xflux2) then
    xlimit = xflux2 / xflux1
    pexgo = xlimit * pexgo
    cexgo = xlimit * cexgo
  end if

```

```

C
C— Update left hand side terms.
C
cex(istk+igs) = cex(istk+igs) + cexgo
C
C— Update gas/NAPL phase mole exchange.
C
pex(istk) = pex(istk) + pexgo
C
C— Update gas/NAPL phase mass exchange.
C
pex(ipt2x5+istk) = pex(ipt2x5+istk)+pexgo*cmw(ic)
C
C— Update NAPL/gas component mole exchange.
C
cex(istk+ins) = cex(istk+ins) - pexgo
C
C— Update NAPL/gas phase mole exchange.
C
pex(ipt2x2+istk) = pex(ipt2x2+istk) - pexgo
C
C— Update NAPL/gas phase mass exchange.
C
pex(ipt2x7+istk) = pex(ipt2x7+istk)-pexgo*cmw(ic)
C
C— Update right hand side terms.
C
rhsex(istk+igs)
+   = rhsex(istk+igs)+cexgo*keq*xmf(i+in)
end if
C
C— Calculate aqueous phase exchange when the aqueous phase is
C— present.
C
if(ipt(4).gt.0.and.iac.eq.ic) then
C
C— PUT USER DEFINED AQUEOUS/NAPL MASS TRANSFER COEFFI-
C— CIENT FUNCTION
C— HERE. ASSIGN IT TO KAO TO UTILIZE BUILT-IN CONTROL.
C
if(limao) then
  kao = kex(5*(ic-1)+2)
C
C— Aqueous/NAPL control on mass transfer coefficient with the greater
C— of the gas phase or aqueous phase advective velocity or aqueous
C— phase diffusional velocity.
C
kao = dmin1( kao, -( dmax1( qavea(i),
+   cmdif(2*ic)/xl(i)**2 )
+   * dlog(kmax(ic-ipt(65)))) )
C
C— PUT USER DEFINED AQUEOUS/NAPL MASS TRANSFER COEFFI-
C— CIENT FUNCTION
C— HERE. ASSIGN IT TO KAO TO BYPASS BUILT-IN CONTROL.
  C
  else
    kao = kex(5*(ic-1)+2)
  end if
  C
  C— Calculate constants.
  C
  sdaq = den(i+ipt1)
  cexao = sdaq * kao
  xmfj = xmf(i+ia)
  C
  C— Prevent exchange from the aqueous phase to the NAPL.
  C
  if(ipt(5).eq.1.and.xmfj.gt.casolt*xmf(i+in))
  +   xmfj = casolt*xmf(i+in)
  C
  C— Update left hand side terms.
  C
  cex(istk+ias) = cex(istk+ias) + cexao
  C
  C— Update aqueous/NAPL phase mole exchange.
  C
  pexao = cexao*(casolt*xmf(i+in)-xmfj)
  C
  C— Adjust the exchange coefficient if the predicted exchange will

```

```

C— exceed the amount of organic component available from the NAPL.
C
  xflux1 = pexao*t(8)
  xflux2 = den(ipt(40)+i)*snt*xmf(i+in)
  if(xflux1.gt.xflux2) then
    xlimit = xflux2 / xflux1
    pexao = xlimit * pexao
    cexao = xlimit * cexao
  end if
  pex(ipt2+istk) = pex(ipt2+istk) + pexao
C
C— Update aqueous/NAPL phase mass exchange.
C
  pex(ipt2x6+istk) = pex(ipt2x6+istk)+pexao*cmw(ic)
C
C— Update NAPL/aqueous component mole exchange.
C
  cex(ins+istk) = cex(ins+istk) - pexao
C
C— Update NAPL/aqueous phase mole exchange.
C
  pex(ipt2x2+istk) = pex(ipt2x2+istk) - pexao
C
C— Update NAPL/aqueous phase mass exchange.
C
  pex(ipt2x7+istk) = pex(ipt2x7+istk)-pexao*cmw(ic)
C
C— Update right hand side terms.
C
  rhsex(ias+istk) = rhsex(ias+istk)
  + cexao * casolt * xmf(in+i)
  end if
  else
C
C— Calculate aqueous/gas exchange where NAPL is absent and both
C— mobile phases are present.
C
  if((ipt(3)+ipt(4)).ne.ipt(3).and.igc.eq.iac) then
C
C— PUT USER DEFINED AQUEOUS/GAS MASS TRANSFER COEFFI-
C— CIENT FUNCTION
C— HERE. ASSIGN IT TO KAG TO UTILIZE BUILT-IN CONTROL.
C
  if(limag) then
    kag = kex(5*(ic-1)+1)
C
C— Gas/aqueous control on mass transfer coefficient with the greater
C— of the gas or aqueous phase advective velocity or diffusional
C— velocity.
C
    kag = dmin1( kag, -( dmax1( qaveg(i)
  + , qavea(i), cmdif(2*ic-1)/xl(i)**2
  + , cmdif(2*ic)/xl(i)**2 )
  + * dlog(kmax(ic) ) ) )
C
C— PUT USER DEFINED AQUEOUS/GAS MASS TRANSFER COEFFI-
C— CIENT FUNCTION
C— HERE. ASSIGN IT TO KAG TO BYPASS BUILT-IN CONTROL.
C
  else
    kag = kex(5*(ic-1)+1)
  end if
C
C— Calculate constants.
C
  sdgas = den(i)
  sdaq = den(ipt1+i)
  keqag = patm*casolt/cvpt
C
C— Update left hand side terms for aqueous phase equations.
C
  cexag = sdaq * kag
  if(sgcont.lt.sgtest) cexag = zer0
  cex(ias+istk) = cex(ias+istk) + cexag
C
C— Update aqueous/gas phase mole exchange.
C
  pexag = cexag * (keqag*xmf(ig+i) - xmf(ia+i))

```

```

  pex(ipt2+istk) = pex(ipt2+istk) + pexag
C
C— Update aqueous/gas phase mass exchange.
C
  pex(ipt2x6+istk) = pex(ipt2x6+istk)+pexag*cmw(ic)
C
C— Update gas/aqueous phase mole exchange.
C
  pex(istk) = pex(istk) - pexag
C
C— Update gas/aqueous phase mass exchange.
C
  pex(ipt2x5+istk) = pex(ipt2x5+istk)-pexag*cmw(ic)
C
C— Update the right hand side terms for the component transport
C— equations.
C
  rhsex(igs+istk) = rhsex(igs+istk) - pexag
  rhsex(ias+istk)
  + rhsex(ias+istk) + cexag*keqag*xmf(ig+i)
  end if
  end if
C
C— Calculate the exchange terms when adsorption is considered.
C— Use Freundlich isotherms for the equilibrium solid phase loading.
C— Adsorption can only occur if the aqueous phase is present. Only
C— include adsorption when NAPL is not present.
C
  if((ipt(6).gt.0).and.(ipt(4).gt.0).and.iac.eq.isc)
  + then
C
C— Set the aqueous phase mole fraction for aqueous/solid exchange
C— to zero if it is nonpositive.
C
  if(xmf(ia+i).le.zer0) then
    xmf = zer0
  else
    xmf = xmf(ia+i)
  end if
C
C— PUT USER DEFINED AQUEOUS/SOLID MASS TRANSFER COEFFI-
C— CIENT FUNCTION
C— HERE. ASSIGN IT TO KSA TO UTILIZE BUILT-IN CONTROL.
C
  if(limas) then
    ksa = kex(5*(ic-1)+5)
C
C— Aqueous/solid control on mass transfer coefficient with the
C— greater of the aqueous phase advective velocity or
C— diffusional velocity.
C
    ksa = dmin1( ksa, -( dmax1( qavea(i),
  + cmdif(2*ic)/xl(i)**2)
  + *dlog(kmax(ic+4*ipt(65))))))
C
C— PUT USER DEFINED AQUEOUS/SOLID MASS TRANSFER COEFFI-
C— CIENT FUNCTION
C— HERE. ASSIGN IT TO KSA TO BYPASS BUILT-IN CONTROL.
C
  else
    ksa = kex(5*(ic-1)+5)
  end if
C
C— Calculate constants. This form puts the mass transfer resistance
C— for solid/aqueous interactions into the aqueous phase.
C
C
C
C— Set the aqueous phase equilibrium mole fraction to zero if the
C— solid phase mass fraction is nonpositive.
C
  if(xmf(is+i).le.zer0) then
    efrac = zer0
  else
    efrac = ( ( xmf(is+i) / bok(1) ) ** bom(1) )
  + / ( cmw(ic) * den(ipt1+i) * 1.0d3 )
  end if
  sdaq = den(ipt1+i)
C

```

```

C— Update left hand side terms for aqueous phase equations.
C
  cexas = sdaq * ksa
  cex(ias+istk) = cex(ias+istk) + cexas
C
C— Update left hand side terms for solid phase equations.
C
  cex(iss+istk) = zer0
C
C— Update aqueous/solid phase mole exchange. This form puts the
C— mass transfer resistance for solid/aqueous interactions into
C— the aqueous phase.
C
  pexas = cexas*(efrac-xmfa)
  pex(ipt2+istk) = pex(ipt2+istk) + pexas
C
C— Update solid/aqueous phase mass exchange. This form puts the
C— mass transfer resistance for solid/aqueous interactions into
C— the aqueous phase.
C
  pex(ipt2x8+istk) = pex(ipt2x8+istk)-pexas*cmw(ic)
C
C— Update aqueous/solid phase mass exchange. This form puts the
C— mass transfer resistance for solid/aqueous interactions into
C— the aqueous phase.
C
  pex(ipt2x6+istk) = pex(ipt2x6+istk)+pexas*cmw(ic)
C
C— Update right hand side terms, first for the aqueous phase and
C— then the solid phase.
C
  rhsex(iss+istk) = rhsex(iss+istk) - pexas
  rhsex(ias+istk) = rhsex(ias+istk) + cexas * efrac
C
C— Calculate terms for two compartment sorption model. This can
C— only be used for one organic component. The slow compartment
C— terms have been calculated above (ipt(6)=1), the fast compartment
C— terms are calculated below (ipt(6)=2).
C
  if((ipt(6).eq.2).and.lctrl(19)) then
C
C— PUT USER DEFINED AQUEOUS/SOLID MASS TRANSFER COEFFI-
C— CIENT FUNCTION
C— HERE. ASSIGN IT TO KSA TO UTILIZE BUILT-IN CONTROL.
C
    if(limas) then
      ksa = xkex*kex(5*(ic-1)+5)
C
C— Aqueous/solid control on mass transfer coefficient with the
C— greater of the aqueous phase advective velocity or
C— diffusional velocity.
C
      ksa = dmin1( ksa, -( dmax1(
+         qavea(i), cmdif(2*ic)/xl(i)**2 )
+         * dlog(kmax(ic+4*ipt(65))) ) ) )
C
C— PUT USER DEFINED AQUEOUS/SOLID MASS TRANSFER COEFFI-
C— CIENT FUNCTION
C— HERE. ASSIGN IT TO KSA TO BYPASS BUILT-IN CONTROL.
C
      else
        ksa = xkex*kex(5*(ic-1)+5)
      end if
C
C— Set the aqueous phase equilibrium mole fraction to zero if the
C— solid phase mass fraction is nonpositive.
C
      if(xmf(is+i+ipt1).le.zer0) then
        efrac = zer0
      else
        efrac = ( ( xmf(is+i+ipt1)/(xbok*bok(1))
+          ** ( xbom*bom(1) ) )
+          / ( cmw(ic) * den(ipt1+i) * 1.0d3 )
      end if
      cexas = sdaq * ksa
      cex(ias+istk) = cex(ias+istk) + cexas
      cex(iss+istk+ipt2) = zer0
      pexas = cexas*(efrac-xmfa)

```

```

  pex(ipt2+istk) = pex(ipt2+istk) + pexas
  pex(ipt2x8+istk)
+   = pex(ipt2x8+istk) - pexas*cmw(ic)
  pex(ipt2x6+istk)=pex(ipt2x6+istk)+pexas*cmw(ic)
  rhsex(iss+istk+ipt2)
+   = rhsex(iss+istk+ipt2) - pexas
  rhsex(ias+istk)=rhsex(ias+istk) + cexas * efrac
  end if
end if
C
C— Calculate aqueous/biophase mole transfer only if NAPL is absent
C— and the bioreaction terms are not inserted directly into the
C— aqueous phase equations.
C
  if(lctrl(3)) then
C
C— PUT USER DEFINED AQUEOUS/BIPHASE MASS TRANS-
C— FER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KAB TO UTILIZE BUILT-
C— IN CONTROL.
C
    if(limab) then
      kab = kex(5*(ic-1)+4)
C
C— Aqueous/biophase control on mass transfer coefficient with the
C— greater of the aqueous phase advective velocity or
C— aqueous phase diffusional velocity.
C
      kab = dmin1( kab, -( dmax1( qavea(i),
+         cmdif(2*ic)/xl(i)**2 )
+         * dlog(kmax(ic+3*ipt(65)))) )
C
C— PUT USER DEFINED AQUEOUS/BIPHASE MASS TRANS-
C— FER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KAB TO UTILIZE BUILT-
C— IN CONTROL.
C
      else
        kab = kex(5*(ic-1)+4)
      end if
C
C— Calculate constants.
C
      sdaq = den(ipt1+i)
C
C— This form assumes that biophase exchange is controlled in the
C— aqueous phase.
C
      sdb = sdaq
C
C— Update left hand side terms for aqueous phase equations. The
C— biomass term accounts for the volume of the reactive phase
C— properly.
C
      cexab = kab * sdb
C
C— Update aqueous/biophase phase mole exchange.
C
      if (.not.lctrl(16)) then
C
C— Set the aqueous and bio phase mole fractions to zero if they
C— are nonpositive.
C
        if(xmf(ia+i).le.zer0) then
          xmfa = zer0
        else
          xmfa = xmf(ia+i)
        end if
        if(xmf(ib+i).le.zer0) then
          xmfb = zer0
        else
          xmfb = xmf(ib+i)
        end if
        pex(ipt2+istk) = pex(ipt2+istk)
+         - cexab * ( xmfa - xmfb )
C
C— Update aqueous/biophase phase mass exchange.
C

```



```

+       pex(ipt2x6+istk) = pex(ipt2x6+istk)
-       - cmw(ic) * cexab * (xmfa - xmf b)
C
C— Update biophase/aqueous phase mole exchange.
C
+       pex(ipt2x4+istk) = pex(ipt2x4+istk)
+       + cexab * ( xmfa - xmf b )
C
C— Update biophase/aqueous phase mass exchange.
C
+       pex(ipt2x9+istk) = pex(ipt2x9+istk)
+       + cmw(ic) * cexab * (xmfa - xmf b)
C
C— Update the right hand side terms for the component transport
C— equations.
C
+       rhsex(ias+istk) = rhsex(ias+istk)
+       + cexab * xmf(ib+i)
+       rhsex(ibs+istk) = rhsex(ibs+istk)
+       + cexab * (xmfa - xmf b)
C
C— Do not consider aqueous/biophase phase mole exchange, however
C— include the bioreaction loss.
C
+       end if
+       end if
100 continue
C
C— Now calculate mass exchange for oxygen. Mass exchange
C— for oxygen is only considered for bioventing simulations. First
C— compute pointers.
C
if(lctrl(3)) then
+       ico = ipt(15) + 2
C
C— Consider moist gas phase.
C
if(icp(ipt(13)+2).eq.ico) then
+       igo = (ipt(13)+1)*ipt(1)
+       igos = (ipt(13)+1)*ipt2
C
C— Consider dry gas phase.
C
else
+       igo = ipt(13)*ipt(1)
+       igos = ipt(13)*ipt2
+       end if
+       iao = ipt(9) + (ipt(14)+1)*ipt(1)
+       ibo = ipt(12) + ipt(17)*ipt(1)
+       iaos = ipt(3)*ipt2 + (ipt(14)+1)*ipt2
+       ibos = (ipt(3)+ipt(4)+ipt(5)+ipt(6))*ipt2 + ipt(17)*ipt2
C
C— Iterate over the nodes.
C
do 200 i = 1, ipt(1)
+       if(lctrl(10)) then
+         cvpt = cvp(ico)
+         casolt = casol(ico)
+       else
+         itemp = (ico-1)*ipt(89)+idepth(i)
+         itemp2 = (ico-1)*ipt(89)+3*ipt(88)+idepth(i)
+         cvpt = cvp(ico) + dtemp(itemp)
+         casolt = casol(ico) + dtemp(itemp2)
+       end if
C
C— PUT USER DEFINED AQUEOUS/BIPHASE MASS TRANS-
C— FER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KABO TO UTILIZE BUILT-
C— IN CONTROL.
C
+       if(limab) then
+         kabo = kex(5*(ic-1)+4)
C
C— Aqueous/biophase control on mass transfer coefficient with the
C— greater of the aqueous phase advective velocity or
C— aqueous phase diffusional velocity.
C
+       kabo = dmin1( kabo, -( dmax1( qavea(i),

```

```

+       cmdif(2*ic)/xl(i)**2 ) * dlog(kmax(ic+3*ipt(65))))
C
C— PUT USER DEFINED AQUEOUS/BIPHASE MASS TRANS-
C— FER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KABO TO BYPASS BUILT-
C— IN CONTROL.
C
+       else
+         kabo = kex(5*(ic-1)+4)
+       end if
C
C— Turn off aqueous/biophase oxygen exchange if the aqueous phase
C— oxygen mole fraction is negative.
C
+       if(xmf(iao+i).lt.zer0) kabo = zer0
C
C— PUT USER DEFINED AQUEOUS/GAS MASS TRANSFER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KAGO TO UTILIZE BUILT-
C— IN CONTROL.
C
+       if(limag) then
+         kago = kex(5*(ic-1)+1)
C
C— Gas/aqueous control on mass transfer coefficient with the greater
C— of the gas or aqueous phase advective velocity or diffusional
C— velocity.
C
+       kago = dmin1( kago, -( dmax1( qaveg(i)
+         , qavea(i), cmdif(2*ic-1)/xl(i)**2, cmdif(2*ic)
+         /xl(i)**2 ) * dlog(kmax(ic)) ) )
C
C— PUT USER DEFINED AQUEOUS/GAS MASS TRANSFER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KAGO TO BYPASS BUILT-
C— IN CONTROL.
C
+       else
+         kago = kex(5*(ic-1)+1)
+       end if
C
C— Turn off gas/aqueous oxygen exchange if the gas phase
C— oxygen mole fraction is negative.
C
+       if(xmf(igo+i).lt.zer0) kago = zer0
C
C— Calculate constants. Use aqueous phase properties for the biophase
C
+       keqag = patm*casolt/cvpt
+       sdaq = den(ipt1+i)
C
C— This form assumes that exchange is into a biophase, the size of
C— which is determined by the maximum allowable biomass. Aqueous
C— phase material properties are used to determine the volume of the
C— biophase.
C
+       sdb = sdaq
C
C— Update left hand side term for the aqueous phase equations.
C— First consider aqueous/gas oxygen mole exchange.
C
+       cexago = sdaq * kago
+       do 200 ii = 0, nodept(i+1)-nodept(i)-1
+         istk = ii + nodept(i)
+         sgcont = sat(istk)
+         if(sgcont.lt.sgtest) cexago = zer0
+         cex(iaos+istk) = cex(iaos+istk) + cexago
C
C— Then consider aqueous/biophase oxygen mole exchange. The biomass
C— term accounts for the volume of the reactive phase properly.
C
+       cexabo = kabo * sdb
C
C— Update aqueous/gas phase mole exchange.
C
+       pexago = cexago * (keqag*xmf(igo+i) - xmf(iao+i))
+       pex(ipt2+istk) = pex(ipt2+istk) + pexago
C
C— Update aqueous/gas phase mass exchange.
C

```

```

      pex(ipt2x6+istk) = pex(ipt2x6+istk) + pexago*cmw(ico)
C
C— Update aqueous/biophase phase mole exchange.
C
      if(.not.lctrl(16)) then
C
C— Set the aqueous and bio phase mole fractions to zero if they
C— are nonpositive.
C
        if(xmf(jao+i).le.zer0) then
          xmfa = zer0
        else
          xmfa = xmf(jao+i)
        end if
        if(xmf(ibo+i).le.zer0) then
          xmfb = zer0
        else
          xmfb = xmf(ibo+i)
        end if
        pexabo = cexabo * (xmfa - xmfb)
        pex(ipt2+istk) = pex(ipt2+istk) - pexabo
C
C— Update aqueous/biophase phase mole exchange.
C
        pex(ipt2x6+istk) = pex(ipt2x6+istk) - pexabo*cmw(ico)
C
C— Update biophase/aqueous phase mole exchange.
C
        pex(ipt2x4+istk) = pex(ipt2x4+istk) + pexabo
C
C— Update biophase/aqueous phase mass exchange.
C
        pex(ipt2x9+istk) = pex(ipt2x9+istk) + pexabo
C
C— Do not consider aqueous/biophase phase mole exchange, however
C— include the bioreaction loss.
C
        end if
C
C— Update gas/aqueous phase mole exchange.
C
        pex(istk) = pex(istk) - pexago
C
C— Update the right hand side terms for the component transport
C— equations.
C
        rhsex(igos+istk) = rhsex(igos+istk) - pexago
        rhsex(iaos+istk)
      +   = rhsex(iaos+istk) + cexago * keqag * xmf(igo+i)
        if(.not.lctrl(16)) then
          rhsex(iaos+istk) = rhsex(iaos+istk)
        +   + cexabo * xmfb
          rhsex(ibos+istk) = rhsex(ibos+istk)
        +   + cexabo * (xmfa - xmfb)
        end if
200      continue
C
C— Now calculate mass exchange for the nutrient. Mass exchange for
C— nutrient is only considered for bioventing simulations. First
C— compute pointers.
C
        if(lctrl(9)) then
          icn = ipt(65)
          ign = ipt(9) - ipt(1)
          ian = ipt(10) - ipt(1)
          ibn = ipt(12) + ipt(1) * (ipt(7)-2)
          igns = (ipt(3)-1)*ipt2
          ians = (ipt(3)+ipt(4)-1)*ipt2
          ibns = (ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(7)-2)*ipt2
C
C— Iterate over the nodes.
C
          do 210 i = 1, ipt(1)
            if(lctrl(10)) then
              cvpt = cvp(icn)
              casolt = casol(icn)
            else
              itemp = (icn-1)*ipt(89)+3*ipt(88)+idepth(i)

```

```

              casolt = casol(icn) + dtemp(itemp)
            end if
C
C— PUT USER DEFINED AQUEOUS/BIPHASE MASS TRANSFER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KABN TO UTILIZE BUILT-IN CONTROL.
C
            if(limab) then
              kabn = kex(5*(ic-1)+4)
C
C— Aqueous/biophase control on mass transfer coefficient with the
C— greater of the aqueous phase advective velocity or
C— aqueous phase diffusional velocity.
C
              kabn = dmin1( kabn, -( dmax1( qavea(i),
      +   cmdif(2*ic)/xl(i)**2)*dlog(kmax(ic+3*ipt(65))))))
C
C— PUT USER DEFINED AQUEOUS/BIPHASE MASS TRANSFER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KABN TO BYPASS BUILT-IN CONTROL.
C
            else
              kabn = kex(5*(ic-1)+4)
            end if
C
C— Turn off aqueous/biophase nutrient exchange if the aqueous phase
C— mole fraction is negative.
C
            if(xmf(jan+i).lt.zer0) kabn = zer0
C
C— PUT USER DEFINED GAS/AQUEOUS MASS TRANSFER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KAGN TO UTILIZE BUILT-IN CONTROL.
C
            if(limag) then
              kagn = kex(5*(ic-1)+1)
C
C— Gas/aqueous control on mass transfer coefficient with the greater
C— of the gas or aqueous phase advective velocity or diffusional
C— velocity.
C
              kagn = dmin1( kagn, -( dmax1( qaveg(i)
      +   , qavea(i), cmdif(2*ic-1)/xl(i)**2, cmdif(2*ic)
      +   /xl(i)**2) * dlog(kmax(ic)) ) )
C
C— PUT USER DEFINED GAS/AQUEOUS MASS TRANSFER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KAGN TO BYPASS BUILT-IN CONTROL.
C
            else
              kagn = kex(5*(ic-1)+1)
            end if
C
C— Calculate constants. Use aqueous phase properties for the biophase
C
            keqag = patm*casolt/cvpt
            sdaq = den(ipt(1)+1)
C
C— This form assumes that exchange is into a biophase, the size of
C— which is determined by the maximum allowable biomass. Aqueous
C— phase material properties are used to determine the volume of the
C— biophase.
C
            sdb = sdaq
C
C— Update left hand side term for the aqueous phase equations.
C— First consider aqueous/gas nutrient mole exchange.
C
            cexagn = sdaq * kagn
            do 210 ii = 0, nodept(i+1)-nodept(i)-1
              istk = ii + nodept(i)
              sgcont = sat(istk)
              if(sgcont.lt.sgtest) cexagn = zer0
              cex(ians+istk) = cex(ians+istk) + cexagn
C
C— Then consider aqueous/biophase nutrient mole exchange. The

```

```

C— biomass term accounts for the volume of the reactive phase
C— properly.
C
      cexabn = kabn * sdb
C
C— Update aqueous/gas phase mole exchange.
C
      pexagn = cexagn * (keqag*xmf(ign+i) - xmf(ian+i))
      pex(ipt2+istk) = pex(ipt2+istk) + pexagn
C
C— Update aqueous/gas phase mass exchange.
C
      pex(ipt2x6+istk) = pex(ipt2x6+istk)+pexago*cmw(icn)
C
C— Update aqueous/biophase phase mole exchange.
C
      if(.not.lctrl(16)) then
C
C— Set the aqueous and bio phase mole fractions to zero if they
C— are nonpositive.
C
          if(xmf(ian+i).le.zer0) then
              xmfa = zer0
          else
              xmfa = xmf(ian+i)
          end if
          if(xmf(ibn+i).le.zer0) then
              xmfb = zer0
          else
              xmfb = xmf(ibn+i)
          end if
          pexabn = cexabn * (xmfa - xmfb)
          pex(ipt2+istk) = pex(ipt2+istk) - pexabn
C
C— Update aqueous/biophase phase mole exchange.
C
      +
      pex(ipt2x6+istk)
      = pex(ipt2x6+istk) - pexabn * cmw(icn)
C
C— Update biophase/aqueous phase mole exchange.
C
      pex(ipt2x4+istk) = pex(ipt2x4+istk) + pexabn
C
C— Update biophase/aqueous phase mass exchange.
C
      pex(ipt2x9+istk) = pex(ipt2x9+istk) + pexabn
      end if
C
C— Update gas/aqueous phase mole exchange.
C
      pex(istk) = pex(istk) - pexagn
C
C— Update the right hand side terms for the component transport
C— equations.
C
      rhsex(igns+istk) = rhsex(igns+istk) - pexagn
      rhsex(ians+istk) = rhsex(ians+istk)
      +
      + cexagn * keqag * xmf(ign+i)
      if(.not.lctrl(16)) then
          rhsex(ians+istk) = rhsex(ians+istk)
          +
          + cexabn * xmfb
          rhsex(ibns+istk) = rhsex(ibns+istk)
          +
          + cexabn * (xmfa - xmfb)
      end if
210 continue
      end if
      end if
C
C— Calculate water mass exchange between the aqueous and gas phases.
C— This is done only when both mobile phases are present and water is
C— present in the gas phase. The presence of NAPL has no effect
C— on this term. Gas phase mole fractions controls water mole
C— exchange.
C
      icw = ipt(15) + 1
      icl = icp(ipt(13)+1)
      if((icl.eq.icw).and.(ipt(3)+ipt(4)).ne.ipt(3)) then
          igw = ipt(13)*ipt(1)

```

```

      iaw = ipt(9) + ipt(14)*ipt(1)
      igws = ipt(13)*ipt2
      do 300 i = 1, ipt(1)
          if(lctrl(10)) then
              cvpt = cvp(icw)
          else
              itemp = (icw-1)*ipt(89)+idepth(i)
              cvpt = cvp(icw) + dtemp(itemp)
          end if
          keqw = cvpt/patm
C
C— PUT USER DEFINED GAS/AQUEOUS MASS TRANSFER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KGAW TO UTILIZE BUILT-
C— IN CONTROL.
C
          if(limag) then
              kgaw = kex(5*(ic-1)+1)
C
C— Gas/aqueous control on mass transfer coefficient with the greater
C— of the gas or aqueous phase advective velocity or diffusional
C— velocity.
C
              kgaw = dmin1( kgaw, -( dmax1( qaveg(i)
              +
              , qavea(i), cmdif(2*ic-1)/xl(i)**2, cmdif(2*ic)
              +
              /xl(i)**2 ) * dlog(kmax(ic)) ) )
C
C— PUT USER DEFINED GAS/AQUEOUS MASS TRANSFER COEFFICIENT
C— FUNCTION HERE. ASSIGN IT TO KGAW TO BYPASS BUILT-
C— IN CONTROL.
C
          else
              kgaw = kex(5*(ic-1)+1)
          end if
C
C— Update constants.
C
      sdbg = den(i)
      sdaq = den(ipt1+i)
C
C— Update left hand side terms for gas phase equations.
C
      cexagw = sdbg * kgaw
      do 300 ii = 0, nodept(i+i)-nodept(i)-1
          istk = ii + nodept(i)
          sgcont = sat(istk)
          if(sgcont.lt.sgstest) cexagw = zer0
          cex(igws+istk) = cex(igws+istk) + cexagw
C
C— Update aqueous/gas phase mole exchange.
C
      pexagw = cexagw * (keqw*xmf(iaw+i) - xmf(igw+i))
      pex(ipt2+istk) = pex(ipt2+istk) - pexagw
C
C— Update aqueous/gas phase mass exchange.
C
      pex(ipt2x6+istk) = pex(ipt2x6+istk) - pexagw*cmw(icw)
C
C— Update gas/aqueous phase mole exchange.
C
      pex(istk) = pex(istk) + pexagw
C
C— Update gas/aqueous phase mass exchange.
C
      pex(ipt2x5+istk) = pex(ipt2x5+istk) + pexagw*cmw(icw)
C
C— Update the right hand side terms for the gas phase transport
C— equations. The aqueous phase does not have a water transport
C— equation since the effect of water mass exchange is accounted
C— for through the phase exchange terms.
C
      rhsex(igws+istk) = rhsex(igws+istk)
      +
      + cexagw * keqw * xmf(iaw+i)
300 continue
      end if
      return
      end

```

Subroutine - napls.f

```

C-----
C
C NAPLS.f - Subroutine which updates the NAPL
C saturations using the finite element method.
C
C Arguments: iconv - integer flag for global convergence
C
C Required Control Flags:
C
C ((16) - convergence criterion for immobile phases
C
C Control Flags computed internally in routine:
C
C ipt(38) - flag specifying time step modification
C-----
subroutine NAPLS(iconv)
include 'dimen.inc'
C
C Declare and define common block variables.
C
common /cb1/ matel(nelmx),node1(nel3),nodept(nnmx),nelpt(nel3),
+ matpt(nn6)
common /cb1e/ aby12(nelmx),aby30(nelmx)
common /cb3/ sat(nnstk3)
common /cb3b/ sat(nnstk3)
common /cb6b/ por(nelmx),srw(nnstk)
common /cb10/ den(nn6)
common /cb10b/ dden(nn6),pmwt(nn3),dent(nn6)
common /cb11/ pex(nns10),rxnp(nn2)
common /cb40/ a(icnl),rhs(nsolve),w(icnl)
C
C Set pointers.
C
ipt(38)=0
ins = ipt(49)
insm = ipt(54)
C
C Begin iterations.
C
do 400 itnapi = 1, ipt(33)
do 110 i = 1, ipt(2)
rhs(i) = zer0
110 a(i) = zer0
do 120 i=1, ipt(0)
C
C Set the pointers to the local nodes i1, i2, and i3. The
C postscript s is for stacked local nodes, and p is for phase
C Compute constants.
C
iel3 = i*3
iel1 = iel3-2
iel2 = iel3-1
i1 = node1(iel1)
i1s = nodept(i1)+nelpt(iel1)
i1psm = insm + i1s
i2 = node1(iel2)
i2s = nodept(i2)+nelpt(iel2)
i2psm = insm + i2s
i3 = node1(iel3)
i3s = nodept(i3)+nelpt(iel3)
i3psm = insm + i3s
d1 = rone
d2 = rone
d3 = rone
term = por(i)*aby30(i)/t(8)
C
C Compute the mass matrix.
C
a11 = term*( 3.0d0*d1 + d2 + d3 )
a12 = term*( d1 + d2 + d3/2.0d0 )
a13 = term*( d1 + d2/2.0d0 + d3 )
a21 = a12
a22 = term*( d1 + 3.0d0*d2 + d3 )

```

```

a23 = term*( d1/2.0d0 + d2 + d3 )
a31 = a13
a32 = a23
a33 = term*( d1 + d2 + 3.0d0*d3 )
C
C Lump the 'mass' matrix.
C
a11 = a11 + a12 + a13
a12 = zer0
a13 = zer0
a22 = a21 + a22 + a23
a21 = zer0
a23 = zer0
a33 = a31 + a32 + a33
a31 = zer0
a32 = zer0
C
C Now compute the right hand side terms.
C
term12 = por(i)*aby12(i)
if (ipt(5).gt.1) then
rhs1 = pex(i1psm)
rhs2 = pex(i2psm)
rhs3 = pex(i3psm)
else
rhs1 = pex(i1psm)
rhs2 = pex(i2psm)
rhs3 = pex(i3psm)
end if
f1 = term12 * ( 2.0d0*rhs1 + rhs2 + rhs3 )
f2 = term12 * ( rhs1 + 2.0d0*rhs2 + rhs3 )
f3 = term12 * ( rhs1 + rhs2 + 2.0d0*rhs3 )
C
C Assemble global matrix and right hand side vector. Note that
C the left hand side matrix is diagonal due to mass lumping and
C the solution is therefore explicit.
C
a(i1s) = a(i1s) + a11
a(i2s) = a(i2s) + a22
a(i3s) = a(i3s) + a33
rhs(i1s) = rhs(i1s) + f1
rhs(i2s) = rhs(i2s) + f2
rhs(i3s) = rhs(i3s) + f3
120 continue
C
C Solve for saturations. Skip any node where the NAPL saturation
C is zero. Determine the max norm of the updated solution.
C
dsat = zer0
satmax = smino
do 150 i = 1, ipt(1)
do 150 ii = 0, nodept(i+1)-nodept(i)-1
imat = nodept(i)+ii
isat = imat + ins
satold=sat(isat)
if(sat(isat).gt.smino) then
if(a(imat).ne.zer0) then
sat(isat) = (dent(ipt(1)*5+i)*satt(isat)
+ rhs(imat)/a(imat))/den(ipt(1)*5+i)
satmax = dmax1(satmax,dabs(sat(isat)))
dsat = dmax1(dsat,dabs(sat(isat)-satold))
end if
else
C
C Set NAPL saturation equal to zero when it falls below zero.
C
sat(isat) = zer0
end if
150 continue
C
C Return when NAPL has only one component.
C
if(ipt(5).eq.1) then
ipt(38) = 1

```



```

b31 = b13
b32 = b23
b33 = term30*( pex1 + pex2 + 3.0d0*pex3 )
C
C--- Lump the 'mass' matrix.
C
if(ictrl(8)) then
  a11 = a11 + a12 + a13
  a12 = zer0
  a13 = zer0
  a22 = a21 + a22 + a23
  a21 = zer0
  a23 = zer0
  a33 = a31 + a32 + a33
  a31 = zer0
  a32 = zer0
end if

C
C--- Now compute the right hand side terms.
C
term12 = por(i)*aby12(i)
rhs1 = cex(i1cs)
rhs2 = cex(i2cs)
rhs3 = cex(i3cs)
f1 = term12 * ( 2.0d0*rhs1 + rhs2 + rhs3 )
f2 = term12 * ( rhs1 + 2.0d0*rhs2 + rhs3 )
f3 = term12 * ( rhs1 + rhs2 + 2.0d0*rhs3 )

C
C--- Assemble global matrix and right hand side vector in banded form.
C
irow1 = (i1-1)*nbw(1)
irow2 = (i2-1)*nbw(1)
irow3 = (i3-1)*nbw(1)
icol11 = 1 + nbw(0)
icol12 = icol11 + (i2 - i1)
icol13 = icol11 + (i3 - i1)
icol22 = icol11
icol21 = icol22 + (i1 - i2)
icol23 = icol22 + (i3 - i2)
icol33 = icol11
icol31 = icol33 + (i1 - i3)
icol32 = icol33 + (i2 - i3)
ab11 = a11 + t(10)*b11
ab12 = a12 + t(10)*b12
ab13 = a13 + t(10)*b13
ab21 = a21 + t(10)*b21
ab22 = a22 + t(10)*b22
ab23 = a23 + t(10)*b23
ab31 = a31 + t(10)*b31
ab32 = a32 + t(10)*b32
ab33 = a33 + t(10)*b33
a(irow1 + icol11) = a(irow1 + icol11) + ab11
a(irow1 + icol12) = a(irow1 + icol12) + ab12
a(irow1 + icol13) = a(irow1 + icol13) + ab13
a(irow2 + icol21) = a(irow2 + icol21) + ab21
a(irow2 + icol22) = a(irow2 + icol22) + ab22
a(irow2 + icol23) = a(irow2 + icol23) + ab23
a(irow3 + icol31) = a(irow3 + icol31) + ab31
a(irow3 + icol32) = a(irow3 + icol32) + ab32
a(irow3 + icol33) = a(irow3 + icol33) + ab33
rhs(i1) = rhs(i1) + f1 - b11 * xmf(i1c)
+ - b12 * xmf(i2c) - b13 * xmf(i3c)
rhs(i2) = rhs(i2) + f2 - b21 * xmf(i1c)
+ - b22 * xmf(i2c) - b23 * xmf(i3c)
rhs(i3) = rhs(i3) + f3 - b31 * xmf(i1c)
+ - b32 * xmf(i2c) - b33 * xmf(i3c)
220 continue
C
C--- Account for nodes at which the NAPL and/or component has been
C--- removed by specifying first type boundary conditions.
C
do 225 i = 1, ipt(1)

```

```

skeep = zer0
do 226 ii = 0, nodept(i+1)-nodept(i)-1
226   skip = skip + sat(nodept(i)+ii+ins)
   if(skip.lt.zer0) then
     rhs(i) = zer0
     nbw0 = nbw(0)
     nbw1 = nbw(1)
     do 227 j = 1,nbw(1)
       a((i-1)*nbw1+j) = zer0
227     continue
     a((i-1)*nbw1+1+nbw0) = rone
   end if
225 continue
C
C--- Collapse full matrix into sparse form used by Harwell. Also
C--- scale array by dividing rows through by the diagonal value.
C
ia = 0
do 230 irow = 1, ipt(1)
  nrow = (irow-1)*nbw(1)
  aii = rone / a(nrow+1+nbw(0))
  rhs(irow) = rhs(irow) * aii
  do 230 icol = 1,nbw(1)
    if (a(nrow+icol) .ne. zer0) then
      ia = ia + 1
      a(ia) = a(nrow+icol) * aii
      irn(ia) = irow
      icn(ia) = icol+irow-nbw(0)-1
    endif
230 continue
C
C--- Solve the linear system using Harwell routines.
C
call ma28ad(ipt(1),ia,a,icnl,irn,irnl,icn,u,ikeep,iw,w,iflag)
if (iflag.lt.0) then
  write (ipt(28),*) 'iflag return from harwell is ',iflag
  write (ipt(28),*) 'in naplx component is: '
+ ,cname(icnapl)
endif
call ma28cd (ipt(1),a,icnl,icn,ikeep,rhs,w,mtype)
C
C--- Update the solution and determine the max norm of the updated
C--- solution.
C
dxmf = zer0
xmfmax = xround
do 250 i = 1, ipt(1)
  xmfold=xmf(icpt+i)
  skip = zer0
  do 240 ii = 0,nodept(i+1)-nodept(i)-1
240   skip = skip + sat(nodept(i)+ii+ins)
   if(skip.gt.smino) then
     xmf(icpt+i) = xmf(icpt+i) + rhs(i)
     if(xmf(icpt+i).lt.xmino) xmf(icpt+i) = xmino
     xmfmax = dmax1(xmfmax,dabs(xmf(icpt+i)))
     dxmf = dmax1(dxmf,dabs(xmf(icpt+i)-xmfold))
     goto 250
   else
     xmf(icpt+i) = zer0
   end if
250 continue
C
C--- Check convergence with the relative change in state variable. Do
C--- not allow NAPL mole fractions less than xmino to control
C--- convergence.
C
if(xmfmax.gt.xmino.and.dxmf/xmfmax.gt.t(16))
+   iconv = iconv + 1
200 continue
return
end

```

Subroutine - prnt.f

```

C
C
C PRNT.f - outputs current values of selected variables
C defined in the logical array 'lprnt'.
C
C Required Control Flags:
C
C lctrl(1) - logical variable controlling presence of flow
C solution
C lctrl(1) = .true. - compute flow solution
C lctrl(1) = .false. - skip flow solution
C lctrl(2) - logical variable controlling presence of transport
C solution
C lctrl(2) = .true. - compute transport solution
C lctrl(2) = .false. - skip transport solution
C lctrl(3) - logical variable controlling inclusion of
C biodegradation
C lctrl(3) = .true. - include biodegradation
C lctrl(3) = .false. - neglect biodegradation
C lctrl(18) - logical variable denoting type of q calculation
C lctrl(18) = .true. - nodal FEM q
C lctrl(18) = .false. - element average q
C lctrl(19) - logical variable controlling type of sorption
C lctrl(19) = .true. - two compartment sorption
C lctrl(19) = .false. - single compartment
C sorption
C lctrl(23) - logical variable controlling the printing of
C contour plot data
C lctrl(23) = .true. - print contour plot data
C lctrl(23) = .false. - do not print contour
C plot data
C
C
C subroutine PRNT (its)
C include 'dimen.inc'
C character*20 infile(4),outpre,outfile(8+ncmp)
C character*10 cname(ncmp)
C common /cb1/ matel(nelm),node1(nel3),nodept(nnm),nelpt(nel3),
C + matpt(nn6)
C common /cb1c/ xnode(nnm),znode(nnm),rbar(nelm),area(nelm)
C common /cb2/ p(nn3)
C common /cb2c/ q(nel4)
C common /cb3/ sat(nnstk3)
C common /cb6b/ por(nelm),srw(nnstk)
C common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
C + chen(ncmp),casol(ncmp),cmdif(ncmp2)
C common /cb9/ xmf(nmf)
C common /cb10/ den(nn6)
C common /cb10b/ dden(nn6),pmwt(nn3),dent(nn6)
C common /cb11/ pex(nns10),rxnp(nn2)
C common /cb64/ bok(nbcmp),bom(nbcmp),krt(ncmp)
C common /cb64b/ bsden(nmbk)
C common /cb86/ str1(ncmpp5),str0(ncmpp5),cmf(ncmpp5),csink(ncmpp5)
C + ,cwsink(ncmpp5),csflux(ncmpp5),cmass1(ncmpp5),cmass0(ncmpp5)
C + ,cphex(ncmpp5),crsink(ncmpp5),tmass1,tmass0
C common /cb90/ infile,outpre,outfile
C common /cb91/ cname
C
C
C Dimension local arrays. rcont, contr, and xm are dimensioned
C with two less than maximum number of output columns divided by
C 10. Currently dimensioned to support 132 column output maximum.
C
C dimension tp(nnstk3),rcont(11),xm(11)
C character contr(11)*10
C
C
C Nout + 2 is the number of output columns desired in outpre.con
C Currently set to 5 columns for 80 column output. Set nout to 6
C for 132 column output.
C
C data nout / 3 /
C
C Print header and time.
C
C write (21,500) its,t(9),t(9)/60.0d0,t(9)/3600.0d0,t(9)/.8640d5

```

```

C
C Print nodal gas phase pressure.
C
C if (lprnt(9)) then
C   Do i = 1,ipt(1)
C     tp(i) = (patm+p(i)) / patm
C   EndDo
C   call fprnt (21,tp(1),ipt(1),
C + '@ Nodal gas phase pressure (atm)')
C endif
C
C Print nodal aqueous phase pressure.
C
C if (lprnt(10)) then
C   Do i = 1,ipt(1)
C     tp(i) = (patm+p(ipt(1)+i)) / patm
C   EndDo
C   call fprnt (21,tp(1),ipt(1),
C + '@ Nodal aqueous phase pressure (atm)')
C endif
C
C Print nodal capillary pressure.
C
C if (lprnt(11)) call fprnt (21,p(2*ipt(1)+1),ipt(1),
C + '@ Nodal gas/aqueous capillary pressure (Pa)')
C
C Print gas phase mass or molar density.
C
C if (lprnt(12)) then
C   if (lprnt(8)) then
C     call fprnt (21,den(1),ipt(1),
C + '@ Gas phase molar density')
C   else
C     call fprnt (21,den(3*ipt(1)+1),ipt(1),
C + '@ Gas phase mass density')
C   end if
C
C Print aqueous phase mass or molar density.
C
C if (lprnt(13)) then
C   if (lprnt(8)) then
C     call fprnt (21,den(ipt(1)+1),ipt(1),
C + '@ Aqueous phase molar density')
C   else
C     call fprnt (21,den(4*ipt(1)+1),ipt(1),
C + '@ Aqueous phase mass density')
C   end if
C
C Print NAPL phase mass or molar density.
C
C if (lprnt(14)) then
C   if (lprnt(8)) then
C     call fprnt (21,den(2*ipt(1)+1),ipt(1),
C + '@ NAPL phase molar density')
C   else
C     call fprnt (21,den(5*ipt(1)+1),ipt(1),
C + '@ NAPL phase mass density')
C   end if
C
C Print gas phase component output.
C
C if (lprnt(15)) then
C   do 10 i=1,ipt(3)
C     ic=icp(i)
C     iptc=(i-1)*ipt(1)
C     if(lprnt(8)) then
C       call fprnt(21,xmf(iptc+1),ipt(1),
C + '@ Nodal gas phase component mole fractions: '
C + //cname(ic))
C     else
C       do 15 ii = 1, ipt(1)

```

```

15      tp(ii) = xmf(iptc+ii) * den(ii) * cmw(ic)
      call fprnt(21,tp(1),ipt(1),
+       '@ Nodal gas phase component concentrations (g/l): '
+       //cname(ic))
      end if
10     continue
      endif
C
C— Print aqueous phase component output.
C
      if (lprnt(16)) then
      do 20 i=1+ipt(3)+ipt(4)
      ic=icp(i)
      iptc=(i-1)*ipt(1)
      if (lprnt(8)) then
      call fprnt(21,xmf(iptc+1),ipt(1),
+       '@ Nodal aqueous phase component mole fractions: '
+       //cname(ic))
      else
      do 25 ii = 1, ipt(1)
25      tp(ii) = xmf(iptc+ii) * den(ipt(1)+ii) * cmw(ic)
      call fprnt(21,tp(1),ipt(1),
+       '@ Nodal aqueous phase component concentrations (g/L): '
+       //cname(ic))
      end if
20     continue
      endif
C
C— Print NAPL phase mole fractions.
C
      if (lprnt(17)) then
      do 30 i=1+ipt(3)+ipt(4),ipt(3)+ipt(4)+ipt(5)
      ic=icp(i)
      iptc=(i-1)*ipt(1)
      if (lprnt(8)) then
      call fprnt(21,xmf(iptc+1),ipt(1),
+       '@ Nodal NAPL phase component mole fractions: '
+       //cname(ic))
      else
      do 35 ii = 1, ipt(1)
35      tp(ii) = xmf(iptc+ii) * den(ipt(4)+ii) * cmw(ic)
      call fprnt(21,tp(1),ipt(1),
+       '@ Nodal NAPL phase component concentrations (g/L): '
+       //cname(ic))
      end if
30     continue
      endif
C
C— Print solid phase mass loadings.
C
      if (lprnt(18)) then
      do 40 i=1+ipt(3)+ipt(4)+ipt(5),ipt(3)+ipt(4)+ipt(5)+ipt(6)
      ic=icp(i)
      iptc=(i-1)*ipt(1)
      if (lprnt(19)) then
      if (i-ipt(3)-ipt(4)-ipt(5).eq.1) then
      call fprnt(21,xmf(iptc+1),ipt(1),
+       '@ Nodal slow solid phase component mass '
+       //loading gm/gm (slow compartment): '
+       //cname(ic))
      do 45 ii = 1, ipt(1)
45      tp(ii) = xmf(iptc+ii)*(rone-xden)
      call fprnt(21,tp(1),ipt(1),
+       '@ Nodal slow solid phase component mass '
+       //loading gm/gm(bulk): '
+       //cname(ic))
      else if (i-ipt(3)-ipt(4)-ipt(5).eq.2) then
      ic = icp(i-1)
      call fprnt(21,xmf(iptc+1),ipt(1),
+       '@ Nodal fast solid phase component mass '
+       //loading gm/gm(fast compartment): '
+       //cname(ic))
      do 50 ii = 1, ipt(1)
50      tp(ii) = xmf(iptc+ii)*xden
      call fprnt(21,tp(1),ipt(1),
+       '@ Nodal fast solid phase component mass '
+       //loading gm/gm(bulk): '
+       //cname(ic))

```

```

      do 55 ii = 1, ipt(1)
55      tp(ii) = xmf(iptc+ii) * xden
+       + xmf(iptc-ipt(1)+ii) * (rone - xden)
      call fprnt(21,tp(1),ipt(1),
+       '@ Nodal total solid phase component mass '
+       //loading gm/gm(bulk): '
+       //cname(ic))
      end if
      else
      call fprnt(21,xmf(iptc+1),ipt(1),
+       '@ Nodal solid phase component mass '
+       //loading gm/gm(bulk): '
+       //cname(ic))
      end if
40     continue
      endif
C
C— Print biophase phase mole fractions.
C
      if (lprnt(19)) then
      do 60 i=1+ipt(3)+ipt(4)+ipt(5)+ipt(6),
+       ipt(3)+ipt(4)+ipt(5)+ipt(6)+ipt(7)-1
      ic=icp(i)
      iptc=(i-1)*ipt(1)
      if (lprnt(8)) then
      call fprnt(21,xmf(iptc+1),ipt(1),
+       '@ Nodal bio-phase component mole fractions: '
+       //cname(ic))
      else
      do 65 ii = 1, ipt(1)
65      tp(ii) = xmf(iptc+ii) * den(ipt(1)+ii) * cmw(ic)
      call fprnt(21,tp(1),ipt(1),
+       '@ Nodal bio-phase component concentrations (g/L): '
+       //cname(ic))
      end if
60     continue
      call fprnt(21,xmf(ipt(1)+iptc+1),ipt(1),
+       '@ Nodal bio-phase biomass concentration')
      endif
c
c— Compute and print an element average total organic soil
c— concentration.
c
      If (lprnt(29).or.lcon(18)) then
      Do jel = 1, ipt(0) ! loop over number of elements
      n3 = (jel-1)*3
      x1 = rbar(jel) * area(jel) * third
      x = x1 * por(jel)
      ipt1x3 = ipt(41)
      i1 = node1(n3+1) ! element node numbers
      i2 = node1(n3+2)
      i3 = node1(n3+3)
      ig1 = i1 ! node1 gas phase storage locations
      ig2 = i2
      ig3 = i3
      ia1 = ipt(1)+ig1 ! node1 aqueous phase storage locations
      ia2 = ipt(1)+ig2
      ia3 = ipt(1)+ig3
      in1 = ipt(1)+ia1 ! node1 napl phase storage locations
      in2 = ipt(1)+ia2
      in3 = ipt(1)+ia3
      ndstk1 = nodept(i1) + nelpt(n3+1) ! node position in stack
      ndstk2 = nodept(i2) + nelpt(n3+2)
      ndstk3 = nodept(i3) + nelpt(n3+3)
      ig1s = ndstk1 ! gas phase stacked node numbers
      ig2s = ndstk2
      ig3s = ndstk3
      ia1s = ipt(2)+ig1s ! aqueous phase stacked node numbers
      ia2s = ipt(2)+ig2s
      ia3s = ipt(2)+ig3s
      in1s = ipt(2)+ia1s ! napl phase stacked node numbers
      in2s = ipt(2)+ia2s
      in3s = ipt(2)+ia3s
      Wnapl = x * ( ! mass of napl
+       den(ipt1x3+in1) * sat(in1s) +
+       den(ipt1x3+in2) * sat(in2s) +
+       den(ipt1x3+in3) * sat(in3s) )
      Onapl = Wnapl ! mass of organic in napl

```



```

Waq = x * ( ! mass of aqueous phase
+ den(ipt1x3+ia1) * sat(ia1s) +
+ den(ipt1x3+ia2) * sat(ia2s) +
+ den(ipt1x3+ia3) * sat(ia3s) )
Oaq = zero ! mass of organic in aqueous phase
Do i = 1, ipt(14)
  ipipt3 = i + ipt(3)
  ic = icp(ipipt3)
  iaqc = (ipipt3-1)*ipt(1)
  Oaq = Oaq + krt(d(ic) * x * cmw(ic) * (
+ den(ia1) * sat(ia1s) * xmf(iaqc+i1) +
+ den(ia2) * sat(ia2s) * xmf(iaqc+i2) +
+ den(ia3) * sat(ia3s) * xmf(iaqc+i3) )
EndDo
Wgas = x * ( ! mass of gas phase
+ den(ipt1x3+ig1) * sat(ig1s) +
+ den(ipt1x3+ig2) * sat(ig2s) +
+ den(ipt1x3+ig3) * sat(ig3s) )
Ogas = zero ! mass of organic in gas phase
Do i = 1, ipt(13)
  igasc = (i-1)*ipt(1)
  ic = icp(i)
  Ogas = Ogas + x * cmw(ic) * (
+ den(ig1) * sat(ig1s) * xmf(igasc+i1) +
+ den(ig2) * sat(ig2s) * xmf(igasc+i2) +
+ den(ig3) * sat(ig3s) * xmf(igasc+i3) )
EndDo
Wslid = bsden(matel(jel))*rbar(jel)*area(jel) ! mass of solid
Oslid = zero ! mass of organic in solid phase
Do i = 1, ipt(16)
  ipipt5 = i + ipt(59)
  ic = icp(ipipt5)
  isc = (ipipt5-1)*ipt(1)
  If (xbok.gt.zero) then
    If (i.eq.1) then
      solden = (rone-xden)*bsden(matel(i))
    Else If (i.eq.2) then
      isum = ipt(3)+ipt(4)+ipt(5)
      ic = icp(isum+i-1)
      solden = xden*bsden(matel(i))
    End If
    Oslid = Oslid + solden *
+ ( xmf(isc+i1) + xmf(isc+i2) + xmf(isc+i3) )
+ * x1
  Else
    Oslid = Oslid + bsden(matel(jel)) *
+ ( xmf(isc+i1) + xmf(isc+i2) + xmf(isc+i3) )
+ * x1
  EndIf
EndDo
tp(jel) = 1.d6*(Onapl+Oaq+Ogas+Oslid)/(Wnapl+Waq+Wgas+Wslid)
tp(jel) = 1.d6*(Onapl+Oaq+Ogas+Oslid)/(Wslid)
EndDo
If (lprnt(29)) call fprnt (21, tp(1), ipt(0),
+ '@ Element average total organic soil concentration (ppm)')
EndIf
C
C— Print nodal gas phase saturation.
C
if (lprnt(20)) then
  if (ipt(1).eq.ipt(2)) then
    call fprnt (21, sat(1), ipt(1), '@ Gas phase saturation')
  else
    call fprnt2 (21, sat(1), ipt(2), '@ Gas phase saturation'
+ , matpt, nodept, ipt(1))
  end if
end if
C
C— Print nodal aqueous phase saturation.
C
if (lprnt(21)) then
  if (ipt(1).eq.ipt(2)) then
    call fprnt (21, sat(ipt(1)+1), ipt(1),
+ '@ Aqueous phase saturation')
  else
    call fprnt2 (21, sat(ipt(2)+1), ipt(2),
+ '@ Aqueous phase saturation', matpt, nodept, ipt(1))
  end if

```

```

endif
C
C— Print nodal NAPL phase saturation.
C
if (lprnt(22)) then
  if (ipt(1).eq.ipt(2)) then
    call fprnt (21, sat(ipt(40)+1), ipt(1),
+ '@ NAPL phase saturation')
  else
    call fprnt2 (21, sat(ipt(49)+1), ipt(2),
+ '@ NAPL phase saturation', matpt, nodept, ipt(1))
  end if
end if
C
C— Print gas phase Darcy velocities.
C
if (lprnt(23)) then
C
C— Print nodal Darcy velocities.
C
if (lctrl(18)) then
  call fprnt (21, q(1), ipt(1),
+ '@ Nodal gas phase x Darcy velocity (m/sec)')
  call fprnt (21, q(ipt(1)+1), ipt(1),
+ '@ Nodal gas phase z Darcy velocity (m/sec)')
C
C— Otherwise print element Darcy velocities.
C
else
  call fprnt (21, q(1), ipt(0),
+ '@ Element gas phase x Darcy velocity (m/sec)')
  call fprnt (21, q(ipt(0)+1), ipt(0),
+ '@ Element gas phase z Darcy velocity (m/sec)')
end if
end if
C
C— Print aqueous phase Darcy velocities.
C
if (lprnt(24)) then
C
C— Print nodal Darcy velocities.
C
if (lctrl(18)) then
  call fprnt (21, q(ipt(40)+1), ipt(1),
+ '@ Nodal aqueous phase x Darcy velocity (m/sec)')
  call fprnt (21, q(ipt(41)+1), ipt(1),
+ '@ Nodal aqueous phase z Darcy velocity (m/sec)')
else
  call fprnt (21, q(ipt(67)+1), ipt(0),
+ '@ Element aqueous phase x Darcy velocity (m/sec)')
  call fprnt (21, q(ipt(68)+1), ipt(0),
+ '@ Element aqueous phase z Darcy velocity (m/sec)')
end if
end if
C
C— Write contour information.
C
if (lctrl(23)) then
  ipcon = 0
  if (lcon(2)) then
    ipcon = ipcon + 1
    contr(ipcon) = ' gas p '
  end if
  if (lcon(3)) then
    ipcon = ipcon + 1
    contr(ipcon) = ' aq p '
  end if
  if (lcon(4)) then
    ipcon = ipcon + 1
    contr(ipcon) = ' cap p '
  end if
  if (lcon(5)) then
    ipcon = ipcon + 1
    contr(ipcon) = ' gas den '
  end if
  if (lcon(6)) then
    ipcon = ipcon + 1

```

```

contr(ipcon) = ' aq den '
end if
if(lcon(7)) then
  ipcon = ipcon + 1
  contr(ipcon) = ' napl den '
end if
if(ipcon.gt.0) then
  write(24,*) ' elapsed time=',t(9)
  write(24,*) ' x z ',(contr(i),i=1,ipcon)
  if(lcon(1)) then
    write(24,*) ' x,z in m; p in pascals; density in mole/m3'
    iden = 0
  else
    write(24,*) ' x,z in m; p in pascals; density in gm/L'
    iden = ipt(41)
  end if
  do 70 i = 1, ipt(1)
    ipcon = 0
    if(lcon(2)) then
      ipcon = ipcon + 1
      rcont(ipcon) = rone + p(i)/patm
    end if
    if(lcon(3)) then
      ipcon = ipcon + 1
      rcont(ipcon) = p(i+ipt(1))
    end if
    if(lcon(4)) then
      ipcon = ipcon + 1
      rcont(ipcon) = p(i+ipt(40))
    end if
    if(lcon(5)) then
      ipcon = ipcon + 1
      rcont(ipcon) = den(i+iden)
    end if
    if(lcon(6)) then
      ipcon = ipcon + 1
      rcont(ipcon) = den(i+ipt(1)+iden)
    end if
    if(lcon(7)) then
      ipcon = ipcon + 1
      rcont(ipcon) = den(i+ipt(40)+iden)
    end if
70 write(24,550) xnode(i),znode(i),(rcont(ii),ii=1,ipcon)
  end if
  if(lcon(8).or.lcon(9).or.lcon(10).or.lcon(11).or.lcon(12)) then
    if(ipt(74).le.nout) then
      ifirst = 1
      ilast = ipt(74)
      ipass = 1
    else
      ifirst = 1
      ilast = nout
      ipass = ipt(74)/nout
      if(mod(ipt(74),nout).gt.zero) ipass = ipass + 1
    end if
    do 80 isweep = 1, ipass
      do 85 i = ifirst,ilast
        if(i.le.ipt(69)) then
          contr(i-ifirst+1) = ' gas '
        else if (i.le.ipt(69)+ipt(70).and.i.gt.ipt(69)) then
          contr(i-ifirst+1) = ' aqueous '
        else if (i.le.ipt(69)+ipt(70)+ipt(71).and.
+ i.gt.ipt(69)+ipt(70)) then
          contr(i-ifirst+1) = ' NAPL '
        else if (i.le.ipt(69)+ipt(70)+ipt(71)+ipt(72).and.
+ i.gt.ipt(69)+ipt(70)+ipt(71)) then
          contr(i-ifirst+1) = ' solid '
        else
          contr(i-ifirst+1) = ' biophase '
        end if
85 continue
      write(24,*) ' elapsed time=',t(9)
      write(24,*) '
      (contr(i-ifirst+1),i=ifirst,ilast)
+ write(24,*) ' x z '
+ (cname(icp(ipt(61)+i)),i=ifirst,ilast)
      if(lcon(1)) then
        write(24,*) ' x,z in m; composition in mole fractions'

```

```

+ ' solid phase loading in g/g; biomass in g/L'
      else
        write(24,*) ' x,z in m; composition in gm/L'
+ ' solid phase loading in g/g; biomass in g/L'
      end if
      do 90 i = 1, ipt(1)
        do 95 ii = ifirst,ilast
          if(lcon(1)) then
            xm(ii-ifirst+1) = rone
          else
C — Convert gas phase mole fractions into concentrations (g/L) if
C — lcon(1) = false.
C
            if(ii.le.ipt(69)) then
              xm(ii-ifirst+1) = den(i)*cmw(icp(ipt(61)+ii))
C
C — Convert aqueous and biophase mole fractions into concentrations
C — (g/L) if lcon(1) = false. Biomass is always gm/L.
C
            else if ((ii.le.ipt(69)+ipt(70).and.ii.gt.
+ ipt(69)).or.(ii.le.ipt(74).and.ii
+ .gt.ipt(74)-ipt(73))) then
              if(icp(ipt(61)+ii).eq.ipt(65)+1) then
                xm(ii-ifirst+1) = rone
              else
                xm(ii-ifirst+1) = den(ipt(1)+i)
+ * cmw(icp(ipt(61)+ii))
              end if
C
C — Convert NAPL mole fractions to concentrations (g/L).
C
            else if (ii.le.ipt(69)+ipt(70)+ipt(71).and.
+ ii.gt.ipt(69)+ipt(70)) then
              xm(ii-ifirst+1) = den(ipt(40)+i)
+ * cmw(icp(ipt(61)+ii))
C
C — Solid phase loadings are always g/g.
C
            else
              xm(ii-ifirst+1) = rone
            end if
          end if
          continue
95 write(24,550) xnode(i),znode(i),(xmf(i+icp(ipt(61)
+ +ipt(74)+ii))*xm(ii-ifirst+1),ii=ifirst,ilast)
        if(ipass.gt.1) then
          ifirst = ifirst + nout
          ilast = min0(ipt(74),ilast + nout)
        end if
80 continue
      end if
      if(lcon(13).or.lcon(14).or.lcon(15)) then
        write(24,*) ' elapsed time=',t(9)
        ipcon = 0
        if(lcon(13)) then
          ipcon = ipcon + 1
          contr(ipcon) = ' gas sat '
        end if
        if(lcon(14)) then
          ipcon = ipcon + 1
          contr(ipcon) = ' aq sat '
        end if
        if(lcon(15)) then
          ipcon = ipcon + 1
          contr(ipcon) = ' napl sat '
        end if
        write(24,*) ' x z ',(contr(i),i=1,ipcon)
        if(ipt(1).eq.ipt(2)) then
          write(24,*) ' x,z in m'
        else
          write(24,*) ' x,z in m; sat stacked'
        end if
        do 100 i = 1, ipt(1)
          do 100 istk = 0, nnodept(i+1)-nodept(i)-1
            ipcon = 0
            if(lcon(13)) then
              ipcon = ipcon + 1

```

```

    rcont(ipcon) = sat(nodept(i)+istk)
end if
if(lcon(14)) then
    ipcon = ipcon + 1
    rcont(ipcon) = sat(nodept(i)+istk+ipt(2))
end if
if(lcon(15)) then
    ipcon = ipcon + 1
    rcont(ipcon) = sat(nodept(i)+istk+ipt(49))
end if
100 write(24,550) xnode(i),znode(i),(rcont(ii),ii=1,ipcon)
end if
if(lcon(16).or.lcon(17)) then
    ipcon = 0
    if(lcon(16)) then
        ipcon = ipcon + 1
        contr(ipcon) = ' gas qx '
        ipcon = ipcon + 1
        contr(ipcon) = ' gas qz '
    end if
    if(lcon(17)) then
        ipcon = ipcon + 1
        contr(ipcon) = ' aq qx '
        ipcon = ipcon + 1
        contr(ipcon) = ' aq qz '
    end if
    write(24,*) ' elapsed time=',t(9)
    write(24,*) ' x z ',(contr(i),i=1,ipcon)
    if(lctrl(18)) then
        write(24,*) ' x,z in m; nodal q in m/s'
        do 110 i = 1, ipt(1)
            ipcon = 0
            if(lcon(16)) then
                ipcon = ipcon + 1
                rcont(ipcon) = q(i)
                ipcon = ipcon + 1
                rcont(ipcon) = q(i+ipt(1))
            end if
            if(lcon(17)) then
                ipcon = ipcon + 1
                rcont(ipcon) = q(i+ipt(40))
                ipcon = ipcon + 1
                rcont(ipcon) = q(i+ipt(41))
            end if
110 write(24,550) xnode(i),znode(i),(rcont(ii),ii=1,ipcon)
        else
            write(24,*)
            + ' x,z in m (element centroids); element q in m/s'
            do 115 i = 1, ipt(0)
                ipcon = 0
                if(lcon(16)) then
                    ipcon = ipcon + 1
                    rcont(ipcon) = q(i)
                    ipcon = ipcon + 1
                    rcont(ipcon) = q(i+ipt(0))
                end if
                if(lcon(17)) then
                    ipcon = ipcon + 1
                    rcont(ipcon) = q(i+ipt(67))
                    ipcon = ipcon + 1
                    rcont(ipcon) = q(i+ipt(68))
                end if
                iel3 = i*3
                iel1 = iel3-2
                iel2 = iel3-1
                i1 = node1(iel1)
                i2 = node1(iel2)
                i3 = node1(iel3)
                xel = third*(xnode(i1)+xnode(i2)+xnode(i3))
                zel = third*(znode(i1)+znode(i2)+znode(i3))
115 write(24,550) xel,zel,(rcont(ii),ii=1,ipcon)
            end if
        end if
    if(lcon(18)) then
        write(24,*) ' elapsed time=',t(9)
        write(24,*) ' x z Total organic soil conc'
        write(24,*) ' x,z in m (element centroids); conc (ppm)'
        Do i = 1, ipt(0)

```

```

        iel3 = i*3
        iel1 = iel3-2
        iel2 = iel3-1
        i1 = node1(iel1)
        i2 = node1(iel2)
        i3 = node1(iel3)
        xel = third*(xnode(i1)+xnode(i2)+xnode(i3))
        zel = third*(znode(i1)+znode(i2)+znode(i3))
        write(24,550) xel,zel,tp(i)
    EndDo
Enddf
end if
C
C— Generate a restart file if desired.
C
if(lctrl(5)) then
    rewind(27)
    write(27,556) '# ipt(i),i=0,7'
    write(27,549) (ipt(i),i=0,7)
    write(27,556) '# t(8),t(2),t(9),its'
    write(27,551) t(8),t(2),t(9),its
    write(27,556) '# str1(i),str0(i),cmf(i),csink(i),cwsink(i)'
    write(27,556) '# crsink(i),csflux(i),cphe(x(i),i=1, ipt(65)+5'
    + ,crsink(i),csflux(i),cphe(x(i),i=1, ipt(65)+5)
    write(27,556) '# cmass1(i),cmass0(i),i=1, ipt(65)*5'
    write(27,555) (cmass1(i),cmass0(i),i=1, ipt(65)*5)
    write(27,556) '# tmass1,tmass0'
    write(27,555) tmass1,tmass0
    write(27,556) '# p'
    iend1 = ipt(41)
    write(27,549) iend1
    write(27,555) (p(i),i=1,iend1)
    write(27,556) '# sat'
    iend2 = ipt(50)
    write(27,549) iend2
    write(27,555) (sat(i),i=1,iend2)
    write(27,556) '# xmf'
    iend3 = ipt(61)*ipt(1)
    write(27,549) iend3
    write(27,555) (xmf(i),i=1,iend3)
    write(27,556) '# pex'
    iend4 = ipt(52)
    write(27,549) iend4
    write(27,555) (pex(ipt(52)+i),i=1,iend4)
    close(27)
    open(27,file=outfile(7),status='unknown')
end if
500 format(7/2('**'))' Solution at: /
+ ' time step number =',i5 /
+ ' time in seconds =',e12.5 /
+ ' time in minutes =',e12.5 /
+ ' time in hours =',e12.5 /
+ ' time in days =',e12.5 /72('**'))
549 format(8i10)
550 format(8e15.7)
551 format(3e15.8,i10)
555 format(5e15.8)
556 format(a,i10)
C
return
end
C
subroutine fprnt (nu,var,ilen,label)
character label*(*)
real*8 var(ilen)
write (nu,100) label
write (nu,101) (i,var(i),i=1,ilen)
100 format (/a)
101 format (4(i5,',',e11.5,1x))
return
end
C
subroutine fprnt2 (nu,var,ilen2,label,matpt,nodept,ilen1)
character label*(*)
real*8 var(ilen2)
integer matpt(ilen2),nodept(ilen2+1),idum(5000)
write (nu,100) label

```



```

C   lctrl(8) - logical variable controlling type of FEM
C       solution for transport
C       lctrl(8) = .true. - use mass lumping
C       lctrl(8) = .false. - full FEM solution
C   lctrl(19) - logical variable controlling type of sorption
C       lctrl(19) = .true. - two compartment sorption
C       lctrl(19) = .false. - single compartment
C       sorption
C
C-----
subroutine SOLID(iconv)
include 'dimen.inc'
character*10 cname(ncmp)
C
C--- Declare and define common block variables.
C
common /cb1/ matel(nelmx),nodel(nel3),nodept(nnm),nelpt(nel3),
+   matpt(nn6)
common /cb1e/ aby12(nelmx),aby30(nelmx)
common /cb6b/ por(nelmx),srw(nnstk)
common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
+   chen(ncmp),casol(ncmp),cmdif(ncmp2)
common /cb9/ xmf(nmf)
common /cb9b/ xmf(nmf)
common /cb40/ a(icnl),rhs(ksolve),w(icnl)
common /cb41/ irm(icnl),icn(icnl),iw(icnl,8),ikeep(icnl,5)
common /cb41b/ nbw(0:2),ia
common /cb62b/ rhsex(nmfs)
common /cb64b/ bsden(nmbk)
common /cb91/ cname
C
C--- Set pointers and sequentially solve for the solid phase
C--- mass loadings.
C
ipt1 = ipt(1)
ipt2 = ipt(2)
do 200 ics = 1, ipt(6)
  iptc = ipt(59) + ics
  icpt = (iptc-1)*ipt1
  icpts = (iptc-1)*ipt2
  ic = icp(iptc)
C
C--- If two compartment sorption is considered, do not advance the
C--- component identifier.
C
if((ics.eq.2).and.lctrl(19)) ic = icp(iptc-1)
C
C--- Zero the finite element matrices.
C
do 210 i = 1, ipt2
  rhs(i) = zer0
  nrow = (i-1)*nbw(1)
  do 210 j = 1, nbw(1)
210   a(nrow+j) = zer0
C
C--- Compute the local finite element matrices.
C
do 220 i = 1, ipt(0)
C
C--- If two compartment sorption is considered, reset the solid phase
C--- mass density for the slow and fast compartments. Ipt(6)=1 for the
C--- slow compartment and ipt(6)=2 for the fast compartment.
C
if((ics.eq.1).and.lctrl(19)) then
  solden = (rone-xden)*bsden(matel(i))
else if((ics.eq.2).and.lctrl(19)) then
  solden = xden*bsden(matel(i))
else
  solden = bsden(matel(i))
end if
C
C--- Set the pointers to the local nodes I1, I2, and I3. The
C--- postscript s is for stacked local nodes, p is for phase, and
C--- c is for component. Compute constants.
C
iel3 = i*3
iel1 = iel3-2

```

```

iel2 = iel3-1
i1 = nodel(iel1)
i1s = nodept(i1)+nelpt(iel1)
i1cs = icpts + i1s
i2 = nodel(iel2)
i2s = nodept(i2)+nelpt(iel2)
i2cs = icpts + i2s
i3 = nodel(iel3)
i3s = nodept(i3)+nelpt(iel3)
i3cs = icpts + i3s
term1 = aby30(i)/t(8)
C
C--- Compute the lumped mass matrix.
C
if(lctrl(8)) then
  amass = 10.0d0 * term1
  else
    aon = term1*5.0d0
    aoff = term1*2.50d0
  end if
C
C--- Compute the exchange terms. First consider phase mass transfer.
C--- This term is updated after a complete pass through all the
C--- components and phases.
C
term2 = aby12(i) * por(i) * cmw(ic) / solden
rhs1 = rhsex(i1cs)
rhs2 = rhsex(i2cs)
rhs3 = rhsex(i3cs)
f1 = term2 * ( 2.0d0*rhs1 +   rhs2 +   rhs3 )
f2 = term2 * (   rhs1 + 2.0d0*rhs2 +   rhs3 )
f3 = term2 * (   rhs1 +   rhs2 + 2.0d0*rhs3 )
C
C--- Assemble lumped global matrix Note that the left hand side matrix
C--- is diagonal due to mass lumping and the solution is explicit.
C
if(lctrl(8)) then
  a(i1) = a(i1) + amass
  a(i2) = a(i2) + amass
  a(i3) = a(i3) + amass
  else
C
C--- Assemble global matrix in banded form.
C
irow1 = (i1-1)*nbw(1)
irow2 = (i2-1)*nbw(1)
irow3 = (i3-1)*nbw(1)
icol11 = 1 + nbw(0)
icol12 = icol11 + (i2 - i1)
icol13 = icol11 + (i3 - i1)
icol22 = icol11
icol21 = icol22 + (i1 - i2)
icol23 = icol22 + (i3 - i2)
icol33 = icol11
icol31 = icol33 + (i1 - i3)
icol32 = icol33 + (i2 - i3)
a(irow1 + icol11) = a(irow1 + icol11) + aon
a(irow1 + icol12) = a(irow1 + icol12) + aoff
a(irow1 + icol13) = a(irow1 + icol13) + aoff
a(irow2 + icol21) = a(irow2 + icol21) + aoff
a(irow2 + icol22) = a(irow2 + icol22) + aon
a(irow2 + icol23) = a(irow2 + icol23) + aoff
a(irow3 + icol31) = a(irow3 + icol31) + aoff
a(irow3 + icol32) = a(irow3 + icol32) + aoff
a(irow3 + icol33) = a(irow3 + icol33) + aon
end if
C
C--- Assemble right hand side.
C
rhs(i1) = rhs(i1) + f1
rhs(i2) = rhs(i2) + f2
rhs(i3) = rhs(i3) + f3
220 continue
C
C--- Solve for solid phase mass fractions. Determine the max norm of
C--- the updated solution.
C
if(lctrl(8)) then

```

```

dxmf = zero
xmfmax = xround
do 250 i = 1, ipt1
  xmfold=xmf(icpt+i)
  if(a(i).ne.zero) then
    xmf(icpt+i) = xmf(icpt+i) + rhs(i)/a(i)
    xmfmax = dmax1(xmfmax,dabs(xmf(icpt+i)))
    dxmf = dmax1(dxmf,dabs(xmf(icpt+i)-xmfold))
  end if
250 continue
else
C
C— Collapse full matrix into sparse form used by Harwell. Also
C— scale array by dividing rows through by the diagonal value.
C
  ia = 0
  do 260 irow = 1, ipt1
    nrow = (irow-1)*nbw(1)
    aii = rone / a(nrow+1+nbw(0))
    rhs(irow) = rhs(irow) * aii
    do 260 icol = 1, nbw(1)
      if (a(nrow+icol) .ne. zero) then
        ia = ia + 1
        a(ia) = a(nrow+icol) * aii
        irn(ia) = irow
        icn(ia) = icol+irow-nbw(0)-1
      endif
260 continue
C

```

```

C— Solve the linear system using Harwell routines.
C
  call ma28ad(ipt(1),ia,a,icn1,irn,irnl,icn,u,ikeep,iw,w,iflag)
  if (iflag .lt. 0) then
    write (ipt(28),*) 'iflag return from harwell is ', iflag
    write (ipt(28),*) 'solid component is: ', cname(ic)
  end if
  call ma28cd (ipt(1),a,icn1,icn,ikeep,rhs,w,mtype)
C
C— Update the solution and determine the max norm of the updated
C— solution.
C
  dxmf = zero
  xmfmax = xround
  do 270 i = 1+icpt, ipt1+icpt
    xmfold = xmf(i)
    xmf(i) = xmf(i) + rhs(i-icpt)
    xmfmax = dmax1(xmfmax,dabs(xmf(i)))
270   dxmf = dmax1(dxmf,dabs(xmf(i)-xmfold))
    end if
C
C— Check convergence with the relative change in state variable.
C
  if(dxmf/xmfmax.gt.t(16)) iconv = iconv + 1
200 continue
return
end

```

Subroutine - tlhs.f

```

C
C
C TLHS.f - Subroutine which forms the left hand side of the
C finite element equations for the two dimensional
C component mass balance equations. Linear triangles
C are used for the basis and weighting functions.
C
C Arguments: iphase - integer scalar denoting the phase
C gas phase: iphase = 1
C aqueous phase: iphase = 2
C icomp - integer scalar denoting the component
C gas phase: icomp = icp(1)-icp(ipt(3))
C aqueous phase: icomp = icp(ipt(3)+1) to
C icp(ipt(4))
C ixc - integer pointer for principal component
C number
C nxc - integer pointer for principal component
C location in xmf
C
C Required Control Flags:
C
C ipt(27) - integer variable indicating type of domain
C ipt(27) = 0 - xz domain
C ipt(27) = 1 - rz domain
C lctrl(4) - logical variable controlling printing of
C element dimensionless numbers
C lctrl(4) = .true. - print element
C dimensionless numbers
C lctrl(4) = .false. - skip element
C dimensionless numbers
C lctrl(8) - logical variable controlling type of FEM
C solution for transport
C lctrl(8) = .true. - use mass lumping
C lctrl(8) = .false. - full FEM solution
C lctrl(18) - logical variable denoting type of q calculation
C lctrl(18) = .true. - nodal FEM q
C lctrl(18) = .false. - element average q
C lctrl(21) - logical variable controlling calculation of
C the hydrodynamic dispersion tensor
C lctrl(21) = .true. - calculate tensor
C lctrl(21) = .false. - read tensor as input
C

```

```

C
C Control Flags computed internally in routine:
C
C t(23) - maximum gas phase cell Peclet number
C t(24) - maximum aqueous phase cell Peclet number
C t(25) - maximum gas phase cell Courant number
C t(26) - maximum aqueous phase cell Courant number
C
C Note: Component balance equations are not needed for
C nitrogen in the gas phase and water in the
C aqueous phase
C
C
C subroutine TLHS(iphase,icomp,ixc,nxc)
C include 'dimen.inc'
C
C— Declare and define common block variables.
C
  common /cb1/ matel(nelmx),nodel(nel3),nodept(nnmx),nelpt(nel3),
+   matpt(nn6)
  common /cb1c/ xnode(nnmx),znode(nnmx),rbar(nelmx),area(nelmx)
  common /cb1d/ gama(nel3),beta(nel3)
  common /cb1e/ aby12(nelmx),aby30(nelmx)
  common /cb2c/ q(nel4)
  common /cb3/ sat(nnst3)
  common /cb6b/ por(nelmx),srw(nnstk)
  common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
+   chen(ncmp),casol(ncmp),cmdif(ncmp2)
  common /cb9/ xmf(nmf)
  common /cb9b/ xmf(nmf)
  common /cb10/ den(nn6)
  common /cb11/ pex(nns10),rxnp(nn2)
  common /cb30/ ibc(nnmx)
  common /cb32/ bcf(nn2)
  common /cb40/ a(icn1),rhs(nsolve),w(icn1)
  common /cb41/ irn(icn1),icn(icn1),iw(icn1,8),ikeep(icn1,5)
  common /cb41b/ nbw(0:2),ia
  common /cb62/ rxn(nmf),cex(nmfs)
  common /cb62b/ rhsex(nmfs)
  common /cb64/ bok(nbcmp),bom(nbcmp),krtcd(ncmp)
  common /cb70/ d(nmd),tort(nelmx),bdist(nmblk),bdisl(nmblk)

```

```

common /cb84/ ibcxmf(nmbc),bcxmf(nmbc),dfxmf(nmbc)
common /cb85/ flux(ncmpp5),sflux(ncmpp5),first(ncmp)
C
C— Compute terms which are not component dependent. These
C— terms are only calculated once per iteration for each phase.
C— Note that for mobile phases, saturation is stacked (i.e.
C— discontinuous) while phase molar density not stacked (i.e.
C— continuous).
C
C— Zero the global matrix and right hand side vector.
C
do 110 i = 1, ipt(1)
  rhs(i) = zero
  nrow = (i-1)*nbw(1)
  do 110 j = 1, nbw(1)
110   a(nrow+j) = zero
C
C— Set pointers. iptc points to the correct location in icp.
C— icpt and icpts point to the correct location in xmf,
C— and stacked xmf storage respectively. ippt and ippts point to
C— the correct location in nonstacked and stacked phase storage.
C— ic is the identity of the icomp component in the iphase phase.
C— iptco points to the organic phase components.
C
ipm1 = iphase-1
iptc = ipm1*ipt(3)+icomp
icpt = (iptc-1)*ipt(1)
icpts = (iptc-1)*ipt(2)
ippt = ipm1*ipt(1)
ippts = ipm1*ipt(2)
ic = icp(iptc)
ipt1 = ipt(1)
ipt0 = ipt(0)
C
C— Compute the local finite element matrices.
C
do 120 i=1, ipt(0)
C
C— Set the pointers to the local nodes i1, i2, and i3. The
C— postscript s is for stacked local nodes, p is for phase, and
C— c is for component. Compute constants.
C
iel3 = i*3
iel1 = iel3-2
iel2 = iel3-1
i1 = node1(iel1)
i1s = nodept(i1)+nelpt(iel1)
i1c = icpt + i1
i1cs = icpts + i1s
i1p = ippt + i1
i1ps = ippts + i1s
i2 = node1(iel2)
i2s = nodept(i2)+nelpt(iel2)
i2c = icpt + i2
i2cs = icpts + i2s
i2p = ippt + i2
i2ps = ippts + i2s
i3 = node1(iel3)
i3s = nodept(i3)+nelpt(iel3)
i3c = icpt + i3
i3cs = icpts + i3s
i3p = ippt + i3
i3ps = ippts + i3s
sd1 = sat(i1ps)*den(i1p)
sd2 = sat(i2ps)*den(i2p)
sd3 = sat(i3ps)*den(i3p)
d1 = den(i1p)
d2 = den(i2p)
d3 = den(i3p)
term30 = por(i)*aby30(i)
term12 = por(i)*aby12(i)
term = term30/t(8)
C
C— If an element has gas saturations below sgtest skip
C— calculation of the local element matrix.
C
if((iphase.eq.1).and.((sat(i1s).lt.sgtest).or.
+ (sat(i2s).lt.sgtest).or.(sat(i3s).lt.sgtest)))) goto 120

```

```

C
C— Consider equilibrium sorption in the aqueous phase.
C
if(iphase.eq.2) then
  term = term * krt(d(ic))
end if
C
C— Compute the mass matrix. These terms are constant for a given
C— phase and component unless the phase density or saturation
C— is updated.
C
a11 = term*(3.0d0*sd1 + sd2 + sd3 )
a12 = term*( sd1 + sd2 + sd3/2.0d0)
a13 = term*( sd1 + sd2/2.0d0+ sd3 )
a21 = a12
a22 = term*( sd1 + 3.0d0*sd2 + sd3 )
a23 = term*( sd1/2.0d0+ sd2 + sd3 )
a31 = a13
a32 = a23
a33 = term*( sd1 + sd2 + 3.0d0*sd3 )
C
C— Lump 'mass' matrix if lctrl(8) = true.
C
if(lctrl(8)) then
  a11 = a11 + a12 + a13
  a12 = zero
  a13 = zero
  a22 = a21 + a22 + a23
  a21 = zero
  a23 = zero
  a33 = a31 + a32 + a33
  a31 = zero
  a32 = zero
endif
C
C— Assign the Darcy velocities.
C
if(lctrl(18)) then
C
C— Compute the stiffness matrix using nodal Darcy velocities.
C
iptq = ipm1*ipt(40)
dqx1 = q(iptq+i1)*den(i1p)
dqx2 = q(iptq+i2)*den(i2p)
dqx3 = q(iptq+i3)*den(i3p)
dqz1 = q(iptq+ipt1+i1)*den(i1p)
dqz2 = q(iptq+ipt1+i2)*den(i2p)
dqz3 = q(iptq+ipt1+i3)*den(i3p)
C
termq = rbar(i) / 24.0d0
sum1x = termq * (2.0d0*dqx1 + dqx2 + dqx3)
sum2x = termq * ( dqx1 + 2.0d0*dqx2 + dqx3)
sum3x = termq * ( dqx1 + dqx2 + 2.0d0*dqx3)
sum1z = termq * (2.0d0*dqz1 + dqz2 + dqz3)
sum2z = termq * ( dqz1 + 2.0d0*dqz2 + dqz3)
sum3z = termq * ( dqz1 + dqz2 + 2.0d0*dqz3)
C
b11 = sum1x * beta(iel1) + sum1z * gama(iel1)
b12 = sum1x * beta(iel2) + sum1z * gama(iel2)
b13 = sum1x * beta(iel3) + sum1z * gama(iel3)
b21 = sum2x * beta(iel1) + sum2z * gama(iel1)
b22 = sum2x * beta(iel2) + sum2z * gama(iel2)
b23 = sum2x * beta(iel3) + sum2z * gama(iel3)
b31 = sum3x * beta(iel1) + sum3z * gama(iel1)
b32 = sum3x * beta(iel2) + sum3z * gama(iel2)
b33 = sum3x * beta(iel3) + sum3z * gama(iel3)
else
C
C— Compute advective terms using element Darcy velocities.
C
termq = rbar(i) / 24.0d0
sum1 = termq * (2.0d0*den(i1p)+den(i2p)+den(i3p))
sum2 = termq * (den(i1p)+2.0d0*den(i2p)+den(i3p))
sum3 = termq * (den(i1p)+den(i2p)+2.0d0*den(i3p))
C
C— Assign the Darcy velocities. Use q here since the saturations and
C— porosities are dropped from the preceding terms.
C

```

```

iptq = i + ipm1 * ipt(67)
qx = q(iptq)
qz = q(iptq+ipt0)
C
C— Compute the stiffness matrix. First compute the advective terms.
C— These terms are constant unless the phase density or Darcy
C— velocity is updated.
C
b1g1 = beta(iel1)*qx + gama(iel1)*qz
b2g2 = beta(iel2)*qx + gama(iel2)*qz
b3g3 = beta(iel3)*qx + gama(iel3)*qz
b11 = sum1 * b1g1
b12 = sum1 * b2g2
b13 = sum1 * b3g3
b21 = sum2 * b1g1
b22 = sum2 * b2g2
b23 = sum2 * b3g3
b31 = sum3 * b1g1
b32 = sum3 * b2g2
b33 = sum3 * b3g3
end if
C
C— Assign correct values to dispersion coefficients. Element wise
C— constant dispersion coefficients are used in this version.
C
if (lctrl(21)) then
dm = third*(sat(i1ps)+sat(i2ps)+sat(i3ps))*por(i)
+ *cmdiff(2*(ic-1)+iphase)*tor(i)
id = (iphase-1)*ipt(68)
d11 = d(id+3*i-2) + dm
d12 = d(id+3*i-1)
d21 = d12
d22 = d(id+3*i) + dm
else
idpt = 8*(ic-1) + (iphase-1)*4
d11 = d(idpt+1)
d12 = d(idpt+2)
d21 = d(idpt+3)
d22 = d(idpt+4)
end if
C
C— Compute the maximum element Peclet and Courant number if desired.
C
if (lctrl(4)) then
pelen = dsqrt(2.0d0*area(i))
pesat = third*(sat(i1ps)+sat(i2ps)+sat(i3ps))
pepore = por(i)
if (lctrl(18)) then
qx = third*(q(iptq+i1)+q(iptq+i2)+q(iptq+i3))
+ qz = third*(q(iptq+ipt1+i1)+q(iptq+ipt1+i2)
+ q(iptq+ipt1+i3))
end if
if (iphase.eq.1) then
stest1 = sgtest
stest2 = sgtest
stest3 = sgtest
else if (iphase.eq.2) then
stest1 = srwmin + srw(i1s)
stest2 = srwmin + srw(i2s)
stest3 = srwmin + srw(i3s)
end if
if ((sat(i1ps).ge.stest1.or.sat(i2ps).ge.stest2
+ .or.sat(i3ps).ge.stest3).and.(dsqrt((d11/pepore)**2
+ (d22/pepore)**2)*pepore*pesat).gt.zer0) then
t(22+iphase) =
+ dmax1(t(22+iphase),dsqrt(qx**2+qz**2)*pelen
+ /dsqrt((d11/pepore)**2+(d22/pepore)**2)
+ *pepore*pesat))
+ t(24+iphase) = dmax1(t(24+iphase),dsqrt(qx**2+qz**2)*t(8)
+ /pelen*pepore*pesat))
else
t(22+iphase) = dmax1(t(22+iphase),zer0)
t(24+iphase) = dmax1(t(24+iphase),zer0)
end if
end if
C
C— Next compute the dispersive terms.

```

```

sum = (d1 + d2 + d3)
sum = rbar(i) * sum / (12.0d0 * area(i))
d1bg1 = d11 * beta(iel1) + d21 * gama(iel1)
d2bg1 = d12 * beta(iel1) + d22 * gama(iel1)
d1bg2 = d11 * beta(iel2) + d21 * gama(iel2)
d2bg2 = d12 * beta(iel2) + d22 * gama(iel2)
d1bg3 = d11 * beta(iel3) + d21 * gama(iel3)
d2bg3 = d12 * beta(iel3) + d22 * gama(iel3)
b11 = b11 + sum * (beta(iel1) * d1bg1 + gama(iel1) * d2bg1)
b12 = b12 + sum * (beta(iel2) * d1bg1 + gama(iel2) * d2bg1)
b13 = b13 + sum * (beta(iel3) * d1bg1 + gama(iel3) * d2bg1)
b21 = b21 + sum * (beta(iel1) * d1bg2 + gama(iel1) * d2bg2)
b22 = b22 + sum * (beta(iel2) * d1bg2 + gama(iel2) * d2bg2)
b23 = b23 + sum * (beta(iel3) * d1bg2 + gama(iel3) * d2bg2)
b31 = b31 + sum * (beta(iel1) * d1bg3 + gama(iel1) * d2bg3)
b32 = b32 + sum * (beta(iel2) * d1bg3 + gama(iel2) * d2bg3)
b33 = b33 + sum * (beta(iel3) * d1bg3 + gama(iel3) * d2bg3)
C
C— Compute the exchange terms. First consider phase mass transfer.
C— This term is updated after a complete pass through all the
C— components and phases. Include the reaction term in the aqueous
C— phase.
C
if (iphase.eq.2.and.lctrl(3).and.lctrl(16)) then
pex1 = pex(i1ps) + (rxnp(i1) - rxn(i1c))/por(i)
pex2 = pex(i2ps) + (rxnp(i2) - rxn(i2c))/por(i)
pex3 = pex(i3ps) + (rxnp(i3) - rxn(i3c))/por(i)
else
pex1 = pex(i1ps)
pex2 = pex(i2ps)
pex3 = pex(i3ps)
end if
pterm1 = term30 * (pex1 + pex2 + pex3/2.0d0)
pterm2 = term30 * (pex1 + pex2/2.0d0 + pex3)
pterm3 = term30 * (pex1/2.0d0 + pex2 + pex3)
b11 = b11 + term30 * (3.0d0*pex1 + pex2 + pex3)
b12 = b12 + pterm1
b13 = b13 + pterm2
b21 = b21 + pterm1
b22 = b22 + term30 * (pex1 + 3.0d0*pex2 + pex3)
b23 = b23 + pterm3
b31 = b31 + pterm2
b32 = b32 + pterm3
b33 = b33 + term30 * (pex1 + pex2 + 3.0d0*pex3)
C
C— Now include the lumped component exchange terms.
C
cex1 = cex(i1cs)
cex2 = cex(i2cs)
cex3 = cex(i3cs)
cterm1 = term30 * (cex1 + cex2 + cex3/2.0d0)
cterm2 = term30 * (cex1 + cex2/2.0d0 + cex3)
cterm3 = term30 * (cex1/2.0d0 + cex2 + cex3)
b11 = b11 + term30 * (3.0d0*cex1 + cex2 + cex3)
b12 = b12 + cterm1
b13 = b13 + cterm2
b21 = b21 + cterm1
b22 = b22 + term30 * (cex1 + 3.0d0*cex2 + cex3)
b23 = b23 + cterm3
b31 = b31 + cterm2
b32 = b32 + cterm3
b33 = b33 + term30 * (cex1 + cex2 + 3.0d0*cex3)
C
C— Now compute the right hand side terms. These terms are constant
C— unless the phase saturation or contacting phase composition is
C— updated.
C
rhsex1 = rhsex(i1cs)
rhsex2 = rhsex(i2cs)
rhsex3 = rhsex(i3cs)
f1 = term12 * (2.0d0*rhsex1 + rhsex2 + rhsex3)
f2 = term12 * (rhsex1 + 2.0d0*rhsex2 + rhsex3)
f3 = term12 * (rhsex1 + rhsex2 + 2.0d0*rhsex3)
C
C— Assemble global matrix and right hand side vector in banded form.
C
irow1 = (i1-1)*nbw(1)
irow2 = (i2-1)*nbw(1)

```



```

irow3 = (i3-1)*nbw(1)
icol11 = 1 + nbw(0)
icol12 = icol11 + (i2 - i1)
icol13 = icol11 + (i3 - i1)
icol22 = icol11
icol21 = icol22 + (i1 - i2)
icol23 = icol22 + (i3 - i2)
icol33 = icol11
icol31 = icol33 + (i1 - i3)
icol32 = icol33 + (i2 - i3)
ab11 = a11 + t(10)*b11
ab12 = a12 + t(10)*b12
ab13 = a13 + t(10)*b13
ab21 = a21 + t(10)*b21
ab22 = a22 + t(10)*b22
ab23 = a23 + t(10)*b23
ab31 = a31 + t(10)*b31
ab32 = a32 + t(10)*b32
ab33 = a33 + t(10)*b33
a(irow1 + icol11) = a(irow1 + icol11) + ab11
a(irow1 + icol12) = a(irow1 + icol12) + ab12
a(irow1 + icol13) = a(irow1 + icol13) + ab13
a(irow2 + icol21) = a(irow2 + icol21) + ab21
a(irow2 + icol22) = a(irow2 + icol22) + ab22
a(irow2 + icol23) = a(irow2 + icol23) + ab23
a(irow3 + icol31) = a(irow3 + icol31) + ab31
a(irow3 + icol32) = a(irow3 + icol32) + ab32
a(irow3 + icol33) = a(irow3 + icol33) + ab33
rhs(i1) = rhs(i1) + f1 - b11 * xmf(i1c) - b12 * xmf(i2c)
+ - b13 * xmf(i3c)
rhs(i2) = rhs(i2) + f2 - b21 * xmf(i1c) - b22 * xmf(i2c)
+ - b23 * xmf(i3c)
rhs(i3) = rhs(i3) + f3 - b31 * xmf(i1c) - b32 * xmf(i2c)
+ - b33 * xmf(i3c)
C
C— Insert third type boundary condition here when the flow solution
C— is not included. First look at the three sides.
C
i1bc = i1p
i2bc = i2p
i3bc = i3p
if(ibcxmf(i1bc)+ibcxmf(i2bc)+ibcxmf(i3bc).eq.4) then
if(((ibcxmf(i1bc)+ibcxmf(i2bc).eq.6).or.
+ (ibcxmf(i1bc)+ibcxmf(i2bc).eq.4)
+ .and.(xnode(i1).eq.xnode(i2)
+ .or.znode(i1).eq.znode(i2))) then
if(ipt(27).eq.1) then
weight = xnode(i1) + xnode(i2)
weight1 = xnode(i1) / weight
weight2 = xnode(i2) / weight
else
weight1 = 0.50d0
weight2 = 0.50d0
end if
dx = xnode(i2) - xnode(i1)
dz = znode(i2) - znode(i1)
rlen = dsqrt(dx**2 + dz**2)
if(ipt(27).eq.1) then
rrad = pi*(xnode(i1)+xnode(i2))
rlen = rlen*rrad
end if
qnorm = zero
if(lctrl(18)) then
qx = q(iptq+i1)*weight1 + q(iptq+i2)*weight2
qz = q(iptq+ipt1+i1)*weight1 + q(iptq+ipt1+i2)*weight2
end if
if(dx.gt.zero) then
qnorm = (dx*qz)/rlen
else
qnorm = (-dx*qz)/rlen
end if
if(dz.gt.zero) then
qnorm = qnorm + (dz*qx)/rlen
else
qnorm = qnorm + (-dz*qx)/rlen
end if
if(dabs(qnorm).lt.(den(i1p)*dfxmf(i1c)*weight1 +
+ den(i2p)*dfxmf(i2c)*weight2)) then

```

```

dfx1 = den(i1p)*dfxmf(i1c)*rlen*weight1
dfx2 = den(i2p)*dfxmf(i2c)*rlen*weight2
else
dfx1 = zero
dfx2 = zero
end if
abc31 = den(i1p)*qnorm*rlen*weight1
abc32 = den(i2p)*qnorm*rlen*weight2
if(ibcxmf(i1bc)+ibcxmf(i2bc).eq.4) then
a(irow1 + icol11) = a(irow1 + icol11) + t(10) * dfx1
rhs(i1) = rhs(i1) + dfx1 * (bcxmf(i1c)-xmf(i1c))
a(irow2 + icol22) = a(irow2 + icol22) + t(10) * dfx2
rhs(i2) = rhs(i2) + dfx2 * (bcxmf(i2c)-xmf(i2c))

```

C
C— Save the flux terms for the mass balance. Also compute the
C— surface flux for output.

```

C
if(.not.lctrl(1)) then
f1xmf = xmf(i1c)
f2xmf = xmf(i2c)
fsave = abc31 * f1xmf + abc32 * f2xmf
+ + dfx1 * (bcxmf(i1c)-xmf(i1c))
+ + dfx2 * (bcxmf(i2c)-xmf(i2c))
flux(5+ixc) = flux(5+ixc) + fsave
if(znode(i1).eq.zero.and.znode(i2).eq.zero)
+ sflux(5+ixc) = sflux(5+ixc) + fsave

```

C
C— Compute the flux for the master component.

```

C
if(icomp.eq.1) then
fsave = abc31 * xmf(nxc+i1) + abc32 * xmf(nxc+i2)
flux(5+ixc) = flux(5+ixc) + fsave
if(znode(i1).eq.zero.and.znode(i2).eq.zero)
+ sflux(5+ixc) = sflux(5+ixc) + fsave
end if
end if
else if(.not.lctrl(1).and.
+ ibcxmf(i1bc)+ibcxmf(i2bc).eq.6) then
a(irow1 + icol11) = a(irow1 + icol11) + t(10) * abc31
rhs(i1) = rhs(i1) + abc31*(bcxmf(i1c)-xmf(i1c))
a(irow2 + icol22) = a(irow2 + icol22) + t(10) * abc32
rhs(i2) = rhs(i2) + abc32*(bcxmf(i2c)-xmf(i2c))

```

C
C— Save the flux terms for the mass balance. Also compute the
C— surface flux for output.

```

C
fsave = abc31 * bcxmf(i1c) + abc32 * bcxmf(i2c)
flux(5+ixc)
+ = flux(5+ixc) + fsave
if(znode(i1).eq.zero.and.znode(i2).eq.zero)
+ sflux(5+ixc) = sflux(5+ixc) + fsave

```

C
C— Compute the flux for the master component.

```

C
if(icomp.eq.1) then
fsave = abc31*bcxmf(nxc+i1) + abc32*bcxmf(nxc+i2)
flux(5+ixc) = flux(5+ixc) + fsave
if(znode(i1).eq.zero.and.znode(i2).eq.zero)
+ sflux(5+ixc) = sflux(5+ixc) + fsave
end if
end if
end if
if(((ibcxmf(i1bc)+ibcxmf(i3bc).eq.6).or.
+ (ibcxmf(i1bc)+ibcxmf(i3bc).eq.4)
+ .and.(xnode(i1).eq.xnode(i3)
+ .or.znode(i1).eq.znode(i3))) then
if(ipt(27).eq.1) then
weight = xnode(i1) + xnode(i3)
weight1 = xnode(i1) / weight
weight3 = xnode(i3) / weight
else
weight1 = 0.50d0
weight3 = 0.50d0
end if
dx = xnode(i3) - xnode(i1)
dz = znode(i3) - znode(i1)
rlen = dsqrt(dx**2 + dz**2)
if(ipt(27).eq.1) then

```

```

      rrad = pi*(xnode(i1)+xnode(i3))
      rlen = rlen*rrad
    end if
    qnorm = zero
    if(lctrl(18)) then
      qx=q(iptq+i1)*weight1+q(iptq+i3)*weight3
      qz=q(iptq+ipt1+i1)*weight1+q(iptq+ipt1+i3)*weight3
    end if
    if(dx.gt.zero) then
      qnorm=(dx*qz)/rlen
    else
      qnorm=(-dx*qz)/rlen
    end if
    if(dz.gt.zero) then
      qnorm = qnorm + (dz*qx)/rlen
    else
      qnorm = qnorm + (-dz*qx)/rlen
    end if
    if(dabs(qnorm).lt.(den(i1p)*dfxmf(i1c)*weight1+
+ den(i3p)*dfxmf(i3c)*weight3)) then
      dfx1=den(i1p)*dfxmf(i1c)*rlen*weight1
      dfx3=den(i3p)*dfxmf(i3c)*rlen*weight3
    else
      dfx1 = zero
      dfx3 = zero
    end if
    abc31 = den(i1p)*qnorm*rlen*weight1
    abc33 = den(i3p)*qnorm*rlen*weight3
    if(ibcxmf(i1bc)+ibcxmf(i3bc).eq.4) then
      a(irow1 + icol11) = a(irow1 + icol11) + t(10) * dfx1
      rhs(i1) = rhs(i1) + dfx1 * (bcxmf(i1c)-xmf(i1c))
      a(irow3 + icol33) = a(irow3 + icol33) + t(10) * dfx3
      rhs(i3) = rhs(i3) + dfx3 * (bcxmf(i3c)-xmf(i3c))
    C
    C— Save the flux terms for the mass balance. Also compute the
    C— surface flux for output.
    C
      if(.not.lctrl(1)) then
        f1xmf = xmf(i1c)
        f3xmf = xmf(i3c)
        fsave = abc31 * f1xmf + abc33 * f3xmf
+         + dfx1 * (bcxmf(i1c)-xmf(i1c))
+         + dfx3 * (bcxmf(i3c)-xmf(i3c))
        flux(5+ixc) = flux(5+ixc) + fsave
        if(znode(i1).eq.zero.and.znode(i3).eq.zero)
+         sflux(5+ixc) = sflux(5+ixc) + fsave
    C
    C— Compute the flux for the master component.
    C
      if(icompeq.1) then
        fsave = abc31 * xmf(nxc+i1) + abc33 * xmf(nxc+i3)
        flux(5+ixc) = flux(5+ixc) + fsave
        if(znode(i1).eq.zero.and.znode(i3).eq.zero)
+         sflux(5+ixc) = sflux(5+ixc) + fsave
      end if
      end if
    else if(.not.lctrl(1).and.
+     ibcxmf(i1bc)+ibcxmf(i3bc).eq.6) then
      a(irow1 + icol11) = a(irow1 + icol11) + t(10) * abc31
      rhs(i1) = rhs(i1) + abc31*(bcxmf(i1c)-xmf(i1c))
      a(irow3 + icol33) = a(irow3 + icol33) + t(10) * abc33
      rhs(i3) = rhs(i3) + abc33*(bcxmf(i3c)-xmf(i3c))
    C
    C— Save the flux terms for the mass balance. Also compute the
    C— surface flux for output.
    C
      if(.not.lctrl(1)) then
        fsave = abc31 * bcxmf(i1c) + abc33 * bcxmf(i3c)
+         + dfx1 * (bcxmf(i1c)-xmf(i1c))
+         + dfx3 * (bcxmf(i3c)-xmf(i3c))
        flux(5+ixc) = flux(5+ixc) + fsave
        if(znode(i1).eq.zero.and.znode(i3).eq.zero)
+         sflux(5+ixc) = sflux(5+ixc) + fsave
    C
    C— Compute the flux for the master component.
    C
      if(icompeq.1) then
        fsave = abc31*bcxmf(nxc+i1) + abc33*bcxmf(nxc+i3)

```

```

      flux(5+ixc) = flux(5+ixc) + fsave
      if(znode(i1).eq.zero.and.znode(i3).eq.zero)
+       sflux(5+ixc) = sflux(5+ixc) + fsave
    end if
  end if
end if
end if
if(((ibcxmf(i2bc)+ibcxmf(i3bc).eq.6).or.
+ (ibcxmf(i2bc)+ibcxmf(i3bc).eq.4))
+ .and.(xnode(i2).eq.xnode(i3)
+ .or.znode(i2).eq.znode(i3))) then
  if(ipt(27).eq.1) then
    weight = xnode(i3) + xnode(i2)
    weight1 = xnode(i2) / weight
    weight3 = xnode(i3) / weight
  else
    weight1 = 0.50d0
    weight3 = 0.50d0
  end if
  dx = xnode(i3)-xnode(i2)
  dz = znode(i3)-znode(i2)
  rlen = dsqrt(dx**2+dz**2)
  if(ipt(27).eq.1) then
    rrad = pi*(xnode(i2)+xnode(i3))
    rlen = rlen*rrad
  end if
  qnorm = zero
  if(lctrl(18)) then
    qx = ( q(iptq+i3) + q(iptq+i2) ) / 2.0d0
    qz = ( q(iptq+ipt1+i3) + q(iptq+ipt1+i2) ) / 2.0d0
  end if
  if(dx.gt.zero) then
    qnorm=(dx*qz)/rlen
  else
    qnorm=(-dx*qz)/rlen
  end if
  if(dz.gt.zero) then
    qnorm = qnorm + (dz*qx)/rlen
  else
    qnorm = qnorm + (-dz*qx)/rlen
  end if
  if(dabs(qnorm).lt.(den(i2p)*dfxmf(i2c)*weight2+
+ den(i3p)*dfxmf(i3c)*weight3)) then
    dfx2=den(i2p)*dfxmf(i2c)*rlen*weight2
    dfx3=den(i3p)*dfxmf(i3c)*rlen*weight3
  else
    dfx2 = zero
    dfx3 = zero
  end if
  abc32 = den(i2p)*qnorm*rlen*weight2
  abc33 = den(i3p)*qnorm*rlen*weight3
  if(ibcxmf(i1bc)+ibcxmf(i3bc).eq.4) then
    a(irow2 + icol22) = a(irow2 + icol22) + t(10) * dfx2
    rhs(i2) = rhs(i2) + dfx2 * (bcxmf(i2c)-xmf(i2c))
    a(irow3 + icol33) = a(irow3 + icol33) + t(10) * dfx3
    rhs(i3) = rhs(i3) + dfx3 * (bcxmf(i3c)-xmf(i3c))
  C
  C— Save the flux terms for the mass balance.
  C
    if(.not.lctrl(1)) then
      f2xmf = xmf(i2c)
      f3xmf = xmf(i3c)
      fsave = abc32 * f2xmf + abc33 * f3xmf
+       + dfx2 * (bcxmf(i2c)-xmf(i2c))
+       + dfx3 * (bcxmf(i3c)-xmf(i3c))
      flux(5+ixc) = flux(5+ixc) + fsave
      if(znode(i2).eq.zero.and.znode(i3).eq.zero)
+       sflux(5+ixc) = sflux(5+ixc) + fsave
    C
    C— Compute the flux for the master component.
    C
      if(icompeq.1) then
        fsave = abc32 * xmf(nxc+i2) + abc33 * xmf(nxc+i3)
        flux(5+ixc) = flux(5+ixc) + fsave
        if(znode(i2).eq.zero.and.znode(i3).eq.zero)
+         sflux(5+ixc) = sflux(5+ixc) + fsave
      end if
    end if

```

```

else if(.not.lctrl(1).and.
+   ibcxmf(i2bc)+ibcxmf(i3bc).eq.6) then
a(irow2 + icol22) = a(irow2 + icol22) + t(10) * abc32
rhs(i2) = rhs(i2) + abc32*(bcxmf(i2c)-xmf(i2c))
a(irow3 + icol33) = a(irow3 + icol33) + t(10) * abc33
rhs(i3) = rhs(i3) + abc33*(bcxmf(i3c)-xmf(i3c))
C
C— Save the flux terms for the mass balance.
C
fsave = abc32 * bcxmf(i2c) + abc33 * bcxmf(i3c)
flux(5+ic) = flux(5+ic) + fsave
if(znode(i2).eq.zer0.and.znode(i3).eq.zer0)
+   sflux(5+ic) = sflux(5+ic) + fsave
C
C— Compute the flux for the master component.
C
if(icompeq.1) then
fsave = abc32*bcxmf(nxc+i2) + abc33*bcxmf(nxc+i3)
flux(5+ixc) = flux(5+ixc) + fsave
if(znode(i2).eq.zer0.and.znode(i3).eq.zer0)
+   sflux(5+ixc) = sflux(5+ixc) + fsave
end if
end if
end if
end if
120 continue
C
C— Calculations for first type boundary conditions.
C
ibcn = ipt(18) + ipt(19) + ipt(20)*ipm1
do 26 i = 1, ipt(19+iphase)
nbc = ibc(ibcn+i)
inode = ibcxmf(ippt+nbc)
C
C— Skip specification of first type boundary condition if the node
C— number is zero.
C
if(inode.eq.1) then
C
C— Compute flux at first type nodes for mass balance before the
C— boundary condition is imposed. Computing the mass balance here
C— lags the mole fractions one sweep.
C
if(iprnt(6)) then
fsave = 0.d0
do 800 irow = 1, ipt1
if(irow.le.nbw(0)) then
jstr = nbw(0) + 1 - irow
jend = nbw(1)
else if(irow.ge.ip11-nbw(0)) then
jstr = 1
jend = nbw(1) - nbw(0) + ip11 - irow
else
jstr = 1
jend = nbw(1)
endif
sum = zer0
do 805 j = jstr,jend
805 sum = sum + a(nbw(1)*irow+j)
+ * xmf(icpt+irow-nbw(0)+j)
800 fsave = fsave + (sum-rhs(irow)) * t(8)
if(.not.lctrl(1)) then
flux(ic+5) = flux(ic+5) + fsave
if(znode(nbc).eq.0.d0)
+   sflux(ic+5) = sflux(ic+5) + fsave
end if
first(ip1c) = first(ip1c)+(sum-rhs(irow)) * t(8)
end if
C
C— Impose first type boundary conditions.
C
rhs(ibc(ibcn+i)) = zer0
nbw0 = nbw(0)
nbw1 = nbw(1)
do 27 j = 1,nbw(1)
a((nbc-1)*nbw1+j) = zer0
27 continue
a((nbc-1)*nbw1+1+nbw0) = rone

```

```

end if
26 continue
C
C— Use solution from flow section to impose boundary fluxes for third
C— type boundaries. First type pressure nodes.
C
if(lctrl(1)) then
do 28 i = 1, ipt(18+ipm1)
nbc = ibc(i+ipm1)*ipt(18))
nbcpc = nbc + ippt
irowbc = (nbc-1)*nbw(1)
icolbc = 1 + nbw(0)
nbcpc = icpt + nbc
if(ibcxmf(nbcpc).eq.3) then
porsum = zer0
do 32 ipor = 1, nodept(i+1)-nodept(i)
32 porsum = porsum + por(matpt(nodept(i)+ipor))
porsum = porsum/dble(nodept(i+1)-nodept(i))
a(irowbc + icolbc) = a(irowbc + icolbc) + t(10)
+ * bcf(nbcpc) * den(nbcpc)
+ rhs(nbc) = rhs(nbc) + bcf(nbcpc) * den(nbcpc)
+ * (bcxmf(nbcpc) - xmf(nbcpc))
end if
28 continue
C
C— Third type nodes with constant volumetric flux.
C
do 29 i = 1, ipt(22+ipm1)
nbc = ibc(i+ipm1)*ipt(22)+ipt(62))
nbcpc = nbc + ippt
irowbc = (nbc-1)*nbw(1)
icolbc = 1 + nbw(0)
nbcpc = icpt + nbc
if(ibcxmf(nbcpc).eq.3) then
porsum = zer0
do 33 ipor = 1, nodept(i+1)-nodept(i)
33 porsum = porsum + por(matpt(nodept(i)+ipor))
porsum = porsum/dble(nodept(i+1)-nodept(i))
a(irowbc + icolbc) = a(irowbc + icolbc) + t(10)
+ * bcf(nbcpc) * den(nbcpc)
+ rhs(nbc) = rhs(nbc) + bcf(nbcpc) * den(nbcpc)
+ * (bcxmf(nbcpc) - xmf(nbcpc))
end if
29 continue
C
C— Well nodes.
C
do 31 i = 1, ipt(24)
nbc = ibc(i+ipt(64))
nbcpc = nbc + ippt
irowbc = (nbc-1)*nbw(1)
icolbc = 1 + nbw(0)
nbcpc = icpt + nbc
if(ibcxmf(nbcpc).eq.3) then
porsum = zer0
do 34 ipor = 1, nodept(i+1)-nodept(i)
34 porsum = porsum + por(matpt(nodept(i)+ipor))
porsum = porsum/dble(nodept(i+1)-nodept(i))
a(irowbc + icolbc) = a(irowbc + icolbc) + t(10)
+ * bcf(nbcpc) * den(nbcpc)
+ rhs(nbc) = rhs(nbc) + bcf(nbcpc) * den(nbcpc)
+ * (bcxmf(nbcpc) - xmf(nbcpc))
end if
31 continue
end if
C
C— Collapse full matrix into sparse form used by Harwell. Also
C— scale array by dividing rows through by the diagonal value.
C
ia = 0
do 30 irow = 1, ipt(1)
nrow = (irow-1)*nbw(1)
if(a(nrow+1+nbw(0)).eq.zer0) a(nrow+1+nbw(0)) = rone
a(i) = rone / a(nrow+1+nbw(0))
rhs(irow) = rhs(irow) * a(i)
do 30 icol = 1,nbw(1)
if (a(nrow+icol).ne. zer0) then
ia = ia + 1

```

```

a(ia) = a(nrow+icol) * ait
irn(ia) = irow
icn(ia) = icol+irow-nbw(0)-1
endif

```

```

30 continue
return
end

```

Subroutine - trans.f

```

C-----
C
C TRANS.f - Subroutine which solves the component molar balance
C equations. Solution is done sequentially starting
C with the most volatile component and moving to the
C least volatile.

```

```

C Arguments: its - integer time step number
C iconv - integer flag for global convergence

```

```

C Required Control Flags:

```

```

C t(14) - convergence criterion for mole fractions
C t(23) - maximum gas phase cell Peclet number
C t(24) - maximum aqueous phase cell Peclet number
C t(25) - maximum gas phase cell Courant number
C t(26) - maximum aqueous phase cell Courant number
C ipt(32) - maximum component balance iterations, also used as
C the criterion for decreasing dt in component balance
C routines
C ipt(35) - maximum number of iterations in component balance
C routines for increasing dt
C lctrl(3) - logical variable controlling inclusion of
C biodegradation
C lctrl(3) = .true. - include biodegradation
C lctrl(3) = .false. - neglect biodegradation
C lctrl(4) - logical variable controlling printing of
C element dimensionless numbers
C lctrl(4) = .true. - print element
C dimensionless numbers
C lctrl(4) = .false. - skip element
C dimensionless numbers
C lctrl(9) - logical variable denoting presence of nutrient
C lctrl(9) = .true. - nutrient considered
C lctrl(9) = .false. - nutrient not considered
C lctrl(15) - logical variable controlling time series output
C lctrl(15) = .true. - output time series
C lctrl(15) = .false. - do not output time series
C lctrl(21) - logical variable controlling calculation of
C the hydrodynamic dispersion tensor
C lctrl(21) = .true. - calculate tensor
C lctrl(21) = .false. - read tensor as input

```

```

C Control Flags computed internally in routine:

```

```

C ipt(37) - flag specifying time step modification

```

```

C subroutine TRANS(its,ibconv)
C include 'dimen.inc'
C character*10 cname(ncmp)

```

```

C----- Declare and define common block variables.

```

```

C common /cb3/ sat(nnstk3)
C common /cb3b/ satt(nnstk3)
C common /cb7b/ cmw(ncmp),cvp(ncmp),cden(ncmp),
C + chen(ncmp),caso(ncmp),cmdif(ncmp2)
C common /cb9/ xmf(nmf)
C common /cb9b/ xmfi(nmf)
C common /cb10/ den(nn6)
C common /cb40/ a(icnl),rhs(ksolve),w(icnl)
C common /cb41/ irn(icnl),icn(icnl),iw(icnl,8),ikeep(icnl,5)
C common /cb41b/ nbw(0:2),ia
C common /cb85/ flux(ncmpp5),sflux(ncmpp5),first(ncmp)

```

```

common /cb91/ cname
data lplot / .false. /

```

```

C----- Initialize variables as needed.
C
C ipt(37)=0
C ipt1 = ipt(1)
C ipt2 = ipt(2)
C ipt3 = ipt(3)
C
C----- Determine which mobile phases are present.
C
C imob1=1
C imob2=2
C if(ipt(3).eq.0) imob1=2
C if(ipt(4).eq.0) imob2=1
C
C----- Update the biological reaction terms. These terms are lagged one
C time step.
C
C if(lctrl(3).and.lctrl(16)) then
C iconv = 0
C if(ipt(28).gt.0) write(ipt(28),*) ' BIO lagged'
C call BIO(tconv,ibconv)
C else
C if(lctrl(3).and.ipt(28).gt.0) write(ipt(28),*)
C + ' BIO separate phase included'
C end if
C
C----- Begin iterations over the set of component balance equations.
C
C ipass = 0
C do 200 it = 1, ipt(32)
C iconv = 0
C
C----- Zero the component mass flux entry for the mass balance.
C
C do 130 i=3,ncmpp5
C sflux(i)=zer0
C 130 flux(i)=zer0
C
C----- Update the biological phase mole fractions every iteration when
C considering a separate biophase.
C
C if(lctrl(3).and..not.lctrl(16)) call BIO(iconv,ibconv)
C
C----- Update the phase molecular weights and densities every iteration.
C
C call MOLEWT
C
C----- Update the phase mole exchange terms every iteration.
C
C call MPEX
C ipass = ipass + 1
C do 100 iphase = imob1,imob2
C t(22+iphase) = zer0
C t(24+iphase) = zer0
C
C----- ipt+1 is the first entry of phase vectors, iptps+1 is the first
C entry of stacked phase vectors,
C
C ipm1 = iphase-1
C
C----- Use DISPER.f to compute the dispersivities if desired.
C

```

```

    if (lctrl(21)) call DISPER(ipphase)
    do 110 icomp = 1, ipt(2+ipphase)
C
C— ic is the identity of the icomp component in the ipphase phase,
C— iptc+1 is the first entry of the icomp component in the ipphase
C— phase.
C
    ic = icp(ipm1*ipt3+icomp)
    iptc = ipt(7+ipphase)+(icomp-1)*ipt(1)
C
C— Compute the pointers to the principal component of each phase.
C
    if(ipphase.eq.1) then
    if (lctrl(9)) then
        ixc = icp(ipt3-1)
        nxc = (ipt3-2)*ipt(1)
    else
        ixc = icp(ipt3)
        nxc = (ipt3-1)*ipt(1)
    end if
    else if(ipphase.eq.2) then
        ipt314 = ipt3 + ipt(14)
        ixc = icp(ipt314+1)
        nxc = (ipt3+ipt(14))*ipt(1)
    end if
C
C— Skip the principal component of each mobile phase (i.e. nitrogen
C— for the gas phase and water for the aqueous phase)
C
    if (ic.eq.ixc) goto 110
C
C— Use TLHS.f to form and assemble the finite element matrices.
C
    call TLHS(ipphase, icomp, ixc, nxc)
C
C— Solve the linear system using Harwell routines.
C
    call ma28ad(ipt(1), ia, a, icnl, irn, irnl, icn, u, ikeep, iw, w, iflag)
    if (iflag .lt. 0) then
        write (ipt(28), *) 'iflag return from harwell is ', iflag
        write (ipt(28), *) 'trans phase/component are: ', ipphase
    +
    + ', ', cname(ic)
    end if
    call ma28cd (ipt(1), a, icnl, icn, ikeep, rhs, w, mtype)
C
C— Update the solution and determine the max norm of the updated
C— solution.
C
    dxmf = zero
    xmfmax = xround
    do 210 i = 1+iptc, ipt(1)+iptc
        xmfold = xmf(i)
        xmf(i) = xmf(i) + rhs(i-iptc)
        xmfmax = dmax1(xmfmax, dabs(xmf(i)))
    210 dxmf = dmax1(dxmf, dabs(xmf(i)-xmfold))
C
C— Check convergence with the relative change in state variable.
C— Update the solution.
C
    if(dxmf/xmfmax.gt.t(14)) iconv = iconv + 1
    110 continue
C
C— Update the mole fraction of the principal component of each
C— phase by the mole fraction constraint after all the components
C— of that phase have been updated.
C
    do 300 i=1, ipt(1)
        xmf(nxc+i) = rone
        do 300 ii=1, ipt(2+ipphase)
            if(icp(ipt3*ipm1+ii).ne.ixc) then
                ipt7p = ipt(7 + ipphase)
                xmf(nxc+i) = xmf(nxc+i)
    +
    - xmf(ipt7p+(ii-1)*ipt1+i)
            end if
    300 continue
    100 continue
C
C— Update the NAPL mole fractions. This is only needed

```

```

C— If there is more than 1 organic component.
C
    if(ipt(5).gt.1) then
        call NAPLX(iconv)
    end if
C
C— Update the solid phase mass fractions.
C
    if(ipt(6).gt.0) call SOLID(iconv)
C
C— Transport equation set is converged if each component equation
C— has converged (i.e. iconv = 0).
C
    if (iconv.eq.0) then
C
C— Use NAPLS.f to update the organic phase saturation.
C
        if(lctrl(24)) call NAPLS(iconv)
C
C— Reset mobile phase saturation if the flow simulator is not
C— being used.
C
        if(.not.lctrl(1)) then
            do 175 i=1, ipt(2)
                ipt49 = ipt(49)
                if(ipt(3).gt.0) then
                    if(sat(i).gt.zer0) then
                        sat(i) = sat(i)+satt(ipt49+i)-sat(ipt49+i)
                    if (sat(i) .lt. zer0) then
                        sat(ipt2+i) = sat(ipt2+i) + sat(i)
                        sat(i) = zer0
                    endif
                else
                    sat(ipt2+i) = sat(ipt2+i) + satt(ipt49+i)
    +
    - sat(ipt49+i)
                end if
            else
                sat(ipt2+i) = sat(ipt2+i) + satt(ipt49+i)
    +
    - sat(ipt49+i)
            end if
    175 continue
        end if
C
C— Increase the time step if the number of iterations is less than
C— or equal to ipt(35).
C
        if (it.le.ipt(35)) ipt(37)=1
C
C— Write plotting data. This is restricted to 6 components from both
C— mobile phases.
C
        if(lctrl(15)) then
            if(lprnt(26)) then
                if(mod(it,ipt(84)).eq.0) lplot = .true.
            else if(.not.lprnt(26)) then
                if (int(t(9)/t(28)).gt.int((t(9)-t(8))/t(28)))
    +
    lplot = .true.
            end if
            if(lplot) then
                lplot = .false.
                if(lprnt(28)) then
                    write(26,776) t(9)
    +
    + ,(xmf(icp(i)), i=ipt(61)+2*ipt(74)+2
    + ,ipt(61)+2*ipt(74)+2*(ipt(81)+ipt(82)),2)
                else
                    write(26,776) t(9)
    +
    + ,(xmf(icp(i)) * cmw(icp((icp(i)-1)/ipt(1)+1))
    + * den(icp(i)-ipt(1))*((icp(i)-1)/ipt(1)))
    + ,i=ipt(61)+2*ipt(74)+2
    + ,ipt(61)+2*ipt(74)+2*ipt(81),2)
    + ,(xmf(icp(i)) * cmw(icp((icp(i)-1)/ipt(1)+1))
    + * den(icp(i)-ipt(1))*((icp(i)-1)/ipt(1)-1))
    + ,i=ipt(61)+2*ipt(74)+2*ipt(81)+2
    + ,ipt(61)+2*ipt(74)+2*(ipt(81)+ipt(82)),2)
            end if
        end if
    end if
C

```



```

i1s = nodept(i1)+nelpt(iel1)
i1x = ip2s + i1s
i1z = i1x + ipt(2)
i2 = node1(iel2)
i2s = nodept(i2)+nelpt(iel2)
i2x = ip2s + i2s
i2z = i2x + ipt(2)
i3 = node1(iel3)
i3s = nodept(i3)+nelpt(iel3)
i3x = ip2s + i3s
i3z = i3x + ipt(2)
C
C— Average mobilities and densities, obtaining centroid values
C— for each element. Mobilities are stacked. Mass densities
C— are used for this expression.
C
xmob = third * (pmob(i1x) + pmob(i2x) + pmob(i3x))
zmob = third * (pmob(i1z) + pmob(i2z) + pmob(i3z))
C
C— If an element has a gas saturation below sgtest, set the mobility
C— of that element equal to zero.
C
if(ipphase.eq.1) then
  if((sat(i1s).lt.sgtest).or.(sat(i2s).lt.sgtest)
+   .or.(sat(i3s).lt.sgtest)) then
    xmob = zero
    zmob = zero
  end if
else
  if((sat(i1s+ipt2).lt.srw(i1s)+srwmin)
+   .or.(sat(i2s+ipt2).lt.srw(i2s)+srwmin)
+   .or.(sat(i3s+ipt2).lt.srw(i3s)+srwmin)) then
    xmob = zero
    zmob = zero
  end if
end if
d1 = den(i1+ip3d)
d2 = den(i2+ip3d)
d3 = den(i3+ip3d)
C
C— Calculate element average density in a manner consistent with
C— the calculation of the pressure gradient.
C
davgx = (d1*dabs(beta(iel1)) + d2*dabs(beta(iel2))
+   + d3*dabs(beta(iel3)))/(dabs(beta(iel1))+
+   dabs(beta(iel2))+dabs(beta(iel3)))
davgz = (d1*dabs(gama(iel1)) + d2*dabs(gama(iel2))
+   + d3*dabs(gama(iel3)))/(dabs(gama(iel1))+
+   dabs(gama(iel2))+dabs(gama(iel3)))
C
C— Calculate pressure heads.
C
p1 = p(i1+ip)
p2 = p(i2+ip)
p3 = p(i3+ip)
h1 = p1 - d1*t(22)*znode(i1)
h2 = p2 - d2*t(22)*znode(i2)
h3 = p3 - d3*t(22)*znode(i3)
C
C— Compute specific discharge.
C
q(ix) = -xmob * davgx * ((h1*beta(iel1) + h2*beta(iel2)
+   + h3*beta(iel3)) / (2.0d0*area(i)))
q(iz) = -zmob * davgz * ((h1*gama(iel1) + h2*gama(iel2)
+   + h3*gama(iel3)) / (2.0d0*area(i)))
100 continue
return
C
C— Finite element q computation. First, initialize q with
C— zeros.
C
200 do 210 i=1,ipt(42)
  q(i)=zer0
210 continue
C
C— Now iterate over the mobile phases.
C
imob1=1

```

```

imob2=2
if(ipt(3).eq.0) imob1=2
if(ipt(4).eq.0) imob2=1
do 220 iphase = imob1,imob2
C
C— Zero the local finite element matrices.
C
do 230 i = 1, ipt(1)
  rhsx(i) = zer0
  rhsz(i) = zer0
  nrow = (i-1)*nbw(1)
  do 230 j = 1, nbw(1)
    az(nrow+j) = zer0
230   a(nrow+j) = zer0
C
C— Set pointers. ip points to non-stacked phase properties, ipq
C— points to q, ip2 points to den, ip2s points to pmob.
C
ipt1 = ipt(1)
ipt2 = ipt(2)
ipm1 = iphase-1
ip = ipm1 * ipt(1)
ip2 = ipm1 * ipt(40)
ip2s = ipm1 * ipt(49)
ip3d = ipm1 * ipt(1) + ipt(41)
do 240 i=1,ipt(0)
C
C— Set the pointers to the local nodes 11, 12, and 13. The
C— postscript s is for stacked local nodes, and p is for phase
C— Compute constants.
C
iel3 = i*3
iel1 = iel3-2
iel2 = iel3-1
i1 = node1(iel1)
i1s = nodept(i1)+nelpt(iel1)
i1x = ip2s + i1s
i1z = i1x + ipt2
i2 = node1(iel2)
i2s = nodept(i2)+nelpt(iel2)
i2x = ip2s + i2s
i2z = i2x + ipt2
i3 = node1(iel3)
i3s = nodept(i3)+nelpt(iel3)
i3x = ip2s + i3s
i3z = i3x + ipt2
C
C— If an element has a gas saturation below sgtest or an aqueous
C— saturation below srw+srwmin, skip calculation of the local
C— element matrix.
C
if((ipphase.eq.1).and.((sat(i1s).lt.sgtest)
+   .or.(sat(i2s).lt.sgtest).or.(sat(i3s).lt.sgtest)))
+   goto 240
if((ipphase.eq.2)
+   .and.((sat(i1s+ipt2).lt.srw(i1s)+srwmin)
+   .or.(sat(i2s+ipt2).lt.srw(i2s)+srwmin)
+   .or.(sat(i3s+ipt2).lt.srw(i3s)+srwmin)))
+   goto 240
C
C— Compute the left hand side matrix.
C
aoff = aby12(i)
aon = 2.0d0 * aoff
C
C— Calculate pressure heads.
C
p1 = p(i1+ip)+patm
p2 = p(i2+ip)+patm
p3 = p(i3+ip)+patm
d1 = den(i1+ip3d)
d2 = den(i2+ip3d)
d3 = den(i3+ip3d)
if(ipphase.eq.1) then
  dg10 = pmw0(i1)*patm/(r*temp(i1))
  dg20 = pmw0(i2)*patm/(r*temp(i2))
  dg30 = pmw0(i3)*patm/(r*temp(i3))

```

```

b1 = p1/(dg10*t(22)) - znode(i1)
h2 = p2/(dg20*t(22)) - znode(i2)
h3 = p3/(dg30*t(22)) - znode(i3)
else
h1 = p1 - d1*t(22) * (znode(i1)-wtdpth)
h2 = p2 - d2*t(22) * (znode(i2)-wtdpth)
h3 = p3 - d3*t(22) * (znode(i3)-wtdpth)
end if

C
C— Calculate mobility terms for the right hand side.
C
pm1x = 2.0d0*pmob(i1x) + pmob(i2x) + pmob(i3x)
pm2x = pmob(i1x) + 2.0d0*pmob(i2x) + pmob(i3x)
pm3x = pmob(i1x) + pmob(i2x) + 2.0d0*pmob(i3x)
pm1z = 2.0d0*pmob(i1z) + pmob(i2z) + pmob(i3z)
pm2z = pmob(i1z) + 2.0d0*pmob(i2z) + pmob(i3z)
pm3z = pmob(i1z) + pmob(i2z) + 2.0d0*pmob(i3z)

C
C— Calculate pressure gradient terms for the right hand side.
C
if(ipphase.eq.1) then
term = -t(22) * rbar(i) / 48.0d0
else
term = -rbar(i) / 48.0d0
end if
dpx = (h1*beta(iel1)*dg10 + h2*beta(iel2)*dg20
+ h3*beta(iel3)*dg30) * term
dpz = (h1*gama(iel1)*dg10 + h2*gama(iel2)*dg20
+ h3*gama(iel3)*dg30) * term
f1x = pm1x * dpx
f2x = pm2x * dpx
f3x = pm3x * dpx
f1z = pm1z * dpz
f2z = pm2z * dpz
f3z = pm3z * dpz

C
C— Calculate mobility/gravity terms for the right hand side.
C
if(ipphase.eq.1) then
term = t(22) * aby12(i) / 2.0d0
dm1z = pmob(i1z) * (( patm*d1 / (p1*dg10)) - rone)*dg10
dm2z = pmob(i2z) * (( patm*d2 / (p2*dg20)) - rone)*dg20
dm3z = pmob(i3z) * (( patm*d3 / (p3*dg30)) - rone)*dg30
f1z = f1z + term * ( 2.0d0*dm1z + dm2z + dm3z )
f2z = f2z + term * ( dm1z + 2.0d0*dm2z + dm3z )
f3z = f3z + term * ( dm1z + dm2z + 2.0d0*dm3z )
else
term = aby12(i) / 2.0d0
dm1z = pmob(i1z) * (d1 - den0(i1)) * t(22)
dm2z = pmob(i2z) * (d2 - den0(i2)) * t(22)
dm3z = pmob(i3z) * (d3 - den0(i3)) * t(22)
f1z = f1z + term * ( 2.0d0*dm1z + dm2z + dm3z )
f2z = f2z + term * ( dm1z + 2.0d0*dm2z + dm3z )
f3z = f3z + term * ( dm1z + dm2z + 2.0d0*dm3z )
end if

C
C— Now assemble the global matrix and right hand side vector in
C— banded form.
C
rhsx(i1) = rhsx(i1) + f1x
rhsx(i2) = rhsx(i2) + f2x
rhsx(i3) = rhsx(i3) + f3x
rhsz(i1) = rhsz(i1) + f1z
rhsz(i2) = rhsz(i2) + f2z
rhsz(i3) = rhsz(i3) + f3z
irow1 = (i1-1)*nbw(1)
irow2 = (i2-1)*nbw(1)
irow3 = (i3-1)*nbw(1)
icol11 = 1 + nbw(0)
icol12 = icol11 + (i2 - i1)
icol13 = icol11 + (i3 - i1)
icol22 = icol11
icol21 = icol22 + (i1 - i2)
icol23 = icol22 + (i3 - i2)
icol33 = icol11
icol31 = icol33 + (i1 - i3)
icol32 = icol33 + (i2 - i3)
a(irow1 + icol11) = a(irow1 + icol11) + aon

```

```

a(irow1 + icol12) = a(irow1 + icol12) + aoff
a(irow1 + icol13) = a(irow1 + icol13) + aoff
a(irow2 + icol21) = a(irow2 + icol21) + aoff
a(irow2 + icol22) = a(irow2 + icol22) + aon
a(irow2 + icol23) = a(irow2 + icol23) + aoff
a(irow3 + icol31) = a(irow3 + icol31) + aoff
a(irow3 + icol32) = a(irow3 + icol32) + aoff
a(irow3 + icol33) = a(irow3 + icol33) + aon
240
C
C— Set up temporary mass matrix.
C
do 25 i = 1, ipt(1)*nbw(1)
25 az(i) = a(i)
C
C— Impose boundary conditions.
C
nbw0 = nbw(0)
nbw1 = nbw(1)
do 26 i = 1, ipt(1)
lvelx = .false.
lvelz = .false.
C
C— No z-velocity normal to the bottom.
C
if(lctrl(28).and.znode(i).eq.znode(ipt(1))) then
rhsz(i) = zero
lvelz = .true.
end if
C
C— No x-velocity normal to the R.H.S. boundary.
C
if(lctrl(29).and.xnode(i).eq.xnode(ipt(1))) then
rhsx(i) = zero
lvelx = .true.
end if
C
C— No x-velocity normal to the L.H.S. boundary. Note that this
C— boundary will be adjusted in the presence of a well.
C
if(lctrl(30).and.xnode(i).eq.xnode(1)) then
rhsx(i) = zero
lvelx = .true.
C
C— Adjust velocity boundary condition in the presence of a well.
C
if(lctrl(12)) then
do 28 iwell = 1, ipt(24)
inw = ibc(ipt(64)+iwell)
if(inw.eq.i) then
if(ipphase.eq.1) then
rhsx(i) = bcf(inw)
else if(ipphase.eq.2.and.qwell.lt.zero) then
rhsx(i) = bcf(inw+ipt(1))
end if
end if
28 continue
end if
end if
C
C— No z-velocity normal to the top. This boundary extends from the
C— well to caplen.
C
if(lctrl(31).and.znode(i).eq.znode(1)
+ .and.xnode(i).le.caplen) then
rhsz(i) = zero
lvelz = .true.
end if
if(lvelx.or.lvelz) then
do 27 j = 1,nbw(1)
if(lvelx) a((i-1)*nbw1+j) = zero
if(lvelz) az((i-1)*nbw1+j) = zero
27 continue
if(lvelx) a((i-1)*nbw1+1+nbw0) = rone
if(lvelz) az((i-1)*nbw1+1+nbw0) = rone
end if
lvelx = .false.

```



```

      lvelz = .false.
26  continue
C
C— Collapse full matrix into sparse form used by Harwell. Also
C— scale array by dividing rows through by the diagonal value.
C
ia = 0
iaz = 0
do 260 irow = 1, ipt(1)
  nrow = (irow-1)*nbw(1)
  if(a(nrow+1+nbw(0)).eq.zer0) then
    a(nrow+1+nbw(0)) = rone
    rhsx(irow) = zer0
  end if
  if(az(nrow+1+nbw(0)).eq.zer0) then
    az(nrow+1+nbw(0)) = rone
    rhsz(irow) = zer0
  end if
  aii = rone / a(nrow+1+nbw(0))
  aii = rone / az(nrow+1+nbw(0))
  rhsx(irow) = rhsx(irow) * aii
  rhsz(irow) = rhsz(irow) * aii
do 270 icol = 1, nbw(1)
  if (a(nrow+icol) .ne. zer0) then
    ia = ia + 1
    a(ia) = a(nrow+icol) * aii
    icn(ia) = icol+irow-nbw(0)-1
    irn(ia) = irow
  endif
  if (az(nrow+icol) .ne. zer0) then

```

```

    iaz = iaz + 1
    az(iaz) = az(nrow+icol) * aii
    icnz(iaz) = icol+irow-nbw(0)-1
    irnz(iaz) = irow
  endif
270 continue
260 continue
C
C— Solve the linear system using Harwell routines.
C
  call ma28ad(ipt(1),ia,a,icnl,irn,irnl,icn,u,ikeep,iw,w,iflag)
  if (iflag .lt. 0)
+   write (ipt(28),*) 'vel iflag return from harwell is ',iflag
  call ma28cd (ipt(1),a,icnl,icn,ikeep,rhsx,w,mtype)
  call
+ ma28ad(ipt(1),iaz,az,icnl,irnz,irnl,icnz,u,ikeepz,iw,wz,iflag)
  if (iflag .lt. 0)
+   write (ipt(28),*) 'vel iflag return from harwell is ',iflag
  call ma28cd (ipt(1),az,icnl,icnz,ikeepz,rhsz,wz,mtype)
C
C— Update nodal values of q.
C
  do 280 i = 1, ipt(1)
    q(i+ip2)=rhsx(i)
280  q(i+ip2+ipt1)=rhsz(i)
220 continue
    return
  end

```

REFERENCES

- Abriola, L.M., Multiphase Migration of Organic Compounds in a Porous Medium. A Mathematical Model, in *Lecture Notes in Engineering*, (C.A. Brebbia and S.A. Orszag eds.), 8, Springer-Verlag, Berlin, 1984.
- Abriola, L.M. and G.F. Pinder, A multiphase approach to the modeling of porous media contamination by organic compounds: 2. Numerical simulation, *Water Resour. Res.*, 21(1), 19-26, 1985.
- Abriola, L.M., Multiphase Flow and Transport Models for Organic Chemicals: A Review and Assessment, EPRI EA-5976, Electric Power Research Institute, Palo Alto, 1988.
- Abriola, L.M., Modeling multiphase migration of organic chemicals in groundwater systems - A review and assessment, *Environ. Health Perspect.*, 83, 117-143, 1989.
- Abriola, L.M., K. Rathfelder, M. Maiza and S. Yadav, VALOR Code Version 1.0: A PC Code for Simulating Immiscible Contaminant Transport in Subsurface Systems, EPRI TR-101018, Electric Power Research Institute, Palo Alto, 1992.
- Abriola, L.M. and K. Rathfelder, Mass balance errors in modeling two-phase immiscible flows: Causes and remedies, *Advances in Water Resources*, 16, 223-239, 1993.
- Abu-El-Sha'r, W.Y., Experimental Assessment of Multicomponent Gas Transport Flux Mechanisms in Subsurface Systems, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, 1993.
- Adenekan, A.E., T.W. Patzek and K. Pruess, Modeling of multiphase transport of multicomponent organic contaminants and heat in the subsurface: Numerical model formulation, *Water Resour. Res.*, 29(11), 3727-3740, 1993.
- Agrelot, J.C., J.J. Malot, and M.J. Visser, Vacuum: Defense System for Groundwater VOC Contamination, in Proc. Fifth National Symposium and Exposition on Aquifer Restoration and Ground Water Monitoring, Columbus, Ohio, 485-494, National Water Well Association, 1985.
- Armstrong, J.E., J. Croise and V. Kaleris, Simulation of rate-limiting processes controlling the vapour extraction of trichloroethylene in sandy soils, *Proc. Con. The Environment and Geotechnics*, April 6-8, Paris, 1993.
- Armstrong, J.E., E.O. Frind and R.D. McClellan, Nonequilibrium mass transfer between the vapor, aqueous, and solid phases in unsaturated soils during vapor extraction, *Water Resour. Res.*, 30(2), 355-368, 1994.
- Atlas, R.M. and R. Bartha, *Microbial Ecology*, 2nd. Ed., Menlo Park, California, The Benjamin/Cummings Publishing Company, 1987.
- Aziz, K. and A. Settari, *Petroleum Reservoir Simulation*, Applied Science Publishers, London, 1979.
- Baehr, A.L., Immiscible Contaminant Transport in Soils with an Emphasis on Gasoline Hydrocarbons, Ph.D. Dissertation, Department of Civil Engineering, University of Delaware, 1984.
- Baehr, A.L., G.E. Hoag and M.C. Marley, Removing volatile contaminants from the unsaturated zone by inducing advective air-phase transport, *J. Contam. Hydrol.*, 4, 1-26, 1989.
- Baehr, A.L. and M.F. Hult, Evaluation of unsaturated zone air permeability through pneumatic tests, *Water Resour. Res.*, 27(10), 2605-2617, 1991.
- Baehr, A.L., J.M. Fischer, M.A. Lahvis, R.J. Baker and N.P. Smith, Method for estimating rates of microbial degradation of hydrocarbons based on gas transport in the unsaturated zone at a gasoline-spill site in Galloway Township, New Jersey, Proceedings of the Technical Meeting of the U.S. Geological Survey Toxic Substances Hydrology Program, Monterey, CA, March 11-15, *Water Resources Investigations Report 91-4034*, 250-55, 1991.
- Baehr, A.L., Joss, C.J., and M.F. Hult, An updated model of induced air flow in the unsaturated zone, *Water Resour. Res.*, 2(31), 417-421, 1995.
- Baker, R.S., J. Ghaemghami, S. Simkins and L.M. Mallory, A vadose column treatability test for bioventing applications, in *Hydrocarbon Bioremediation*, 32-39, Lewis Pub., Boca Raton, 1994.
- Barbee, G.C., Fate of chlorinated aliphatic hydrocarbons in the vadose zone and ground water, *Ground Water Mon. Rev.*, 129-140, Winter 1994.

- Baveye, P. and A. Valocchi, An evaluation of mathematical models of the transport of biologically reacting solutes in saturated soils and aquifers, *Water Resour. Res.*, 25(6), 1413-21, 1989.
- Bear, J., *Dynamics of Fluids in Porous Media*, Elsevier Publishing Co., New York, 1972.
- Beckett, G.D. and D. Huntley, Characterization of flow parameters controlling soil vapor extraction, *Ground Water*, 32(2), 239-247, 1994.
- Benson, D.A., P. Huntley and P.C. Johnson, Modeling vapor extraction and general transport in the presence of NAPL mixtures and nonideal conditions, *Ground Water*, 31(3), 437-445, 1993.
- Berndtson, M.J. and A.L. Bunge, A mechanistic study of forced aeration for in-place remediation of vadose zone soils, Proc. Petroleum Hydrocarbons and Organic Chemicals in Ground Water, 249-263, National Water Well Assoc., 1991.
- Bloes, M.B., K.M. Rathfelder, and D.M. Mackay, Laboratory studies of vapor extraction for remediation of contaminated soil, in *Subsurface Contamination by Immiscible Fluids*, 255-262, A.A. Balkema, Rotterdam, 1992.
- Borden, R.C. and P.B. Bedient, Transport of dissolved hydrocarbons influenced by oxygen-limited biodegradation, 1. Theoretical development, *Water Resour. Res.*, 22, 1973-82, 1986.
- Breedveld, G.D., G. Olstad, et al., Nutrient demand in bioventing of fuel oil pollution, in *In Situ Aeration: Air Sparging, Bioventing, and Related Remediation Processes* (R.E. Hinchee et al. eds.), 391-399, Battelle Press, Columbus, 1995.
- Brock, T.D., D.W. Smith and M.T. Madigan, *Biology of Microorganisms*, 4th ed., Prentice-Hall, Englewood Cliffs, NJ, 1984.
- Brusseau, M.L. and P.S.C. Rao, Sorption nonideality during organic contaminant transport in porous media, *CRC Critical Reviews in Environmental Control*, 19(1), 33-99, 1989.
- Brusseau, M.L., Transport of organic chemicals by gas advection in structured or heterogeneous porous media: Development of a model and application to column experiments, *Water Resour. Res.*, 27(12), 3189-3199, 1991.
- Brusseau, M.L., Rate-limited mass transfer and transport of organic solutes in porous media that contain immobile immiscible organic liquid, *Water Resour. Res.*, 28(1), 33-46, 1992.
- Celia, M.A., E.T. Boulouton and R.L. Zarba, A general mass-conservation numerical solution for the unsaturated flow equation, *Water Resour. Res.*, 26(7), 1483-1496, 1990.
- Chen, Y-M., L.M. Abriola, P.J.J. Alvarez, P.J. Anid and T.M. Vogel, Modeling transport and biodegradation of benzene and toluene in sandy aquifer material: Comparisons with experimental measurements, *Water Resour. Res.*, 28(7), 1833-47, 1992.
- Chen, Y-M., Mathematical Modeling of *In-Situ* Bioremediation of Volatile Organics in Variably Saturated Aquifers, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, 1996.
- Chiang, C.Y., J.P. Salanitro, E.Y. Chai, J.D. Colthart and C.L. Klein, Aerobic biodegradation of benzene, toluene, and xylene in a sandy aquifer - Data analysis and computer modeling, *Ground Water*, 27(6), 823-34, 1989.
- Cho, H.J. and P.R. Jaffe, The volatilization of organic compounds in unsaturated porous media during infiltration, *J. Contam. Hydrol.*, 6, 387-410, 1990.
- Cho, J.S. and D.C. DiGiulio, Pneumatic pumping test for soil vacuum extraction, *Environmental Progress*, 11(3), 228-233, 1992.
- Clarke, A.N., M.M. Megehee and D.J. Wilson, Soil clean up by *in situ* aeration. XII. Effect of departures from Darcy's Law on soil vapor extraction, *Separation Sci. and Tech.*, 28(9), 1671-1690, 1993.
- Cleary, R.W. and M.J. Unga, Groundwater pollution and hydrology, Mathematical models and computer programs, *Research Report No. 78-WR-15*, Water Resources Program, Princeton University, Princeton, New Jersey, 1978.
- Corey, A.T., *Mechanics of Immiscible Fluids in Porous Media*, Water Resources Publications, Fort Collins, 1986.
- Croise, J. and V. Kaleris, Field measurements and numerical simulation of pressure drop during air stripping in the vadose zone, in *Subsurface Contamination by Immiscible Fluids*, (K.U. Weyer ed.), 239-246, A.A. Balkema, Rotterdam, 1992.

- Crow, W.L., E.R. Anderson and E. Minugh, Subsurface Venting of Hydrocarbon Vapors from an Underground Aquifer, American Petroleum Institute, Washington, D.C., 1985.
- Crow, W.L., E.R. Anderson and E. Minugh, Subsurface venting of hydrocarbon emanating from hydrocarbon product on groundwater, *Ground Water Mon. Rev.*, 51-57, Winter, 1987.
- Dean, J.A. (Ed.), *Lange's Handbook of Chemistry*, 13th Ed., McGraw-Hill, New York, 1985.
- Demond, A.H. and P.V. Roberts, Effect of interfacial forces on two-phase capillary pressure relationships, *Water Resour. Res.*, 27(3), 423-437, 1991.
- DiGiulio, D.C., *Ground Water Issue - Evaluation of Soil Venting*, U.S. Environmental Protection Agency, EPA/540/S-92/004, 1992.
- Downey, D.C. and M.G. Elliott, Performance of selected in-situ soil decontamination technologies: An Air Force perspective, *Environ. Progress*, 9(3), 169-73, 1990.
- Duff, I.S., MA28 - A Set of Fortran Subroutines for Sparse Unsymmetric Linear Equations. Report No. AERE-R.8730, AERE, Harwell, 88 pp, 1979.
- Dupont, R.R., W.J. Doucette and R.E. Hinchee, Assessment of in-situ bioremediation potential and the application of bioventing at a fuel-contaminated site, in *In situ Bioreclamation*, (R.E. Hinchee and R.F. O'fenbuttel, eds.), Butterworth-Heinemann, Boston, 262-282, 1991.
- Dupont, R.R., Fundamentals of bioventing applied to fuel contaminated sites, *Environ. Progress*, 12(1), 45-53, 1993.
- Edwards, K.B. and L.C. Jones, Air permeability from pneumatic tests in oxidized till, *J. Environ. Eng.*, 120(2), 329-347, 1994.
- English, C.W. and R.C. Loehr, Degradation of organic vapors in unsaturated soils, *J. of Haz. Materials*, 28, 55-63, 1991.
- Falta, R.W., I. Javandel, K. Pruess and P.A. Witherspoon, Density-driven flow of gas in the unsaturated zone due to evaporation of volatile organic compounds, *Water Resour. Res.*, 25(10), 2159-70, 1989.
- Falta, R.W., K. Pruess, I. Javandel, and P.A. Witherspoon, Numerical modeling of steam injection for the removal of nonaqueous phase liquids from the subsurface 2. Code validation and application, *Water Resour. Res.*, 28(2), 451-465, 1992.
- Fan, S. and K.M. Scow, Biodegradation of trichloroethylene and toluene by indigenous microbial populations in soil, *Applied and Environmental Microbiology*, 59(6), 1911-1918, 1993.
- Fen, C.S., Mathematical Modeling and Analysis of Flux Mechanisms Controlling Multicomponent Vapor Transport in the Subsurface, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, 1993.
- Fine, P. and B. Yaron, Outdoor experiments on enhanced volatilization by venting of kerosene component from soil, *J. Contam. Hydrol.*, 12, 355-374, 1993.
- Freeze, R.A. and J.A. Cherry, *Groundwater*, Prentice-Hall, Englewood Cliffs, N.J., 1979.
- Frind, E.O., W.H.M. Duynisveld, O. Strelbel and J. Boettcher, Modeling of multicomponent transport with microbial transformation in groundwater: The Fuhrberg case, *Water Resour. Res.*, 26(8), 1707-19, 1990.
- Fuller, M.E., D.Y. Mu, and K.M. Scow, Biodegradation of trichloroethylene and toluene by indigenous microbial populations in vadose sediments, *Microb. Ecol.*, 29, 311-325, 1995.
- Gannon, K. and D.J. Wilson, Soil clean up by *in-situ* aeration, II. Effects of impermeable caps, soil permeability, and evaporative cooling, *Separation Sci. and Tech.*, 24(11), 831-862, 1989.
- Gerbasi, P.J., and M.A. Memoli, The value of vapor extraction, *Water Envir. & Tech.*, 6(3), 40-6, 1994.
- Gibson, T.L., A.S. Abdul *et al.*, Vapor extraction of volatile organic compounds from clay soil: A long term field pilot study, *Ground Water*, 31(4), 616-626, 1993.
- Gierke, J.S., N.J. Hutzler, and J.C. Crittenden, Modeling the movement of volatile organic chemicals in columns of unsaturated soil, *Water Resour. Res.*, 26(7), 1529-47, 1990.
- Gierke, J.S., N.J. Hutzler and D.B. McKenzie, Vapor transport in unsaturated soil columns: Implications for vapor extraction, *Water Resour. Res.*, 28(2), 323-335, 1992.

- Gomez-Lahoz, C., J.M. Rodriguez-Maroto and D.J. Wilson, Soil clean up by *in-situ* aeration: VI. effects of variable permeabilities, *Separation Sci. and Tech.*, 26(2), 133-163, 1991.
- Goss, K-U., Effects of temperature and relative humidity on the sorption of organic vapors on quartz sand, *Environ. Sci. Technol.*, 26(11), 2287-2293, 1992.
- Haley, J.L., B. Hanson, C. Enfield and J. Glass, Evaluating the effectiveness of groundwater extraction systems, *Groundwater Monitoring Rev.*, 11(1), 119-24, 1991.
- Hayden, N.J., T.C. Voice, M.D. Annable and R.B. Wallace, Change in gasoline constituent mass transfer during soil venting, *J. of Environ. Engr.*, 120(6), 1598-1614, 1994.
- Heid, J.G., J.J. McMahon, R.F. Nielsen and S.T. Yuster, Study of the permeability of rocks to homogeneous fluids, 230-246, American Petroleum Institute, New York, 1950.
- Hinchee, R.E. and M. Arthur, Bench scale studies of the soil aeration process for bioremediation of petroleum hydrocarbons, *J. Appl. Biochem. & Biotech.*, 28/29, 901-906, 1991.
- Hinchee, R.E., D.C. Downey, R.R. Dupont, P. Aggarwal and P. Miller, Enhancing biodegradation of petroleum hydrocarbons through soil venting, *J. of Hazardous Materials*, 27, 315-325, 1991.
- Hinchee, R.E., Bioventing of Petroleum Hydrocarbons, in *Handbook of Bioremediation*, 39-59, Lewis Pub., Boca Raton, 1994.
- Ho, C.K. and K.S. Udell, An experimental investigation of air venting of volatile liquid hydrocarbon mixtures from homogeneous and heterogeneous porous media, *J. Contam. Hydrol.*, 11, 291-216, 1992.
- Ho, C.K., S-W. Liu and K.S. Udell, Propagation of evaporation and condensation fronts during multicomponent soil vapor extraction, *J. Contam. Hydrol.*, 16, 381-401, 1994.
- Hoag, G.E., C.J. Bruell, and M.C. Marley, A Study of the Mechanisms Controlling Gasoline Hydrocarbon Partitioning and Transport in Groundwater Systems, NTIS publication No. PB85-242907, October 1984.
- Hoag, G.E. and M.C. Marley, Gasoline residual saturation in unsaturated uniform aquifer materials, *J. of Environ. Engr.*, 112(3), 586-604, 1986.
- Hoffman, G.D., L.M. Abriola and K.D. Pennell, An investigation of interphase mass transfer NAPL dissolution processes, *EOS*, Trans. American Geophysical Union, 73(43), 127, 1993.
- Holman, H-Y. and Y.W. Tsang, Effects of soil moisture on biodegradation of petroleum hydrocarbons, in *In Situ Aeration: Air Sparging, Bioventing, and Related Remediation Processes* (R.E. Hinchee et al. eds.), 323-332, Battelle Press, Columbus, 1995.
- Huesemann, M.H. and K.O. Moore, The effects of soil type, crude oil type and loading, oxygen, and commercial bacteria on crude oil bioremediation kinetics as measured by soil respirometry, in *Hydrocarbon Bioremediation*, 58-71, Lewis Pub., Boca Raton, 1994.
- Hutzler, N.F., B.E. Murphy and J.S. Gierke, State of Technology Review of Soil Vapor Extraction Systems, Cooperative Agreement CR-814319-01-1, Hazardous Waste Engineering Research Laboratory, Environmental Protection Agency, Cincinnati, 1989.
- Huyakorn, P.S. and G.F. Pinder, *Computational Methods in Subsurface Flow*, Academic, San Diego, 1983.
- Johnson, P.C., M.W. Kemblowski and J.D. Colthart, Quantitative analysis for the cleanup of hydrocarbon-contaminated soils by *in-situ* soil venting, *Ground Water*, 28(3), 403-412, 1990.
- Johnson, P.C., A.L. Baehr, R.A. Brown, R.E. Hinchee, and G.E. Hoag, *Innovative Site Remediation Technology: Vol. 8 - Vacuum Vapor Extraction*, American Academy of Environmental Engineers, Annapolis, 1995.
- Joss, C.J., A three-dimensional, Multispecies vapor transport model for the design of vapor extraction and bioventing systems, *Proc. Petroleum Hydrocarbons and Organic Chemicals in Ground Water*, 493-507, National Water Well Assoc., 1993.
- Kaluarachchi, J.J. and J.C. Parker, Multiphase flow with a simplified model for oil entrapment, *Trans. in Porous Media*, 7, 1-14, 1992.
- Kampbell, D.H. and J.T. Wilson, Bioventing to treat fuel spills from underground storage tanks, *J. Haz. Materials*, 28, 75-80, 1991.

- Kayano, S. and D.J. Wilson Soil clean up by *in situ* aeration. X. Vapor stripping of mixtures of volatile organics obeying Raoult's Law, *Separation Sci. and Tech.*, 27(12), 1525-1554, 1992.
- Kearl, P.M., N.E. Korte, T.A. Gleason and J.S. Beale, Vapor extraction experiments with laboratory soil columns: Implications for field programs, *Waste Management*, 11, 231-239, 1991.
- Kellems, B.L., A. Leeson, and R.E. Hinchee, Review of bioremediation experience in Alaska, in *Hydrocarbon Bioremediation*, 438-443, Lewis Pub., Boca Raton, 1994.
- Kinzelbach, W., W. Schafer and J. Herzer, Numerical modeling of nitrate and enhanced denitrification processes in aquifers, *Water Resour. Res.*, 27(6), 1123-1135, 1991.
- Klinkenberg, L.J., *The Permeability of Porous Media to Liquids and Gases in Drilling and Production Practice*, American Petroleum Institute, New York, 1941.
- Lapidus, L. and G.F. Pinder, *Numerical Solution of Partial Differential Equations in Science and Engineering*, John Wiley and Sons, New York, 1982.
- Leeson, A., P. Kumar, et al., Statistical analyses of the U.S. Air Force bioventing initiative results, in *In Situ Aeration: Air Sparging, Bioventing, and Related Remediation Processes* (R.E. Hinchee et al. eds.), 223-235, Battelle Press, Columbus, 1995.
- Lingineni, S. and Dhir, V.K., Modeling of soil venting processes to remediate unsaturated soils, *J. of Environmental Engineering*, 118(1), 135-152, 1992.
- Litchfield, C.D., *In situ bioremediation: Basis and practices*, in *Biotreatment of Industrial and Hazardous Waste*, 167-195, McGraw-Hill, New York, 1993.
- Lyman, W.J., W.F. Rheel and D.H. Rosenblatt (Eds.), *Handbook of Chemical Property Estimation Methods: Environmental Behavior of Organic Compounds*, McGraw-Hill Book Co., New York, 1982.
- Mackay, D.M. and J.A. Cherry, Groundwater contamination: Pump-and-treat remediation, *Environ. Sci. Technol.*, 23(6), 632-36, 1989.
- Marley, M.C. and G.E. Hoag, Induced soil venting for recovery/restoration of gasoline hydrocarbons in the vadose zone, in *Proc., Petroleum Hydrocarbons and Organic Chemicals in Ground Water - Prevention, Detection and Restoration*, Houston, Texas, 473-503, National Water Well Association, 1984.
- Massmann, J.W., Applying groundwater flow models in vapor extraction system design, *J. Environ. Eng.*, 115(1), 129-149, 1989.
- Massmann, J.W. and D.F. Farrier, Effects of atmospheric pressures on gas transport in the vadose zone, *Water Resour. Res.*, 28(3), 777-791, 1992.
- Massmann, J.W. and M. Madden, Estimating air conductivity and porosity from vadose-zone pumping tests, *J. Environ. Eng.*, 120(2), 313-328, 1994.
- McBride, J.F., C.S. Simmons and J.W. Cary, Interfacial spreading effects on one-dimensional organic liquid imbibition in water-wetted porous media, *J. of Contaminant Hydrol.*, 11, 1-25, 1992.
- McCann, M., P. Boersma, J. Danko, and M. Guerriero, Remediation of a VOC contaminated Superfund site using soil vapor extraction, groundwater extraction, and treatment: A case study, *Environ. Progress*, 13(3), 208-213, 1994.
- McClellan, R.D. and R.W. Gilham, Vapour extraction of trichloroethylene under controlled conditions at the Borden site, in *Subsurface Contamination by Immiscible Fluids*, 89-96, A.A. Balkema, Rotterdam, 1992.
- McWhorter, D.B., Unsteady radial flow of gas in the vadose zone, *J. Contam. Hydrol.*, 5, 297-314, 1990.
- Mendoza, C.A. and E.O. Frind, Advective-dispersive transport of dense organic vapors in the unsaturated one 1. Model development, *Water Resour. Res.*, 26(3), 379-387, 1990.
- Mercer, J.W. and R.M. Cohen, A review of immiscible fluids in the subsurface: properties, models, characterization and remediation, *J. of Contaminant Hydrology*, 6, 107-163, 1990.
- Metcalf, D.E. and G.J. Farquhar, Modeling gas migration through unsaturated soil from waste disposal sites, *Water Air & Soil Pollution*, 32, 247-259, 1987.

- Miller, R.N., A Field Scale Investigation of Enhanced Petroleum Hydrocarbon Biodegradation in the Vadose Zone Combining Soil Venting as an Oxygen Source with Moisture and Nutrient Addition, Ph.D. Dissertation, Utah State Univ., Logan, UT, 1990.
- Miller, R.N., D.C. Downey, V.A. Carmen, R.E. Hinchee, and A. Leeson, A summary of bioventing performance at multiple air force sites, in *Hydrocarbon Bioremediation* (R.E. Hinchee et al. eds.), 397-411, Lewis Pub., Boca Raton, 1994.
- Millington, R.J. and J.P. Quirk, Permeability of porous solids, *Trans. of the Faraday Society*, 57, 1200-1207, 1961.
- Milly, P.C.D., A mass-conservative procedure for time-stepping in models of unsaturated flow, *Adv. Water Resources*, 8, 32-36, 1985.
- Molz, F.J., M.A. Widdowson and L.D. Benfield, Simulation of microbial growth dynamics coupled to nutrient and oxygen transport in porous media, *Water Resour. Res.*, 22, 1207-1216, 1986.
- Mohr, D.H. and P.H. Merz, Application of a 2D air flow model to soil vapor extraction and bioventing case studies, *Ground Water*, 33(3), 433-444, 1995.
- Moore, B.J., J.E. Armstrong, et al., The influence of temperature on bioventing, in *In Situ Aeration: Air Sparging, Bioventing, and Related Remediation Processes* (R.E. Hinchee et al. eds.), 333-340, Battelle Press, Columbus, 1995.
- Mu, D.Y. and K.M. Scow, Effect of trichloroethylene (TCE) and toluene concentrations on TCE and Toluene biodegradation and the population density of TCE and toluene degraders in soil, *Applied and Environmental Microbiology*, 60(7), 2661-2665, 1994.
- Mualem, Y., A new model for predicting the hydraulic conductivity of unsaturated porous media, *Water Resour. Res.*, 12(3), 513-522, 1976.
- Mueller, J.G., P.J. Chapman and E.H. Pritchard, Creosote-contaminated sites, *Environ. Sci. Technol.*, 23, 1197-1201, 1989.
- Munz, C. and P.V. Roberts, The ratio of gas-phase to liquid-phase mass transfer coefficients in gas-liquid contacting processes, in *Gas Transfer at Water Surfaces* (W. Brutsaert and G.H. Kirka eds.), D. Reidel, Dordrecht, 35-45, 1984.
- Mutch Jr., R.D. and D.J. Wilson, Soil clean up by *in-situ* aeration. IV. Anisotropic permeabilities, *Separation Sci. and Tech.*, 25(1 & 2), 1-29, 1990.
- National Research Council, *Alternatives for Ground Water Cleanup*, National Academy Press, Washington, D.C., 1994.
- Nelson, C.H., R.J. Hicks and S.D. Andrews, In situ bioremediation: An integrated system approach, in *Hydrocarbon Bioremediation*, 125-133, Lewis Pub., Boca Raton, 1994.
- Norris, R.D., K. Dowd and C. Maudlin, The use of multiple oxygen sources and nutrient delivery systems to effect in situ bioremediation of saturation and unsaturated soils, in *Hydrocarbon Bioremediation*, 405-410, Lewis Pub., Boca Raton, 1994.
- Novak, J.T., R.G. Young and S. Forsling, Bioavailability of contaminants sorbed to soil organic matter, Proc. Petroleum Hydrocarbons and Organic Chemicals in Ground Water, 335-349, National Ground Water Association, 1993.
- Ogata, A. and R.B. Banks, A solution of the differential equation of longitudinal dispersion in porous media, Geological Survey Professional Paper 411-A, Washington, D.C.: United States Government Printing Office, 1961.
- Ong, S.K., R.E. Hinchee, R. Hoeppe and R. Schultz, In situ respirometry for determining aerobic degradation rates, *In situ Bioremediation*, (R.E. Hinchee and R.F. O'fenbuttel, eds.), Butterworth-Heinemann, Boston, 541-45, 1991.
- Ong, S.K., A. Leeson, R.E. Hinchee, et al., Cold climate applications of bioventing, in *Hydrocarbon Bioremediation*, 444-453, Lewis Pub., Boca Raton, 1994.
- Osejo, R.E. and D.J. Wilson, Soil clean up by *in situ* aeration. IX. Diffusion constants of volatile organics and removal of underlying liquid, *Separation Sci. and Tech.*, 26(12), 1433-1466, 1991.

- Parker, J.C., R.J. Lenhard and T. Kuppasamy, A parametric model for constitutive properties governing multiphase flow in porous media, *Water Resour. Res.*, 23, 618-624, 1987.
- Pedersen, T.A. and J.T. Curtis, *Soil Vapor Extraction Technology - Reference Handbook*, U.S. EPA, EPA/540/2-91/003, 1991.
- Pennell, K.D., R.D. Rhue, P.S.C. Rao and C.T. Johnson, Vapor-phase sorption of p-xylene and water on soils and clay minerals, *Environ. Sci. Technol.*, 26(4), 756-762, 1992.
- Perry, R.H. and C.H. Chilton, *Chemical Engineers' Handbook*, McGraw-Hill, New York, 1973.
- Philip, J.R., Theory of infiltration, *Advances in Hydroscience*, (ed. V.T. Chow), 5, Academic Press, New York, 1969.
- Pignatello, J.J. and B. Xing, Mechanisms of slow sorption of organic chemicals to natural particles, *Environ. Sci. Technol.*, 30(1), 1-11, 1996.
- Powers, S.E., C.O. Loureiro, L.M. Abriola and W.J. Weber, Jr., Theoretical study of the non-equilibrium dissolution of NAPL in subsurface systems, *Water Resour. Res.*, 27(4), 463-77, 1991.
- Powers, S.E., L.M. Abriola and W.J. Weber, Jr., An experimental investigation of NAPL dissolution in saturated subsurface systems: Steady-state mass transfer rates, *Water Resour. Res.*, 28(10), 2691-2705, 1992.
- Powers, S.E., L.M. Abriola and W.J. Weber, Jr., An experimental investigation of NAPL dissolution in saturated subsurface systems: Transient mass transfer rates, *Water Resour. Res.*, 30(2), 321-332, 1994.
- Rainwater, K., M.R. Zaman, B.J. Claborn and H.W. Parker, Experimental and modeling studies of in situ volatilization: vapor-liquid equilibrium or diffusion-controlled process?, Proc. Petroleum Hydrocarbons and Organic Chemicals in Ground Water, 357-371, National Water Well Assoc., 1989.
- Rathfelder, K. W.W.-G. Yeh and D. Mackay, Mathematical simulation of soil vapor extraction systems: model development and numerical examples, *J. Contam. Hydrol.*, 8, 263-297, 1991.
- Rathfelder, K.M. and L.M. Abriola, Mass conservative numerical solutions of the head-based Richards equation, *Water Resour. Res.*, 30(9), 2579-2586, 1994.
- Rathfelder, K.M., J.R. Lang, and L.M. Abriola, Soil vapor extraction and bioventing: Applications, limitations, and future research directions, *Reviews of Geophysics IUGG Quadrennial Report*, American Geophysical Union, 1067-1082, 1995.
- Reeves, H.W. and L.M. Abriola, A decoupled approach to the simulation of flow and transport of non-aqueous organic phase contaminants through porous media, *Modeling Surface and Sub-surface Flows, Developments in Water Science*, Proceedings of the VII International Conference on Computational Methods in Water Resources, Vol. 1, 35, 147-52, Elsevier, 1988.
- Reeves, H.W., Volatilization and Vapor Phase Transport of Organic Contaminants in the Subsurface, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, 1993.
- Reeves, H.W. and L.M. Abriola, An iterative-compositional model for subsurface multiphase flow, submitted to *Journal of Contaminant Hydrology*, 15, 249-276, 1994.
- Reid, R.C., J.M. Prausnitz and T.K. Sherwood, *The Properties of Gases and Liquids*, McGraw-Hill, New York, 1977.
- Reisinger, H.J., E.F. Johnstone, and P. Hubbard, Jr., Cost effectiveness and feasibility comparison of bioventing vs. conventional soil venting, in *Hydrocarbon Bioremediation*, 40-57, Lewis Pub., Boca Raton, 1994.
- Riddick, J.A., W.B. Bunger, and T.K. Sakano, *Organic Solvents: Physical Properties and Methods of Purification*, John Wiley, New York, 1986.
- Rijnaarts, H.H.M., A. Bachmann, J.C. Jumelet and A.J. Zehnder, Effect of desorption and intraparticle mass transfer on the aerobic biomineralization of α -hexachlorocyclohexane in a contaminated calcareous soil, *Environ. Sci. Technol.*, 24, 1349-1354, 1990.
- Roberts, L.A. and D.J. Wilson, Soil clean up by *in-situ* aeration: XI. Cleanup time distributions for statistically equivalent variable permeabilities, *Separation Sci. and Tech.*, 28(8), 1539-1559, 1993.
- Roberts, P.V., G.D. Hopkins, C. Munz and A.H. Riojas, Evaluating two-resistance models for air stripping of volatile organic organic contaminants in a countercurrent packed column, *Environ. Sci. Technol.*, 19(2), 164-173, 1985.

- Rodriguez-Maroto, J.M. and D.J. Wilson, Soil clean up by *in-situ* aeration: VII. High-speed modeling of diffusion kinetics, *Separation Sci. and Tech.*, 26(6), 743-760, 1991.
- Rodriguez-Maroto, J.M., C. Gomez-Lahoz and D.J. Wilson, Soil clean up by *in-situ* aeration: VIII. Effects of system geometry on vapor extraction efficiency, *Separation Sci. and Tech.*, 26(8), 1051-1064, 1991.
- Roy, W.R. and R.A. Griffin, An analytical model for in situ extraction of organic vapors, *J. of Haz. Materials*, 26, 301-317, 1991.
- Sayles, G.D., R.E. Hinchee, C.M. Vogel, R.C. Brenner and R.N. Miller, An evaluation of concurrent bioventing of jet fuel and several soil warming methods: A field study at Eielson Air Force Base, Alaska, Abstracts from the Symposium on Bioremediation of Hazardous Wastes: Research, Development, and Field Evaluations, Dallas, TX, May 4-6, United States Environmental Protection Agency, EPA/600/R-93/054, 27-33, 1993.
- Sayles, G.D., A. Leeson, et al., Cold climate bioventing with soil warming in Alaska, in *In Situ Aeration: Air Sparging, Bioventing, and Related Remediation Processes* (R.E. Hinchee et al. eds.), 297-306, Battelle Press, Columbus, 1995.
- Schwarzenbach, R.P., P.M. Gschwend and D.M. Imboden, *Environmental Organic Chemistry*, John Wiley, New York, 1993.
- Schwille, F., Migration of organic fluids immiscible with water in the unsaturated zone, in *Pollutants in Porous Media*, (B. Yaron, G. Dagan and J. Goldshmid eds.), Springer-Verlag, Berlin, 1984.
- Scow, K.M. and M. Alexander, Effect of diffusion on the kinetics of biodegradation: Experimental results with synthetic aggregates, *Soil Sci. Soc. Am. J.*, 56, 128-134, 1992.
- Scow, K.M. and J. Hutson, Effect of diffusion and sorption the kinetics of biodegradation: Theoretical considerations, *Soil Sci. Soc. Am. J.*, 56, 119-127, 1992.
- Scow, K.M., Effect of sorption-desorption and diffusion processes on the kinetics of biodegradation of organic chemicals in soil, in *Sorption and Degradation of Pesticides and Organic Chemicals in Soil*, Special Pub. No. 32, Soil Science Society of America, 1993.
- Simunek, J., T. Vogel and M. Th. van Genuchten, The SWMS_2D Code for Simulating Water Flow and Solute Transport in Two-Dimensional Variably Saturated Media, Version 1.2, Research Report No. 132, U.S. Salinity Laboratory, Riverside, Calif., 1994.
- Sittler, S.P., G.L. Swinford and D.G. Gardner, Use of thermal-enhanced soil vapor extraction to accelerate remediation of diesel-affected soils, *Proc. Petroleum Hydrocarbons and Organic Chemicals in Ground Water*, 413-425, National Ground Water Association, 1993.
- Sleep, B.E. and J.F. Sykes, Modeling the transport of volatile organics in variably saturated media, *Water Resour. Res.*, 25(1), 81-92, 1989.
- Sleep, B.E. and J.F. Sykes, Biodegradation of volatile organic compounds in porous media with natural and forced gas-phase advection, *In situ Bioremediation*, (R.E. Hinchee and R.F. O'fenbuttel, eds.), Butterworth-Heinemann, Boston, 245-61, 1991.
- Speitel, G.E. and E.R. Alley, Bioremediation of unsaturated soils contaminated with chlorinated solvents, *J. Haz. Materials*, 28, 81-90, 1991.
- Stephanotos, B.N., Modeling the transport of gasoline vapors by an advective-diffusion unsaturated zone model. Proceedings, Petroleum Hydrocarbons and Organic Chemicals in Groundwater: Prevention, Detection and Restoration, National Water Well Association, Houston, TX, 591-611, 1988.
- Strang, G. and G.J. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1973.
- Sykes, J.F., S. Soyupak and G.J. Farquhar, Modeling of leachate organic migration and attenuation in ground waters below sanitary landfills, *Water Resour. Res.*, 18, 135-45, 1982.
- Texas Research Institute, Examination of Venting for the Removal of Gasoline Vapors from Contaminated Soil, American Petroleum Institute, Washington, D.C., 1980.
- Texas Research Institute, Forced Venting to Remove Gasoline Vapor from a Large-Scale Model Aquifer, American Petroleum Institute, Washington, D.C., 1984.

- Thomas, J.M. and C.H. Ward, Subsurface microbial ecology and bioremediation, *J. of Haz. Materials*, 32, 179-194, 1992.
- Thornton J.S. and W.L. Wootan, Venting for the removal of hydrocarbon vapors from gasoline contaminated soil, *J. Environ. Sci. Health*, A17(1), 31-44, 1982.
- Thorstenson, D.C. and D.W. Pollock, Gas transport in unsaturated porous media: The adequacy of Fick's Law, *Reviews of Geophysics*, 27(1), 61-78, 1989.
- Travis, C.C. and J.M. Macinnis, Vapor extraction of organics from subsurface soils — Is it effective?, *Environ. Sci. Technol.*, 26(10), 1885-1887, 1992.
- U.S. Environmental Protection Agency, *Innovative Treatment Technologies - Overview and Guide to Information Sources*, EPA/540/9-91/002, 1991.
- van Eyk, J., Venting and bioventing for the in situ removal of petroleum from soil, in *Hydrocarbon Bioremediation*, 243-251, Lewis Pub., Boca Raton, 1994.
- van Genuchten, M.T., A comparison of numerical solutions of the one-dimensional unsaturated-saturated flow and mass transport equations, *Adv. in Water Resources*, 5, 47-55, 1980.
- Voss, C.I., SUTRA, A finite element simulation model for saturated-unsaturated, fluid-density-dependent groundwater flow with energy transport or chemically-reactive single-species solute transport, U.S. Geological Survey, Reston, VA, 1984.
- Weber, W.J., Jr. and C.T. Miller, Modeling the sorption of hydrophobic contaminants by aquifer materials I. Rates and equilibria, *Water Res.*, 22(4), 457-464, 1988.
- Weber, W.J., Jr., P.M. McGinley and L.M. Katz, Sorption phenomena in subsurface systems: Concepts, models and effects on contaminant fate and transport, *Water Res.*, 25(5), 499-528, 1991.
- Weber, W.J., Jr., P.M. McGinley and L.M. Katz, A distributed reactivity model for sorption by soils and sediments 1. Conceptual basis and equilibrium assessments, *Environ. Sci. Technol.*, 26, 1955-1962, 1992.
- Weber, W.J., Jr. and F. DiGiano, *Process Dynamics in Environmental Systems*, John Wiley, New York, 1996.
- Welty, C., C.J. Joss, A.L. Baehr, and J.A. Dillow, Use of a three-dimensional air-flow model coupled with an optimization algorithm to aid in the design of soil venting systems, Proc. Petroleum Hydrocarbons and Organic Chemicals in Ground Water, 221-231, National Water Well Association, 1991.
- Welty, J.R., C.E. Wicks and R.W. Wilson, *Fundamentals of Momentum, Heat and Mass Transfer*, John Wiley, New York, 1984.
- Widdowson, M.A., F.J. Molz and L.D. Benfield, A numerical transport model for oxygen- and nitrate-based respiration linked to substrate and nutrient availability in porous media, *Water Resour. Res.*, 24, 1553-65, 1988.
- Wilkins, M.D., L.M. Abriola, and K.D. Pennell, An experimental investigation of rate limited nonaqueous phase liquid volatilization in unsaturated porous media: Steady state mass transfer, *Water Resour. Res.*, 31(9), 2159-2172, 1995.
- Williamson, K.J. and P.L. McCarty, A model of substrate utilization by bacterial films, *J. Water Pollut. Control Fed.*, 48(1), 9-24, 1976.
- Wilson, D.J., Soil clean up by *in situ* aeration: V. Vapor stripping from fractured bedrock, *Separation Sci. and Tech.*, 25(3), 243-262, 1990.
- Wilson, D.J. and A.N. Clarke, Removal of semivolatiles from soils by steam stripping 1. A local equilibrium model, *Separation Sci. and Tech.*, 27(11), 1337-1359, 1992.
- Wilson, J.L. and S.H. Conrad, E. Hagan, W.R. Mason, and W. Peplinski, The pore level spatial distribution and saturation of organic liquids in porous media, Proc. of NWWA Conference on Petroleum Hydrocarbons and Organic Chemicals in Groundwater, National Water Well Association, Dublin, OH, 107-132, 1988.
- Wilson, J.L., S.H. Conrad, W.R. Mason, W. Peplinski and E. Hagan, Laboratory Investigation of Residual Liquid Organics from Spills Leaks and the Disposal of Hazardous Wastes in Groundwater, Robert S. Kerr Environmental Research Laboratory, EPA/600/6-90/004, April 1990.

- Wilson, J.L., Pore scale behavior of spreading and non-spreading liquids in the vadose zone, in *Subsurface Contamination by Immiscible Fluids*, (K.U. Weyer ed.), 107-114, A.A. Balkema, Rotterdam, 1992.
- Wilson, J.L., Bioventing of chlorinated solvents for ground-water cleanup through bioremediation, in *Handbook of Bioremediation*, 117-129, Lewis Pub., Boca Raton, 1994.
- Yeh, G-T., On the computation of Darcian velocity and mass balance in the finite element modeling of groundwater flow, *Water Resour. Res.*, 17(5), 1529-1534, 1981.
- Zaidel, J. and D. Russo, Analytical models of steady state organic species transport in the vadose zone with kinetically controlled volatilization and dissolution, *Water Resour. Res.*, 29(10), 3343-3356, 1993.
- Zienkiewicz, O.C. and R.L. Taylor, *The Finite Element Method*, McGraw-Hill, London, 1991.
- Zwick, T.C., A. Leeson, et al., Soil moisture effects during bioventing in fuel contaminated arid soils, in *In Situ Aeration: Air Sparging, Bioventing, and Related Remediation Processes* (R.E. Hinchey et al. eds.), 333-340, Battelle Press, Columbus, 1995.

