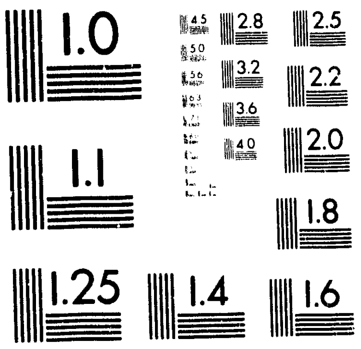
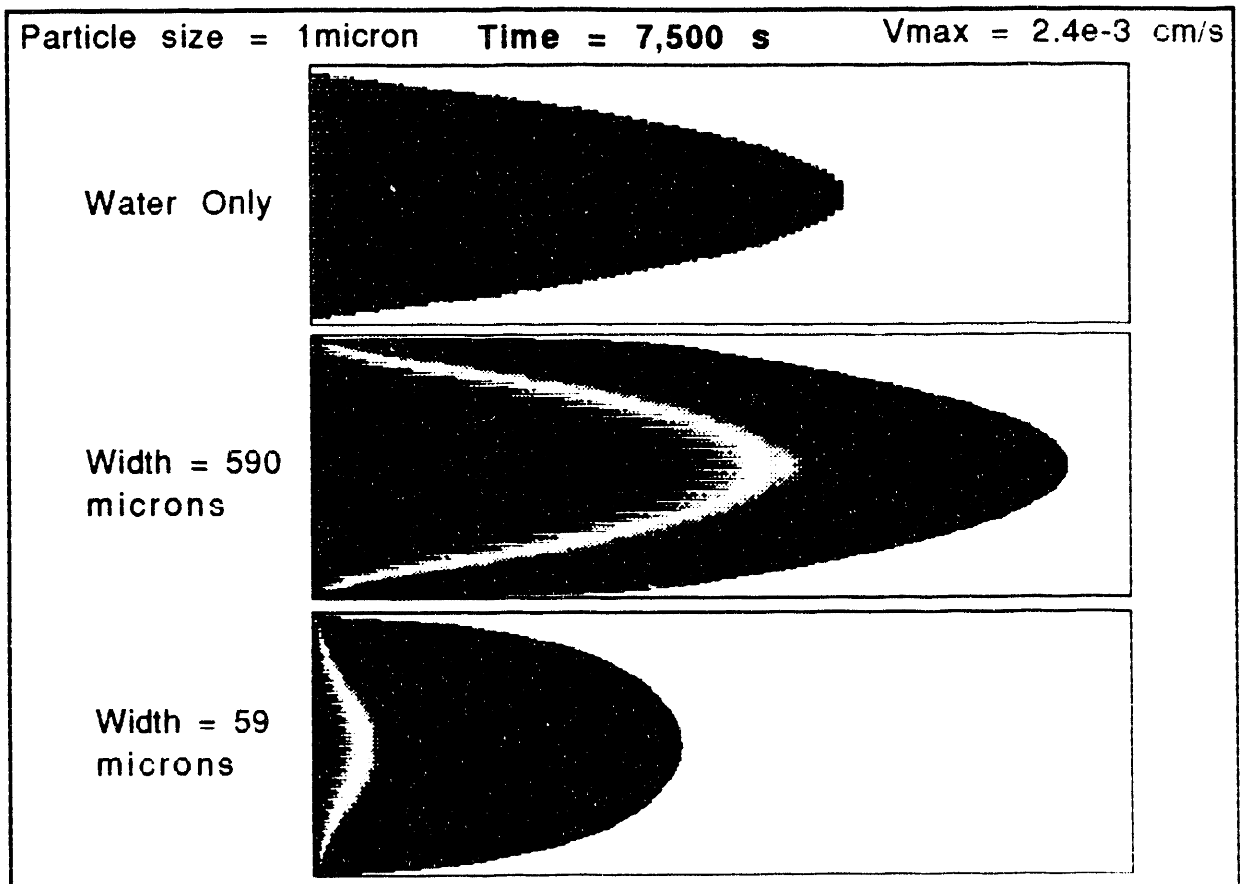

1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

1 of 1

CTCN : COLLOID TRANSPORT CODE - NUCLEAR

A User's Manual



Comparison of Colloid Transport in a Large and Small Fracture

by

Rohit Jain

CONTENTS

CTCN : COLLOID TRANSPORT CODE NUCLEAR

A User's Manual

by

Rohit Jain

ABSTRACT

This report describes the CTCN computer code, designed to solve the equations of transient colloidal transport of radionuclides in porous and fractured media. This Fortran 77 package solves systems of coupled nonlinear differential-algebraic equations with a wide range of boundary conditions. The package uses the Method of Lines technique with a special section which forms finite-difference discretizations in up to four spatial dimensions to automatically convert the system into a set of ordinary differential equations. The CTCN code then solves these equations using a robust, efficient ODE solver. Thus CTCN can be used to solve population balance equations along with the usual transport equations to model colloid transport processes or as a general problem solver to treat up to four-dimensional differential-algebraic systems.

----

1. MODEL DESCRIPTION

1.1 Nature and Purpose

The CTCN code is a baseline version of a comprehensive code for quantifying hydrological colloidal migration of radionuclides for the Yucca Mountain Project. It is designed to solve the unsteady population balance equations along with mass, energy and momenta conservation equations in up to four Cartesian axes. The code is designed to incorporate a wide range of boundary conditions and submodels within the main equations and thus can be used for many other colloidal transport problems.

Yucca Mountain is the site for a proposed HLW repository. Earlier reports indicate that colloidal transport plays a significant part in the overall migration of radionuclides through groundwater(Saltelli et al., 1984; Fried et al., 1976; Champ et al., 1982; Travis and Nuttall, 1985; Fried et al., 1975; Ho and Miller, 1986; Means et al., 1978; Champ et al., 1984; Gschwend and Reynolds, 1987;). The population balance model developed to analyze colloid transport requires numerical solution to the coupled, nonlinear system of resulting differential-algebraic equations. CTCN is designed to solve these equations. In anticipation of future alterations to the model, flexibility and robustness have been a major criteria for development of CTCN.

This manual describes the model, its numerical solution and includes a user's guide with several sample problems.

1.2 Mathematical Model

1.2.1 Governing Equations : The population balance permits complete treatment of the colloid problem including birth, growth, capture, and dissolution(Randolph and Larson, 1988). The idea is to establish a phase space consisting of the three spatial and one temporal coordinates (external coordinates) as well as each property to be tracked as a separate axis (internal coordinates).

The usual transport equations are modified to include the internal axes. The rate of change of the properties is assumed to be continuous thus leading to the definition of a velocity along each property axis corresponding to the growth term. Radioactive decay is treated by assigning a concentration property axis to each species to be tracked. It must be noted that the species will not be conserved along the internal axes and so we have to include 'death' and 'birth' terms into the equations.

The equations for CTCN are ,

(i) Component mass balance for each dissolved species 'i',

$$\frac{\partial c_i}{\partial t} + (\vec{v} \,.\, \nabla)\, c_i = D_i \nabla^2 c_i - \lambda_i c_i - S_i$$

where,

c_i = concentration of species 'i'.

λ_i = radioactive decay constant for species 'i'.

D_i = effective diffusivity.

S_i = sink term representing transfer of species between the colloids and the matrix.

t = time

v = velocity of the fluid

(ii) Population balance for each colloidal species 'i',

$$\frac{\partial \psi_{ti}}{\partial t} + \nabla \,.\, (\vec{v}\, \psi_{nk}) - D_b \nabla^2 \psi_{nk} + \sum_{j=1}^{m} \frac{\partial (v_i \psi_{ti})}{\partial \xi_j} + D_k - B_k = 0$$

where,

ψ_{ti} = number density of colloid type 'i.'

ξ_j = property axis for property type 'j'.

$v_i = \frac{d\,\xi_j}{d\,t}$ = property growth rate

D_b = Brownian diffusion coefficient.

B_k = colloid birth function (due to nucleation, agglomeration etc.).

D_k = colloid death function (due to agglomeration, dissolution etc.).

(iii) Fluid velocity:

When there is flow through porous media we can have the saturated flow regime or the unsaturated flow regime. We may also view the problem with a macroscopic or a microscopic viewpoint. Eventually all these cases will be incorporated as user specified subroutines. We presently use the Effective Medium Theory (Sharma and Yortsos, 1987).

$$q = \frac{k\,\Delta p}{\eta_f L}$$

where,

q = superficial fluid velocity.
k = permeability.
Δp = macroscopic pressure drop .
η_f = fluid viscosity .
L = macroscopic length.

$$u_r = \frac{q\, r_p^2}{8\,k}\left(1 + \frac{r_m^4 - r_p^4}{r_p^4 + \left(\frac{Z}{2} - 1\right) r_m^4} \right)$$

where,

u_r = mean fluid velocity in pores of radius 'r'.
Z = coordination number.
r_p = pore radius.
r_m = mean pore radius.

We also incorporate the species and overall mass balance as well as the energy balance in the usual forms.

1.2.2. Submodels : The model requires submodels for the following phenomena:

Birth (heterogeneous and homogeneous nucleation) rate
Death rate
Growth Rate
Agglomeration rate
De-agglomeration rate
Adsorption/desorption of radionuclides
Decay of Nuclei within or on a colloid

Randolph and Larson give birth mechanisms. H. van Olphen points out specific mechanisms for clay colloids. Jantzen and Bibler proposed that iron and glass waste canisters react under geological conditions to produce iron silicate colloids.

Capture models have been given by McDowell-Boyer et al. for three types of removal: filter caking, straining, and physico-chemical removal (Brownian motion, inertia, interception, hydrodynamic and sedimentation).

Agglomeration is treated by the birth and death rate terms in the population balance. Smoluchowski first developed a diffusion limited agglomeration model for monodispersed particles. Hidy and Brock reviewed his work and extensions by Swift and Friedlander to polydisperse size distributions of agglomerating particles.

Recent work by Hurd included charge effects in a Witten-Sander model and showed that the resulting fractal-like particle has a fractal dimension of two but the number of calculations required make this model useless for now.

Since these submodels are in various stages of development at LANL, the code has been designed to incorporate them easily in later versions.

1.3. Numerical Model

In spite of the great importance of these equations to particulate system dynamics, solutions have been difficult to obtain. Analytical solutions are available only for a few simple forms of the initial conditions and submodel formulations. Numerical solutions have been reported but the optimum method for solving the equations has not been determined.One promising class of methods for solving the resulting equations is collocation on finite elements (Gelbard and Seinfeld, 1978). It is a widely used method but there is considerable doubt and a large degree of arbitrariness in the placement of collocation points. Traditional collocation methods are not accurate since, due to large variations in magnitude of the size variable, a single high-order polynomial fit is not feasible.

Marchai et al.(1988) present an effective argument for the use of finite-difference techniques in the solution of the population balance equations. They note that the methods using decomposition of the dependent variable(s) on a set of orthogonal functions suffer from a number of disadvantages. For example, the choice of the set of orthogonal functions is arbitrary, and the ways to achieve the mathematical transformation are different from one problem to another. Despite successes with these techniques, a degree of arbitrariness always remains. Moreover their result is the PSD as a continuous function whereas experimental data using particle counters is a histogram because of the inherent lumping of the PSD in measurement. The theoretical message is here stronger than the experimental measurements (while the moments transformations approach yields less information than the experimental data); the model is too precise for [illegible] computational adjustment of kinetic parameters.

Thus we use finite-differencing techniques to solve the resulting equatio[illegible] Due to uncertainties in the submodels at this point, the code has been design[illegible] handle a general system of coupled, nonlinear PDEs and ODEs. The Meth[illegible] Lines is a handy, reliable tool to numerically approximate the solution of sys[illegible] of differential-algebraic equations. The PDEs are converted into sets of co[illegible] first order ODEs by discretizing all but the temporal axis. Thus we get an OD[illegible] each spatial point. These ODEs are then solved using a robust, adaptive [illegible] solver. This idea was first applied by E. Rothe in 1930 (Liskovets, 19[illegible] equations of parabolic type and indeed is a 'natural' method for parabolic PD[illegible] has been seen to be a very robust method for a variety of problems, notab[illegible] 'driven cavity' problem using the streamline-vorticity formulation of the N[illegible] Stokes equations where it outperformed the ADI method (Melgaard and Si[illegible] 1981). Most general PDE software uses the method of lines (Melgaar[illegible] Sincovec, 1981; Scheisser, 1971; Sincovec and Madsen, 1974; Loeb, 1974; [illegible] 1973; Hyman, 1976). Until recently, most of these packages addressed prob[illegible] one or two spatial dimensions. However, recent advances in solving large [illegible] of ODEs accurately and economically allow solution of large systems (upto four dimensions) feasible.

If a survey is made of general purpose solvers for differential systems(Machura and Sweet, 1980), it is seen that despite the emergence of many new methods for the solution of such systems, the method of choice continues to be the method of lines. It has been used in a variety of codes(put all MOL code ref.s). The user implements this method by discretizing the spatial derivatives etc. with a choice from an array of finite difference methods available in the code and choosing from two general purpose ODE solvers.

To accommodate many yet to be defined submodels, constitutive relationships, and boundary conditions, the CTCN code is designed to be a general differential-algebraic solver. It is capable of solving equations of the form,

$$\frac{\partial u_i}{\partial t} = g\left(u_i, x, y, z, r, t, \frac{\partial u_i}{\partial x}, \frac{\partial u_i}{\partial y}, \frac{\partial u_i}{\partial z}, \frac{\partial u_i}{\partial r}, \frac{\partial^2 u_i}{\partial x^2}, \frac{\partial^2 u_i}{\partial y^2}, \frac{\partial^2 u_i}{\partial z^2}, \frac{\partial^2 u_i}{\partial r^2}, \frac{\partial (xf)_i}{\partial x}, \frac{\partial (yf)_i}{\partial y}, \frac{\partial (zf)_i}{\partial z}, \frac{\partial (rf)_i}{\partial r} \right)$$

Here, the subscript ' i ' indicates the different dependent variables, x, y, z, and r are the four independent spatial axes and xf, yf, zf, and rf are terms comprising of flux expressions for which it is sometimes desirable to difference them as a whole expression. These flux variables can also be differenced directionally, i.e. forward or backward differencing depending on the direction of the characteristics oriented into the axes. Thus we have four such terms, one corresponding to each of the four independent axes, along which its first derivative is to be taken. These terms themselves can be functionals of the form,

$$xf(i) = g\left(u_i, x, y, z, r, t, \frac{\partial u_i}{\partial x}, \frac{\partial u_i}{\partial y}, \frac{\partial u_i}{\partial z}, \frac{\partial u_i}{\partial r}, \frac{\partial^2 u_i}{\partial x^2}, \frac{\partial^2 u_i}{\partial y^2}, \frac{\partial^2 u_i}{\partial z^2}, \frac{\partial^2 u_i}{\partial r^2} \right)$$

and similarly for yf(i), zf(i) and rf(i).

Apart from the equations, we need grid specifications and boundary and initial conditions. The equations are solved on a finite, Cartesian, equispaced, orthogonal grid and with specific initial conditions specified by the user. Initial conditions need not be consistent with the boundary conditions.

A wide variety of boundary conditions can be incorporated into the problem domain. In general, the following types of boundary conditions are used,

$$a_i u_i + b_i \frac{\partial u_i}{\partial x} = c_i \qquad \text{at the x - boundaries}$$

and similarly for the other boundaries. CTCN incorporates boundary conditions by using fictitious points and cubic interpolation. It is sometimes necessary to impose a boundary condition that should actually be enforced at infinity but is enforced numerically at a large distance from the last meaningful grid point. CTCN obviates the use of such artificial and error-prone procedures. This kind of condition is handled by setting all coefficients (a, b and c) for that boundary condition to be set to zero. The code then interpolates the values of the dependent variables as required into the fictitious points.

The code offers many options for the type of finite-difference spatial discretizations to be performed. It incorporates these into the equations along with the boundary conditions to form a set of ordinary differential equations. If there are NPDE dependent variables on (NX * NY * NZ * NR) grid points in the four axes respectively, then we get (NPDE * NX * NY * NZ * NR) ODEs. As can be seen, the design of the code is geared towards flexibility and generality. This version of CTCN should be seen as a research version designed to be able to incorporate submodels, equations and boundary conditions under development and also to assist the development of these very submodels. Once these parameters have been set, the code can be 'fixed' in the sense that the user would only have to input a data file giving values for the parameters entering the equations etc. There would be no need for user-specified subroutines and consequently CTCN could be set up so that only the output and input data files would be visible to the user.

The ODEs resulting from any MOL(method of lines) discretization are stiff, i.e. components of the solution have time constants that vary in orders of magnitudes. This inherent stiffness is compounded by stiff features within the original system of equations (e.g. reactions with drastically different rate constants, diffusion etc.). There has been a lot of work on solving huge systems of stiff ODEs largely motivated by the generality and effectiveness of the MOL approach. Hindmarsh consolidated these efforts into his set of codes ODEPACK at LLNL. Of these codes, experience has shown the code LSODES to be the most effective package for the problems we consider. However, LSODES is not efficient for large problems (> 20-40,000 ODEs). Recent research by Hindmarsh and Brown resulted in the development of an experimental solver LSODPK which has so far proved to be more effective than LSODES.

The code is designed to accommodate either solver with minimal modifications.

2.0 CODE DESCRIPTION

2.1 Files and Operating Parameters

CTCN is targeted for use on Sun 4.0 and VAX computers running UNIX or VMS operating systems. Efforts have been made to make the code as machine-independent as possible. CTCN is written in Fortran77 and can be used on any computer running UNIX or VMS without any modifications except possibly the cpu-timing calls. A working knowledge of Fortran77 is necessary to use the code. For two-dimensional problems, a post-processing option is available to create ASCII input files for the NCSA Imagetool graphics package.

The package consists of the following files and subroutines:

Files	Routines
makefile	
inc.for	include file
dims.h	include file
user.f	**main** program, subroutines **eqn**, **func** and **bound**.
face.f	subroutine **face**.

setup.f	subroutines **bset, bcal, fder, dercal** and **set**.
lsodes.f	ODE solver LSODES.
lsodpk.f	ODE solver LSODPK.

These files have counterparts for VMS systems.

2.2 Subroutines and their description

See fig. 1 for a structural overview of CTCN. A list of the tasks performed by the various subroutines follows.

1. main program	sets up choice of options for CTCN and the ODE solver. Defines the grid points and initial conditions.
2. subroutine eqn	defines the equations for the dependent variables.
3. subroutine func	defines the flux terms.
4. subroutine bound	defines the boundary conditions.
5. subroutine face	incorporates boundary conditions into the initial data. Calls the ODE solver for the different output times, times the code, prints data for CTCN and some of the ODE solver's outputs, allocates array sizes for the ODE solver and may be used to change some of the other optional inputs to the ODE solver.
6. subroutine set	calculates the right hand side of the ODEs for the ODE solver.
7. subroutine dercal	evaluates the spatial derivatives of the dependent variables.
8. subroutine fder	evaluates the first derivatives of the flux terms.
9. subroutine bset	evaluates the values of the dependent variables at the fictitious points to incorporate the boundary conditions into the discretization.
10. subroutine bcal	calculates the boundary values.
11. subroutine lsodes (and lsodpk)	driver routine for the ODE solver package LSODES (and the package LSODPK).

2.3 Step-wise Operation

For any particular problem the user has to modify files 'user.f', 'inc.for' and possibly 'face.f' according to the step-wise procedure detailed below.

Step 1 : Enter the dimensions of the system in file 'inc.for'. This is a one-line file of the form,

parameter (npde=1, nx= 101, ny=1, nz=1,nr=1).

Set the unused axes to have at least one grid point and that for a one-dimensional problem, the x-axis has to be the independent axis for correct results. For two-dimensional problems, the x and y axes have to be used (and not the x and z axes or the y and z axes etc.) and so on. Also, once the axes are chosen, they must be set so that the number of gridpoints is in the order NX $\leq$ NY $\leq$ NZ $\leq$ NR. This does not include the dummy axes and is essential for maximum efficiency when using the LSODES solver.

Step 2 : Modify the MAIN program in file 'user.f'. Set the following variables (see example problems),

(i) output times - tout, tint, tlast, nout.

(ii) ODE solver options - itol, rtol, atol, mf.

(iii) Evaluation flags for the derivatives - meth.

(iv) Grid points and initial conditions - x, y, z, r and u.

Step 3 : Modify subroutine EQN in file 'user.f'. Enter the equations for the system using either the one grid point form or the one call form.

Step 4 : Modify subroutine FUNC in file 'user.f'. Enter the flux terms, if any, for the system using either the one grid point form or the one call form.

Step 5 : Modify subroutine BOUND in file 'user.f' so as to reflect the boundary conditions for the system.

Step 6 : Modify the block data module BDAT in file 'user.f'. Here, one only needs to make the dimensions of variable meth in the main program consistent with the number of zero's entered in the data statement in BDAT.

Once these steps have been completed, the code is ready to use. It may be necessary to modify subroutine FACE in file 'face.f' if the cpu timing calls need to be modified for a particular computer or if the printing routine needs to be modified.

The operation of the code for UNIX machines is as follows (commands to be typed in are in italics) :

(1)The user modifies files 'inc.for' and 'user.f' according to his problem as detailed later.

(2)If the ODE solver needs to be changed or the printing and/or code timing calls need to be modified, the user may modify file 'face.f'.

(3)The user types in the *make* command and an executable file 'ctcx' is created.

(4)Typing in the filename for the executable *ctcx* begins execution. The output file 'ctcn.out' is created at the end of the run.

The operation of the code for VMS machines is as follows:

(1)The user modifies files 'inc.for' and 'user.for' according to his problem as detailed later.

(2)If the ODE solver needs to be changed or the printing and/or code timing calls need to be modified, the user may modify file 'face.for'.

(3)The command @*make* has to be executed once before using a third-party make. This executes the file 'make.com'. This is necessary only once after each fresh login since this file links with a script containing default rules and software for the make command.

(4)The user types in the *make* command and an executable file 'ctcn.exe' is created.

(5)Typing in the command *run ctcn* begins execution. The output file 'ctcn.out' is created at the end of the run.

3.0. EXAMPLE PROBLEMS

In this section a complete description of the code is given by showing its usage for solving two example problems. The first problem is a simple diffusion equation in one dimension while the second one is a two-dimensional fracture problem that uses the population balance approach in its most elementary form but is readily extensible to a complete model with aggregation, birth terms etc

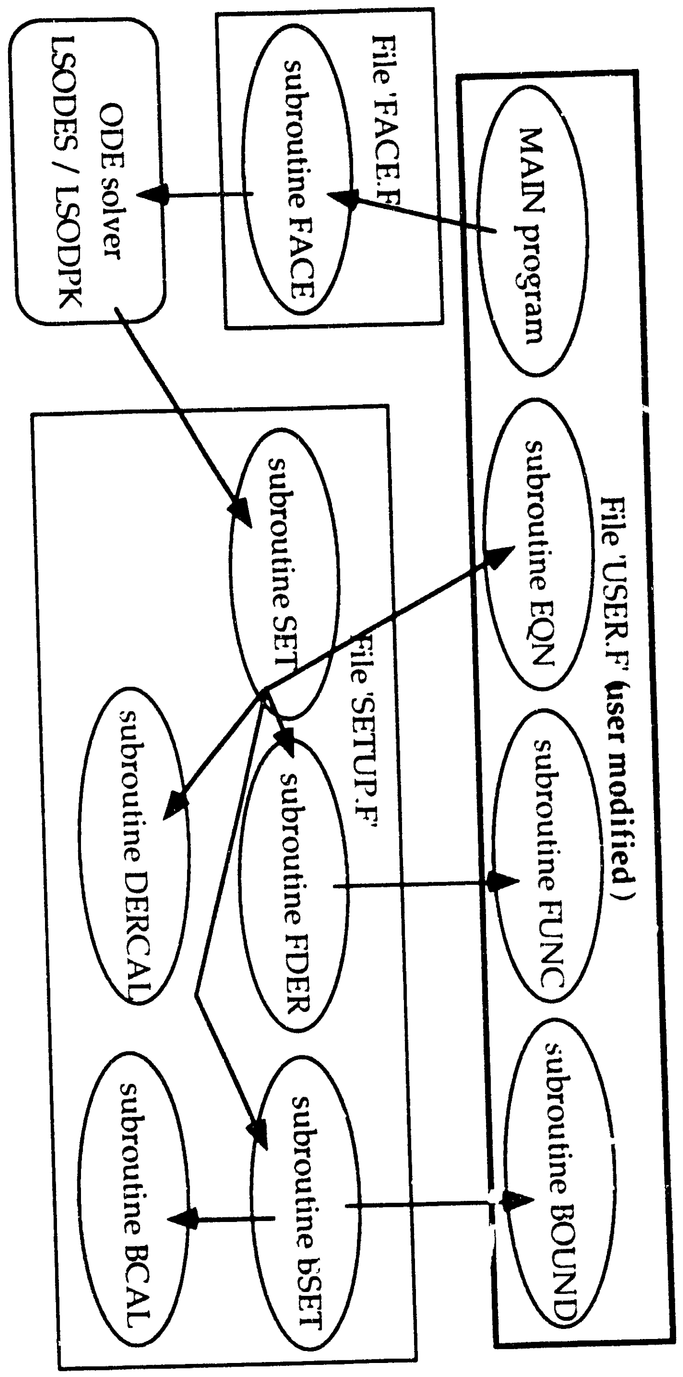


Fig. 1 : Structurai Overview of CTCN

3.1. Example 1 : Conduction in a Semi-infinite Medium : A semi-infinite medium, $x \geq 0$, is initially at zero temperature. For times $t > 0$, the boundary surface at $x = 0$ is subjected to a temperature T_0 (= 1.0). We wish to calculate the temperature distribution at any given time. The equation is ,

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

Initial Conditions : $u = 0$.
Boundary Conditions :
at $x = 0$, $u = 1.0$ and at x = infinity, $u = 0$.

We use a grid of 101 points distributed evenly in $0 \leq x \leq 1.0$ taking $\alpha = 0.005$ and seek the solution for $t = 5$.

The analytical solution is

$$T = T_0 \operatorname{erfc}\left(\frac{x}{\sqrt{4\alpha t}}\right)$$

Detailed below is the complete procedure for solving the first example equations.

Step 1 : Modify file 'inc.for' to reflect the dimensionality of the problem. We will solve this one-dimensional PDE on a grid of 101 points. In this case file 'inc.for' is written as,

parameter (npde = 1, nx = 101, ny = 1, nz = 1, nr = 1)

Note that we must set the unused axes to have at least one grid point and that for a one-dimensional problem, the x-axis has to be the independent axis for correct results.

Step 2: Modify the main program. The following variables are to be set,

(i) output times : Set the number and values of the output times where the solution is to be calculated. For our problem we want output at t = 5.0 s. The integration must begin at $t = 0$ and end at $t = 5.0$. The following variables are to be set,

1. dimension TOUT as TOUT(1) since we need output at just one time.
2. TINT - Integration starts at t = TINT and initial values are given at this time. Set TINT = 0.
3. TOUT(NOUT) - array of size NOUT. Contains the values for the output times. Set TOUT(1) = 5.0.
4. TLAST - Integration ends at t = TLAST. Set TLAST = 5.0.
5. NOUT - number of output times. Set NOUT = 1.

(ii) ODE solver options : Four parameters are to be set. Three are concerned with error tolerances for solving the ODEs. Relative and absolute error tolerances can be specified in scalar or vector form, i.e. for the whole system of equations or for any number of subsystems (e.g. for each dependent variable etc.). ITOL, RTOL and ATOL are these three parameters. If RTOL(relative error tolerance) and/or ATOL(absolute error tolerance) are vectors they must be dimensioned accordingly and ITOL(flag) must be set to two, otherwise it should be set to one. The tolerances may be set as vectors if one of the dependent variables differs significantly in magnitude with the others

and so would require a different level of accuracy during integration. A good rule for choosing the tolerances is that if the data requires precision of 10^{-n} then RTOL should be set to $10^{-(n+1)}$ and ATOL should be two orders of magnitude less than RTOL.
The fourth parameter is the ODE integration flag MF. Usually it should be set to 10 for a non-stiff problem and 22 for a stiff problem if LSODPK is to be used and 10 and 222 respectively if LSODES is to be used. For more details see the documentation for these solvers. The changes to be made are,

1. ITOL - Set ITOL = 1 (RTOL and ATOL are scalars).
2. RTOL - Set RTOL = 10^{-4} (need a precision of 10^{-3}).
3. ATOL - Set ATOL = 10^{-6}.
4. MF - Set MF = 22 (the problem is stiff and the solver is LSODPK).

(iii) Evaluation flags for the derivatives : For every dependent variable we must indicate which spatial and/or flux derivatives are to be calculated and by what methods. Since there are two spatial and one flux derivatives for each of the four axes, there are (2+1) x 4 = 12 flags for each dependent variable. Total number of flags = 12 * NPDE. Thus we allocate integers for a variable METH of dimensions (NPDE, 12). The flag corresponding to each term and the options for its value and their significance are given in the comments in the code.

1. METH - Looking at the equations we see that we need to calculate only the second derivative with respect to x for the one dependent variable. Thus we need only to set a value for
METH(1,11). We choose to use centered second order differencing for the derivatives. Thus, we set
meth(1, 11) = 2

(iv) Grid points and initial conditions : The grid points must be given values and the initial conditions for the system must be specified. Since our problem is one-dimensional we need to set DX, X(J) and U(1, J, 1,1,1) as follows,

```
c grid extends from 0 to 1.0. grid spacing dx = ***
   dx=1.0/(nx-1)
   do 20 j=1,nx
c  set values for x(j) and initial conditions in u(1,j,1,1,1)
     x(j)=(j-1)*dx
   u(1,j,1,1,1)=0.0
20  continue
```

Step 3 : Modify subroutine EQN for the problem so as to describe the right-hand side of the equations. The time derivatives of the dependent variables are to be defined in terms of the grid points (x, y, z, r), time t and the spatial (ux -> first derivative of u(1) with respect to x, uxx -> second derivative of u(1) with respect to x etc.) and flux derivatives (fx -> derivative of flux term xf(1) with respect to x) of the dependent variables. This subroutine may be written in two forms. In the first form the subroutine defines the equations at one spatial mesh point for each call

This form is the most applicable and easy for the user. Sometimes, however, the second form of the subroutine in which the user defines all the equations at all the grid points in one call to subroutine EQN, is more efficient. The second form may be used if the user needs to prescribe special discretizations, use unequal grid spacings, incorporate global integral relationships into the equations or to include ODEs as boundary conditions. This would be essential for any detailed population balance model since terms like the birth of particles into a size region would be given by an integral term involving particles with sizes less than this particular size range (aggregation). The choice of the second form is indicated to CTCN by setting IX > NPDE. We give both forms of the subroutine for our example problem,

Form 1 : One call defines equations at one grid point.

```
      subroutine eqn(ut,u,ux,uy,uz,ur,uxx,uyy,uzz,urr,fx,fy,fz,fr,t,x,y
     &           ,z,r,ix)
      integer ix,npde
       include 'inc.for'
       real ut(npde),u(npde),ux(npde),uy(npde),uz(npde),ur
     &           (npde),uxx(npde),uyy(npde),uzz(npde),urr(npde),
     &           fx(npde),fy(npde),fz(npde),fr(npde),t,x,y,z,r
c ***modify the lines below to reflect the equations to be solved****
      ut(1)=0.005*uxx(1)
c  **********no need to set ix***********
c *****end of problem specific part*****
       return
      end
```

Form 2 : One call defines equations at all grid points.

```
      subroutine eqn(ut,u,ux,uy,uz,ur,uxx,uyy,uzz,urr,fx,fy,fz,fr,t,x,y
     &           ,z,r,ix)
      integer ix,npde
       include 'inc.for'
c ******note changes in dimensioning for this form of the sub.****
       real ut(npde,nx,ny,nz,nr),u(npde,nx,ny,nz,nr)ux(npde,nx,ny,nz,nr
     &  ),uy(npde,nx,ny,nz,nr),uz(npde,nx,ny,nz,nr),ur(npde,nx,ny,
     &  nz,nr),uxx(npde,nx,ny,nz,nr),uyy(npde,nx,ny,nz,nr),uzz(
     &  npde,nx,ny,nz,nr),urr(npde,nx,ny,nz,nr),fx(npde,nx,ny,nz,nr)
     &  ,fy(npde,nx,ny,nz,nr),fz(npde,nx,ny,nz,nr),fr(npde,nx,ny,nz,
     &  nr),t,x(nx),y(ny),z(nz),r(nr)
c***end of changes in dimensioning for this form of the subroutine**
c ***modify the lines below to reflect the equations to be solved****
c ***note the do loop for this one dimensional problem involves
c  only the x subscript. All other subscripts are one.*****
       do 10 j = 1, nx
         ut(1,j,1,1,1)=0.005*uxx(1,j,1,1,1)
  10 continue
```

```
c***This form is indicated to CTCN by setting ix to be greater than
c   npde******
      ix = 4
c  *****end of problem specific part*****
      return
      end
```

Step 4 : When the equations involve the derivative of a flux function, that function should be defined in subroutine FUNC to retain the divergence form of the equations in the discretizations. These terms can also optionally be differenced by directional differencing. Subroutine FUNC defines four flux terms so as to provide the first derivative with respect to each of the axes. These terms are XF, YF, ZF and RF for the x, y, z and r axes respectively. The structure of the subroutine and the definition of the four terms are analogous with the structure of subroutine EQN and the definition of the time derivatives in it. Again, we may use subroutine FUNC in the same two forms as subroutine EQN except that then CTCN's directional differencing option may not be used to difference the flux terms. For this example we do not use a flux term.

```
      subroutine func(xf,yf,zf,rf,u,ux,uy,uz,ur,uxx,uyy,uzz,urr,t,x,y,z
     &            ,r,ix,npde)
      real xf(npde),yf(npde),zf(npde),rf(npde),u(npde),ux
     &             (npde),uy(npde),uz(npde),ur(npde),uxx(npde),uyy
     &             (npde),uzz(npde),urr(npde),t,x,y,z,r,v,p
      integer ix,npde
c *****modify this part for the specific problem******
c        no flux term
c *****end of problem-specific part********
      return
      end
```

Step 5 : When the boundary conditions are not defined in subroutine EQN using the second form (define the equations at all grid points in one call), the user must set them in subroutine BOUND. This is done by giving values to the variable B. As mentioned earlier, the general form of the boundary conditions is,

$$a_i\, u_i + b_i \frac{\partial u_i}{\partial x} = c_i$$

for the x-axis and analogously for the other axes. Thus the variable B is an array consisting of 3 x 2 x 4 x NPDE elements since there are four axes with two boundaries each and three coefficients (a, b and c) for each dependent variable (=NPDE dependent variables) at each boundary. Of these elements we define only the ones relevant to the problem, i.e. for a one-dimensional problem we define only those concerned with the x-boundaries. If the actual boundary condition is to be enforced at infinity, it is frequently more useful to not impose a boundary condition at an artificially large distance in the grid but to allow cubic interpolation at that boundary. This option is chosen by setting the coefficients (a

c) to zero. We make use of this option for our problem at the boundary x = 0.2. Thus for our problem the subroutine BOUND looks as follows,

```
    subroutine bound(u,b,x,y,z,r,t)
    integer npde
     include 'inc.for'
    real u(npde),b(npde,24),x,y,z,r,t
c This subroutine defines the boundary conditions - a*u + b*(du/dn) = c
c   du/dn is the outward normal derivative at the boundary
c    a,b,c are functions of x,y,z,r,t,u(i).
c The a,b and c values are entered in variable b in the following way:
c b(i,1) , b(i,2) and b(i,3) ---> a,b and c at x=x(1) for the i-th component
c b(i,4) , b(i,5) and b(i,6) ---> a,b and c at x=x(nx) for the i-th component
c b(i,7) , b(i,8) and b(i,9) ---> a,b and c at y=y(1) for the i-th component
c b(i,10) , b(i,11) and b(i,12) ---> a,b and c at y=y(ny) for the i-th component
c b(i,13) , b(i,14) and b(i,15) ---> a,b and c at z=z(1) for the i-th component
c b(i,16) , b(i,17) and b(i,18) ---> a,b and c at z=z(nz) for the i-th component
c b(i,19) , b(i,20) and b(i,21) ---> a,b and c at r=r(1) for the i-th component
c b(i,22) , b(i,23) and b(i,24) ---> a,b and c at r=r(nr) for the i-th component
c CAUTION: Enter all values of b. Default values are not zero!
c******begin problem-specific part. set values for b.********
c  at x=0,
c   u(1) = 1
     b(1,1)=1.0
     b(1,2)=0.0
     b(1,3)=1.0
c  at x=1.0,
c  At x = infinity, u  = 0.
     b(1,1)=0.0
     b(1,2)=0.0
     b(1,3)=0.0
c******end problem-specific part********
      return
      end
```

Step 6 : Modify the block data module BDAT. There is only one line that needs modification and that is the data statement for variable METH. It must be made dimensionally consistent with METH in the main program. As explained before, for this problem METH has dimension 12. Therefore our modification is,

DATA METH/12 * 0/

We now give a complete listing of the file USER.F for the first example problem. Comment lines are *italicized* . Actual Fortran code is in plain type and the problem-specific parts of this code are in **bold type**. Modifications are preceded by asterisks in the comment lines.

3.1.1. Code Listing for example 1

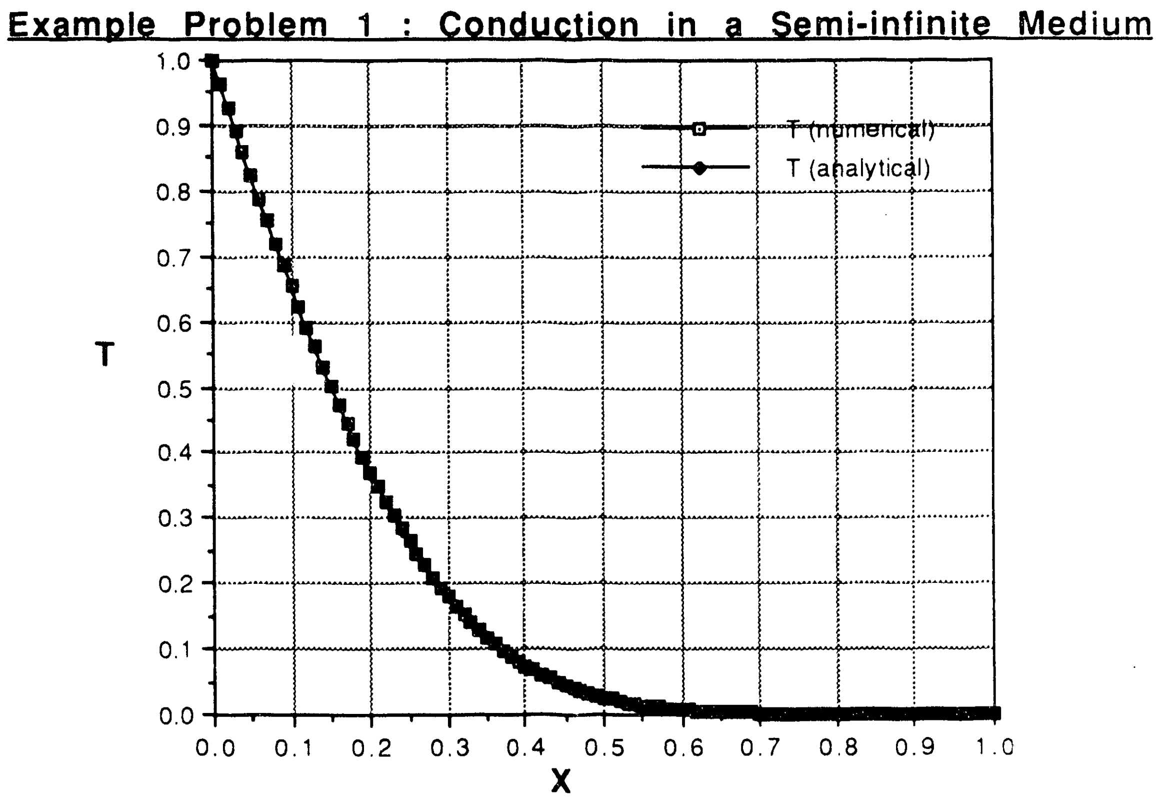
```
      program user
c This program interfaces with CTCN to solve systems of coupled, nonlinear
c PDEs of upto second order in upto four spatial axes and first order in
c time. Thus we solve for u(i,j,k,l,m) where,
c            i = 1 to npde (number of PDEs)
c            j = 1 to nx (number of points in the x axis)
c            k = 1 to ny (number of points in the y axis)
c            l = 1 to nz (number of points in the z axis)
c            m = 1 to nr (number of points in the r axis)
c  The problem should be set up so that the axes should be used in order x,y,z,r.
c  thus, a 1-dimensional problem MUST use the x-axis only.  A 2-D problem must
c  use the x and y as principal axes (not x and z or y and r or y and z etc.).
c  after this the number of points must be ordered as nx <= ny <= nz <= nr.
c  This is only necessary for using the LSODES solver efficiently.
c    Thus for a two dimensional 10 x 61 grid, nr=nz=1, nx=10 , ny=61.
c The basic problem parameters are defined in this main program. The eqn.s
c are written out in subroutine eqn, the boundary conditions in subroutine
c bound and flux terms in subroutine func. Boundary conditions may be of the
c general form, --> a*u + b*(du/dn) = c --> a,b and c are functions of the
c spatial coordinates,time and other u variables. du/dn is the derivative
c normal to the boundary. Boundary conditions need not be specified at all,
c in which case a cubic extrapolation technique is used.
c      For some problems it may be desirable to difference certain terms by
c skewed differences by considering the variation with u of the terms to be
c differenced. in this case, these terms may be entered using subroutine
c func and the appropriate differencing method selected. See subroutine
c func for further details.
c      Thus, to solve any given system, the user needs to modify this program
c and  include file INC.FOR.
c Include file 'inc.for' sets up the dimensions.
       integer npde,nx,ny,nz,nr,meth,idim
       include 'inc.for'
       double precision u(npde,nx,ny,nz,nr),tol,dx,dy,dz,dr,du,x,y,z,r
       common /c1/ tol,dx,dy,dz,dr,du
       common /c6/ x(nx),y(ny),z(nz),r(nr)
       common /c7/ idim,meth(npde,12)
c This is the user-defined program specific to a particular problem
c and sets up parameters for CTCN.
c      tout(i)---> array of output times. nout=i
c      tint---> initial value for time.
c      tlast---> final value of t. Integration is done upto tlast.
c      rtol---> relative tolerance for the ODE integrator.
c      atol---> absolute tolerance for the ODE integrator. Can be a vector
c               defined for each ODE, if so, set itol(below)=2.
c   *********this is where the  modifications begin***********
```

```
c    ***** modification**** dimension  tout*********
      double precision tout(1),tint,tlast,rtol,atol
c      nout---> no. of output times. Dimension of the tout array.
c     mf---> ODE integration method flag, the solver LSODPK is used.
c      itol---> 1 or 2 accordingly as atol is a scalar or a vector.
       integer nout,mf,itol
c Bdat is the block data subprogram at the end of this file.
      external bdat
c Integration starts from t=tint
c    ***modification***set  tint****
      tint=0.d0
c Enter array of times when output is required.
c****modification*****set   tout(nout)=******
        tout(1)=5.d0
c Integration stops at t=tlast.
c*****modification******set   tlast********
      tlast=5.d0
c The number of output times=array size for tout=nout
c  *****modification*****set   nout******
      nout=1
c Itol,rtol and atol are used to specify error tolerances for LSODPK-the ODE
c integrator.
c****modifications****set  itol,rtol  and  atol******
      itol=1
      rtol=1.d-4
      atol=1.d-6
c ode solvers require an ODE integration flag
c           for LSODPK solver           = mf = 22 for stiff problems, &
c                                  = 10 for non-stiff problems.
c           for LSODES solver           = mf = 222 for stiff problems, &
c                                  = 10 for non-stiff problems.
c*****modification*****set   mf********
      mf=22
c The type of differencing for each term is indicated by meth.
c     meth(i,1)= flag for ur(i), i.e. du/dr for the i-th pde.
c     meth(i,2)= flag for urr(i), i.e. d2u/dr2 for the i-th pde.
c     meth(i,3)= flag for fr(i), i.e. d(rf)/dr for the i-th pde.
c     meth(i,4)= flag for uz(i), i.e. du/dz for the i-th pde.
c     meth(i,5)= flag for uzz(i), i.e. d2u/dz2 for the i-th pde.
c     meth(i,6)= flag for fz(i), i.e. d(zf)/dz for the i-th pde.
c     meth(i,7)= flag for uy(i), i.e. du/dy for the i-th pde.
c     meth(i,8)= flag for uyy(i), i.e. d2u/dy2 for the i-th pde.
c     meth(i,9)= flag for fy(i), i.e. d(yf)/dy for the i-th pde.
c     meth(i,10)= flag for ux(i), i.e. du/dx for the i-th pde.
c     meth(i,11)= flag for uxx(i), i.e. d2u/dx2 for the i-th pde.
c     meth(i,12)= flag for fx(i), i.e. d(xf)/dx for the i-th pde.
```

```
c    if meth(*,*) = 2 ---> second order centered differences
c               = 4 ---> fourth order centered differences
c               = -2 ---> second order skewed differences
c                      ( only for flux terms written into rf,zf,yf,xf )
c               = -4 ---> third order skewed differences
c                      ( only for flux terms written into rf,zf,yf,xf )
c  Default value = 0
c******modification(s)*******set  meth  (  *,  *)  ********
        meth(1,11)=2
c Enter the axes grid sizes, default = 1.d0. dr=delta(r),...etc.
c********modification(s)******enter  grid  sizes****
     dx=1.d0/dble(nx-1)
c Enter grid point values.Input r(i)--> grid points on r axis....etc.
c*******modification(s)*****enter  grid  values*******
     do 20 j =1,nx
       x(j)=(j-1)*dx
c Enter initial conditions.  no default values.
c*****modifications*******enter  initial  conditions******
          u(1,j,1,1,1)=0.D0
 20  continue
c Call the interfacing subroutine CTCN
      call ctcn(u,rtol,atol,tint,tlast,tout,nout,mf,itol)
     stop
     end

      subroutine eqn(ut,u,ux,uy,uz,ur,uxx,uyy,uzz,urr,fx,fy,fz,fr,t,x,y
     &           ,z,r,ix)
c This subroutine is used to enter the PDEs
      integer ix,npde
      include 'inc.for'
c  *****optional  modifications*****change  dimension  statement  if  second  form
c     of subroutine EQN   is to be used*********
      double precision ut(npde),u(npde),ux(npde),uy(npde),uz(npde),ur
     &            (npde),uxx(npde),uyy(npde),uzz(npde),urr(npde),
     &            fx(npde),fy(npde),fz(npde),fr(npde),t,x,y,z,r
c Input the PDEs in the form ut(i)=f(u(i),x,y,z,r,t,ux(i),uy(i),uz(i),ur(i)
c    ,uxx(i),uyy(i),uzz(i),urr(i),fx(i),fy(i),fz(i),fr(i)
c****modifications******problem  specific  equations****
      ut(1)=5.d-3*uxx(1)
      return
      end

      subroutine func(xf,yf,zf,rf,u,ux,uy,uz,ur,uxx,uyy,uzz,urr,t,x,y,z
     &           ,r,ix,npde)
c This subroutine defines flux terms for which it is desirable that they be
c differenced as a whole, using centered or skewed differences. Four flux
```

```
c terms are defined corresponding to the four axes so that it is possible
c to calculate their first derivatives with respect to the corresponding axes.
c  *****optional  modifications*****change  dimension  statement  if  second  form
c     of subroutine func   is to be used*********
    double precision xf(npde),yf(npde),zf(npde),rf(npde),u(npde),ux
   &            (npde),uy(npde),uz(npde),ur(npde),uxx(npde),uyy
   &            (npde),uzz(npde),urr(npde),t,x,y,z,r
    integer ix,npde
c*******modifications******enter  flux  terms  for  the  problem*****
c             no flux term
    return
    end

    subroutine bound(u,b,x,y,z,r,t)
c This subroutine defines the boundary conditions - a*u + b*(du/dn) = c
c    a,b,c are functions of x,y,z,r,t,u(i). du/dn is the normal derivative at
c    the boundary.
c The a,b and c values are entered in variable b in the following way:
c b(i,1) , b(i,2) and b(i,3) ---> a,b and c at x=x(1) for the i-th component
c b(i,4) , b(i,5) and b(i,6) ---> a,b and c at x=x(nx) for the i-th component
c b(i,7) , b(i,8) and b(i,9) ---> a,b and c at y=y(1) for the i-th component
c b(i,10) , b(i,11) and b(i,12) ---> a,b and c at y=y(ny) for the i-th component
c b(i,13) , b(i,14) and b(i,15) ---> a,b and c at z=z(1) for the i-th component
c b(i,16) , b(i,17) and b(i,18) ---> a,b and c at z=z(nz) for the i-th component
c b(i,19) , b(i,20) and b(i,21) ---> a,b and c at r=r(1) for the i-th component
c b(i,22) , b(i,23) and b(i,24) ---> a,b and c at r=r(nr) for the i-th component
c CAUTION: Enter all values of b. Default values are not zero!
    integer npde
     include 'inc.for'
    double precision u(npde),b(npde,24),x,y,z,r,t
c****modification(s)*****enter   boundary   conditions******
c at x=0,
c       u(1) = 1
    b(1,1)=1.d0
    b(1,2)=0.d0
    b(1,3)=1.d0
c at x= 0.2, _
c      At infinity , u(1) = 0.
    b(1,4)=0.d0
    b(1,5)=0.d0
    b(1,6)=0.d0
     return
    end

    block data bdat
      integer npde,nx,ny,nz,nr,meth,idim,mff
      include 'inc.for'
c Include file dims.h sets up all the common blocks.
      include 'dims.h'
c***modification***make number of zeros in data statement consistent with
c     dimensioning of meth for this problem*******
      data meth/12*0/
     end
```

Include file 'inc.for' : **parameter(npde=1, nx=11, ny=11, nz=41, nr=1)**



3.2. Example 2 :Three-dimensional Fracture Problem: The system consists of a narrow rectangular fracture which contains colloids of various sizes flowing in groundwater at a certain velocity. The colloids are instantly adsorbed into the rock matrix on coming in contact with it. In the principal flow direction, the transport is dominated by the convection term while in the other direction, diffusion is the dominant process. Since the diffusivity is a function of particle size, the PSD (particle size distribution) at the outlet is very different from the PSD at the inlet.

For this example, we are taking a very idealized boundary condition (instantaneous adsorption) and neglecting effects like agglomeration, birth, death,

etc. However, as can be seen, these terms are very easily incorporated into this example.
The equation is,

$$\frac{\partial u}{\partial t} = - v_{av}\left(1 - \left(\frac{y}{\delta} \right)^2 \right) \frac{\partial u}{\partial x} + D_x \frac{\partial^2 u}{\partial x^2} + \mathbf{D}_y \frac{\partial^2 u}{\partial y^2}$$

The velocity profile above can be derived for a rectangular fracture. Here v_{av} (average velocity) = 2/3 v_{max} (maximum velocity). The parameters for the simulation are taken from Rundberg's report.

$$v_{max} = 0.0024 \text{ cm/s}$$
$$D_x = 0.0011 \text{cm}^2\text{/s}$$
$$\delta = \text{half-width of the fracture} = 0.0295 \text{ cm}$$
$$\text{Length of fracture} = 12 \text{ cm.}$$

Assuming spherical particles and substituting values for water at 20 $^\circ$ C into the Stokes-Einstein equation, we get

$$\mathbf{D}_y = \frac{2.13 \times 10^{-9}}{\text{radius (in microns)}} \text{ cm}^2\text{/s}$$

If we were to include aggregation and other processes, we would use another axis as he size axis. This procedure will be followed here also since it makes data interpretation easier. Thus we have three independent axes. We will use a grid of 11 x 11 x 41 (NX x NY x NZ) where, as indicated in the previous example, we must use the axes as follows,

X ----> Size axis. We use a logarithmic scaling from 1 nm to 1 μ.
Thus, $X = \ln (r / 1\mu) / \ln (1\text{nm} / 1\mu)$ where r is the radius of the particle.
Y ----> Width of the fracture, $0 \le Y \le 0.0259$ cm.
Z ----> Length of the fracture, $0 \le X \le 12$ cm.

Initial Conditions :	$u = 0$	
Boundary Conditions :		
at $X = 0$,	no boundary condition imposed	(size axis).
at $X = 1$,	no boundary condition imposed	(size axis).
at $Y = 0$,	$\partial u / \partial y = 0$	(symmetry).
at $Y = 0.0295$,	$u = 0$	(instant adsorption).
at $Z = 0$,	$u = 1.0$	(input PSD).
at $Z = 12$,	boundary condition at infinity, i.e. $u = 0$ at Z = infinity.	

We seek a solution at t = 5000 s. Other simulations in two dimensions (taking particles of_one particular size) with a variety of boundary conditions at the tuff and considering unsaturated flow have been performed using CTCN.

3.2.1. Code listing for example 2 :

```
      program user
c This program interfaces with CTCN to solve systems of coupled, nonlinear
c PDEs of upto second order in upto four spatial axes and first order in
c time. Thus we solve for u(i,j,k,l,m) where,
c             i = 1 to npde (number of PDEs)
c             j = 1 to nx (number of points in the x axis)
```

```
c               k = 1 to ny (number of points in the y axis)
c               l = 1 to nz (number of points in the z axis)
c               m = 1 to nr (number of points in the r axis)
c  The problem should be set up so that the axes should be used in order x,y,z,r.
c  thus, a 1-dimensional problem MUST use the x-axis only.  A 2-D problem must
c  use the x and y as principal axes (not x and z or y and r or y and z etc.).
c also, after this the number of points must be ordered as nx <= ny <= nz <= nr.
c  This is only necessary for using the LSODES solver efficiently.
c   Thus for a two dimensional 10 x 61 grid, nr=nz=1, nx=10 , ny=61.
c T'' basic problem parameters are defined in this main program. The eqn.s
c are written out in subroutine eqn, the boundary conditions in subroutine
c bound and flux terms in subroutine func. Boundary conditions may be of the
c general form, --> a*u + b*(du/dn) = c --> a,b and c are functions of the
c spatial coordinates,time and other u variables. du/dn is the derivative
c normal to the boundary. Boundary conditions need not be specified at all,
c in which case a cubic extrapolation techniqu is used.
c      For some problems it may be desirable to difference certain terms by
c skewed differences by considering the variation with u of the terms to be
c differenced. in this case, these terms may be entered using subroutine
c func and the appropriate differencing method selected. See subroutine
c func for further details.
c      Thus, to solve any given system, the user needs to modify this program
c and  include file INC.FOR.
c Include file 'inc.for' sets up the dimensions.
      integer npde,nx,ny,nz,nr,meth,idim
      include 'inc.for'
      double precision u(npde,nx,ny,nz,nr),tol,dx,dy,dz,dr,du,x,y,z,r
     .common /c1/ tol,dx,dy,dz,dr,du
      common /c6/ x(nx),y(ny),z(nz),r(nr)
      common /c7/ idim,meth(npde,12)
c This is the user-defined program specific to a particular problem
c and sets up parameters for CTCN.
c      tout(i)---> array of output times. nout=i
c      tint---> initial value for time.
c      tlast---> final value of t. Integration is done upto tlast.
c      rtol---> relative tolerance for the ODE integrator.
c      atol---> absolute tolerance for the ODE integrator. Can be a vector
c               defined for each ODE, if so, set itol(below)=2.
c   *********this is where the  modifications begin***********
c    ***** modification**** dimension tout*********
      double precision tout(1),tint,tlast,rtol,atol
c      nout---> no. of output times. Dimension of the tout arra
c     mf---> ODE integration method flag, the solver LSODPK is used.
c      itol---> 1 or 2 accordingly as atol is a scalar or a vector.
      integer nout,mf,itol
c Bdat is the block data subprogram at the end of this file.
```

```
      external bdat
c Integration starts from t=tint
c    ***modification***set  tint****
      tint=0.d0
c Enter array of times when output is required.
c****modification*****set   tout(nout)=******
        tout(1)=5.d3
c Integration stops at t=tlast.
c*****modification******set   tlast********
      tlast=5.d3
c The number of output times=array size for tout=nout
c   *****modification*****set  nout******
      nout=1
c Itol,rtol and atol are used to specify error tolerances for LSODPK-the ODE
c integrator.
c****modifications****set  itol,rtol  and  atol******
      itol=1
      rtol=1.d-4
      atol=1.d-6
c ode solvers require an ODE integration flag
c          for LSODPK solver          = mf = 22 for stiff problems, &
c                                   = 10 for non-stiff problems.
c          for LSODES solver          = mf = 222 for stiff problems, &
c                                   = 10 for non-stiff problems.
c*****modification*****set   mf********
      mf=22
c The type of differencing for each term is indicated by meth.
c    meth(i,1)= flag for ur(i), i.e. du/dr for the i-th pde.
c    meth(i,2)= flag for urr(i), i.e. d2u/dr2 for the i-th pde.
c    meth(i,3)= flag for fr(i), i.e. d(rf)/dr for the i-th pde.
c    meth(i,4)= flag for uz(i), i.e. du/dz for the i-th pde.
c    meth(i,5)= flag for uzz(i), i.e. d2u/dz2 for the i-th pde.
c    meth(i,6)= flag for fz(i), i.e. d(zf)/dz for the i-th pde.
c    meth(i,7)= flag for uy(i), i.e. du/dy for the i-th pde.
c    meth(i,8)= flag for uyy(i), i.e. d2u/dy2 for the i-th pde.
c    meth(i,9)= flag for fy(i), i.e. d(yf)/dy for the i-th pde.
c    meth(i,10)= flag for ux(i), i.e. du/dx for the i-th pde.
c    meth(i,11)= flag for uxx(i), i.e. d2u/dx2 for the i-th pde.
c    meth(i,12)= flag for fx(i), i.e. d(xf)/dx for the i-th pde.
c    if meth(*,*) = 2 ---> second order centered differences
c                 = 4 ---> fourth order centered differences
c                 = -2 ---> second order skewed differences
c                      ( only for flux terms written into rf,zf,yf,xf )
c                 = -4 ---> third order skewed differences
c                      ( only for flux terms written into rf,zf,yf,xf )
c   Default value = 0
```

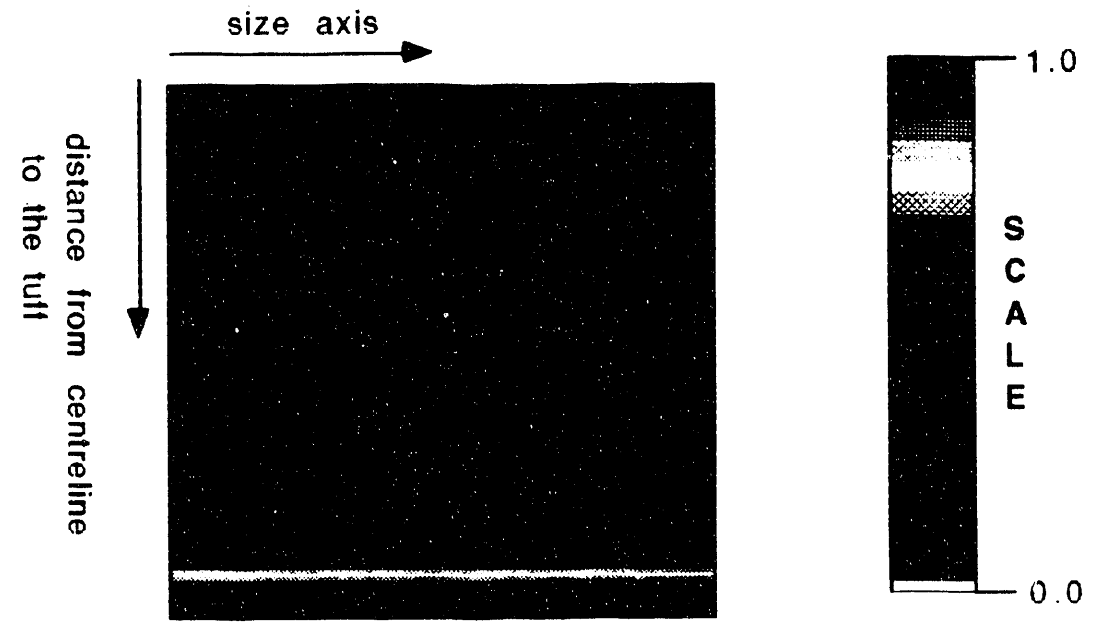
```
c*******modification(s)*******set meth ( *, *) ********
       meth(1,4)=2
       meth(1,5)=2
       meth(1,8)=2
c Enter the axes grid sizes, default = 1.d0. dr=delta(r),...etc.
c********modification(s)******enter grid sizes****
    dx=1.d0/dble(nx-1)
    dy=2.95d-2/dble(ny-1)
    dz=1.2d1/dble(nz-1)
c Enter grid point values.Input r(i)--> grid points on r axis....etc.
c*******modification(s)*****enter grid values*******
    do 10 i=1,nx
      x(i)=(i-1)*dx
  10 continue
    do 20 i=1,ny
      y(i)=(i-1)*dy
  20 continue
    do 30 i=1,nz
      z(i)=(i-1)*dz
  30 continue
c Enter initial conditions.  no default values.
c*****modifications*******enter initial conditions******
    do 50 l=1,nz
    do 50 k=1,ny
    do 50 j=1,nx
            u(1,j,k,l,1)=0.D0
  50  continue
c Call the interfacing subroutine CTCN
      call ctcn(u,rtol,atol,tint,tlast,tout,nout,mf,itol)
     stop
     end

      subroutine eqn(ut,u,ux,uy,uz,ur,uxx,uyy,uzz,urr,fx,fy,fz,fr,t,x,y
     &           ,z,r,ix)
c This subroutine is used to enter the PDEs
      integer ix,npde
       include 'inc.for'
c  *****optional modifications*****change dimension statement if second fo
c      of subroutine EQN  is to be used*********
      double precision ut(npde),u(npde),ux(npde),uy(npde),uz(npde),ur
     &           (npde),uxx(npde),uyy(npde),uzz(npde),urr(npde),
     &           fx(npde),fy(npde),fz(npde),fr(npde),t,x,y,z,r
c Input the PDEs in the form ut(i)=f(u(i),x,y,z,r,t,ux(i),uy(i),uz(i),ur(i)
c    ,uxx(i),uyy(i),uzz(i),urr(i),fx(i),fy(i),fz(i),fr(i)
c****modifications******problem specific equations****
      ut(1) = - 2.4d-3*(1.d0-(y/2.95d-2)**2)*uz(1)+1.1d-3*uzz(1)+2.13d-9/
```

```
   &          (exp(x*6.90776) *1.d-3)*uyy(1)
c**** 6.90776 = ln ( 1 micron / 1 nm ) = ln (1000) ****
    return
    end

    subroutine func(xf,yf,zf,rf,u,ux,uy,uz,ur,uxx,uyy,uzz,urr,t,x,y,z
   &          ,r,ix,npde)
c This subroutine defines flux terms for which it is desirable that they be
c differenced as a whole, using centered or skewed differences. Four flux
c terms are defined corresponding to the four axes so that it is possible
c to calculate their first derivatives with respect to the corresponding axes.
c *****optional modifications*****change dimension statement if second form
c     of subroutine func  is to be used*********
    double precision xf(npde),yf(npde),zf(npde),rf(npde),u(npde),ux
   &          (npde),uy(npde),uz(npde),ur(npde),uxx(npde),uyy
   &          (npde),uzz(npde),urr(npde),t,x,y,z,r
    integer ix,npde
c*******modifications******enter flux terms for the problem*****
    return
    end

    subroutine bound(u,b,x,y,z,r,t)
c This subroutine defines the boundary conditions - a*u + b*(du/dn) = c
c    a,b,c are functions of x,y,z,r,t,u(i). du/dn is the normal derivative at
c    the boundary.
c The a,b and c values are entered in variable b in the following way:
c b(i,1) , b(i,2) and b(i,3) ---> a,b and c at x=x(1) for the i-th component
c b(i,4) , b(i,5) and b(i,6) ---> a,b and c at x=x(nx) for the i-th component
c b(i,7) , b(i,8) and b(i,9) ---> a,b and c at y=y(1) for the i-th component
c b(i,10) , b(i,11) and b(i,12) ---> a,b and c at y=y(ny) for the i-th component
c b(i,13) , b(i,14) and b(i,15) ---> a,b and c at z=z(1) for the i-th component
c b(i,16) , b(i,17) and b(i,18) ---> a,b and c at z=z(nz) for the i-th component
c b(i,19) , b(i,20) and b(i,21) ---> a,b and c at r=r(1) for the i-th component
c b(i,22) , b(i,23) and b(i,24) ---> a,b and c at r=r(nr) for the i-th component
c CAUTION: Enter all values of b. Default values are not zero!
    integer npde
     include 'inc.for'
    double precision u(npde),b(npde,24),x,y,z,r,t
c****modification(s)*****enter boundary conditions******
c at x=0,
c     no boundary condition imposed
    b(1,1)=0.d0
    b(1,2)=0.d0
    b(1,3)=0.d0
c at x=1,
c     no boundary condition imposed
```

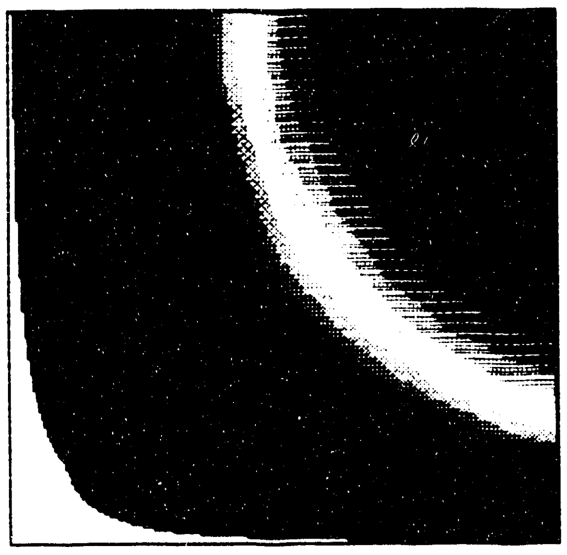
Plots for Example Problem 2

The number density of colloids is plotted against the width and the size axes. Thus these plots are for a particular point on the length axis. The time is 5000 sec.

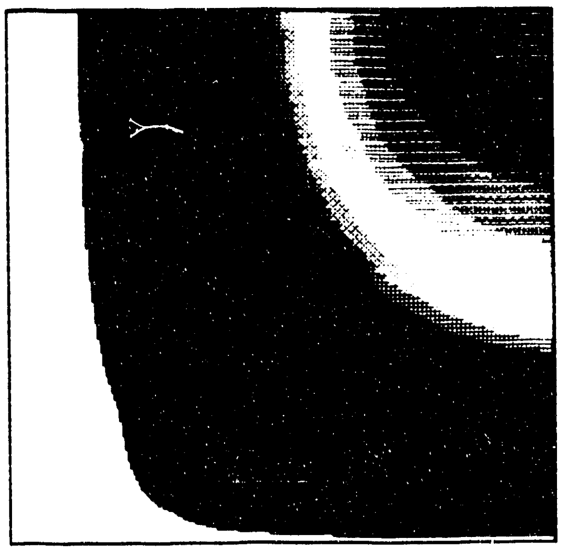
At x = 0. (input)



At x = 3.0.

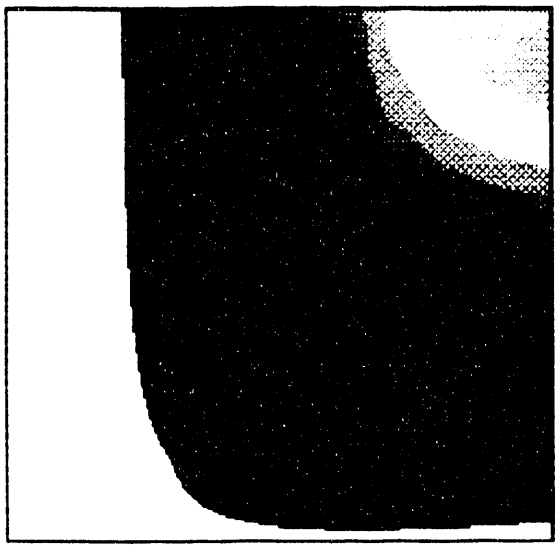


At x = 6.0.

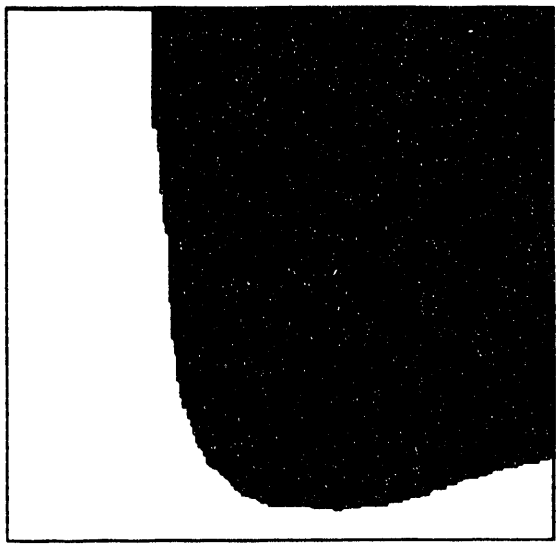


Plots for Example Problem 2 (contd.)

At x = 9.



At x = 12.



```
      b(1,4)=0.d0
      b(1,5)=0.d0
      b(1,6)=0.d0
c  at y=0,
c       ∂( u(1) ) / ∂y = 0
      b(1,7)=0.d0
      b(1,8)=1.d0
      b(1,9)=0.d0
c  at y= 0.0295 ,
c        u(1)  = 0
      b(1,10)=1.d0
      b(1,11)=0.d0
      b(1,12)=0.d0
c  at z=0,
c       enter the input psd as a function of size(=x). u = function(x)
      b(1,13)=1.d0
      b(1,14)=0.d0
      b(1,15)=1.d0
c  at z=12,
c       boundary condition at infinity
      b(1,16)=0.d0
      b(1,17)=0.d0
      b(1,18)=0.d0
       return
      end

      block data bdat
        integer npde,nx,ny,nz,nr,meth,idim,mff
        include 'inc.for'
c Include file dims.h sets up all the common blocks.
        include 'dims.h'
c***modification***make number of zeros in data statement consistent with
c       dimensioning of meth for this problem*******
        data meth/12*0/
       end
```

Include file 'inc.for' : **parameter(npde=1, nx=11, ny=11, nz=41, nr=1)**

4. TEST CASES

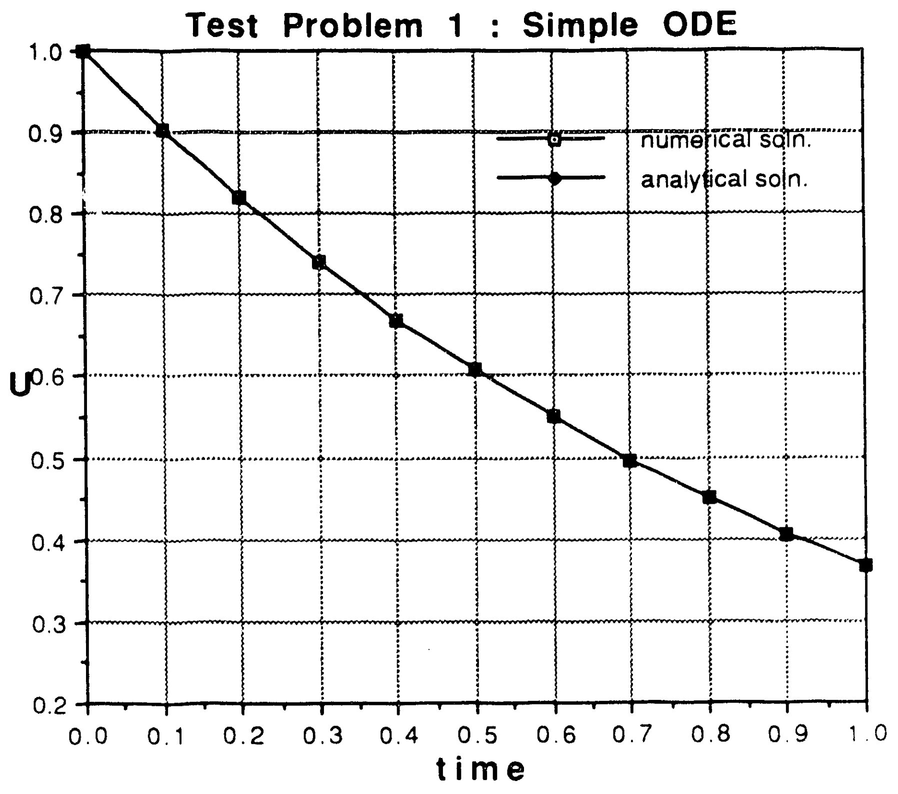
4.1. Simple ODE

This is a simple ordinary differential equation of the form,

$$du / dt = -u .$$

The dimensionality of this problem is zero. The initial condition is taken to be, $u_0 = 1.0$. The analytical solution is

$$u = u_0 \exp(-t).$$



4.2. Euler Equations of Gas Dynamics (Hyman,1976)

Three variables (density, momentum and energy) of a gas are describ the following equations,

$$\frac{\partial w}{\partial t} + \frac{\partial}{\partial x} F(w) = \delta \frac{\partial^2 w}{\partial x^2} \quad \text{where} \quad w = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \quad \text{and} \quad F(w) = v\, w + \begin{pmatrix} 0 \\ p \\ p\, v \end{pmatrix}$$

Here, u_1 = mass density, , u_2 = momentum,, u_3 = total energy per unit volum
Therefore, v = velocity = u_2 / u_1 and p = pressure = $2/3\,(u_3 - 1/2\, u_1 v^2)$.
$\delta = 0.006$ = artificial dissipation coefficient.

Initial Conditions :
For $0 \le x \le 3/4$ For $3/4 \le x \le 2$

$u_1 = 1.0$	$u_1 = 0.125$
$u_2 = 0.0$	$u_2 = 0.0$
$u_3 = 1.5$	$u_3 = 0.15$

Boundary Conditions :

At both boundaries ($x = 0$ and $x = 2$), they are:

$$(u_1)_x = 0, \quad u_2 = 0.0, \quad (u_3)_x = 0.$$

Results agreed with Hyman's data within the error tolerance specified for the ODE solver.

4.3. **Reaction Diffusion Equations** (Hyman, 1976)

A system of autocatalytic chemical reactions, described by Robertson, can be written in the form,

$$A \xrightarrow{k_1} B \quad , \quad k_1 = 0.04$$

$$B + C \xrightarrow{k_2} A + C \quad , \quad k_2 = 10^4$$

$$2\,B \xrightarrow{k_3} C + B \quad , \quad k_3 = 3 \times 10^7$$

The differential equations describing the kinetics of this system are very stiff, and Robertson's model has become a standard test problem for stiff ODE methods. We modify the equations to include spatial variations and passive diffusion. The corresponding system of PDEs for these reactions is,

$$\frac{\partial u}{\partial t} = - k_1\, u + k_2\, v\, w + d \frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial v}{\partial t} = k_1\, u - k_2\, v\, w - k_3\, v^2 + d \frac{\partial^2 v}{\partial x^2}$$

$$\frac{\partial w}{\partial t} = k_3\, v^2 + d \frac{\partial^2 w}{\partial x^2}$$

where the independent variable x lies between 0 and 1, the diffusion coefficient $d = 0.1$ and $u = [A]$. $v = [B]$ and $w = [C]$ denote the concentrations of the chemical components.

Initial Conditions : $v = w = 0$ and $u = 1.5$ (for $x \geq 0.50$) and $u = 0.5$ (for $x < 0.5$).

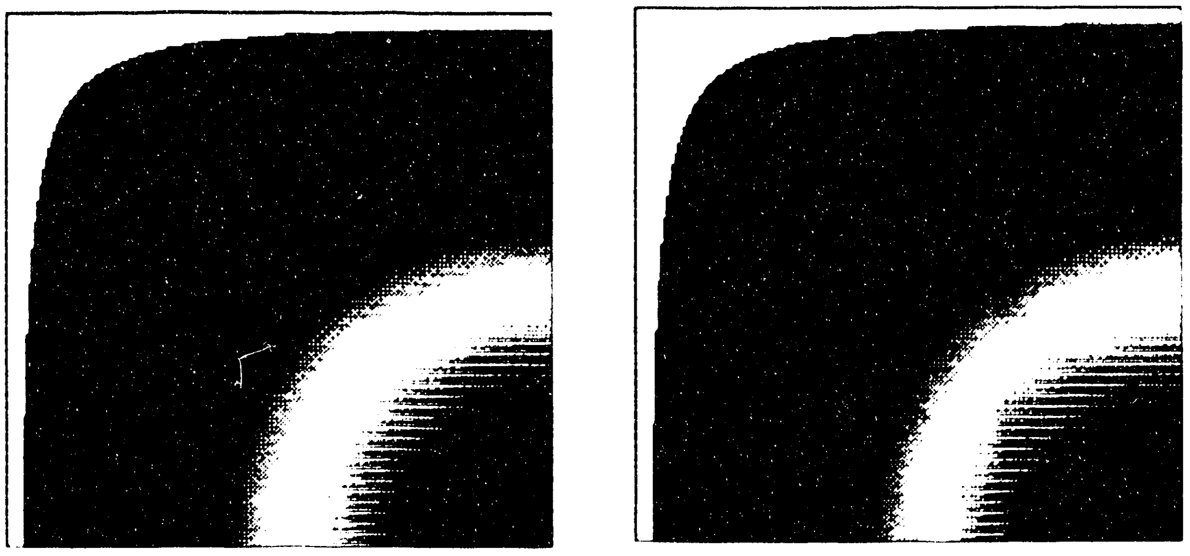
Boundary Conditions : At both boundaries ($x = 0$ and $x = 1$), we have

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = \frac{\partial w}{\partial x} = 0$$

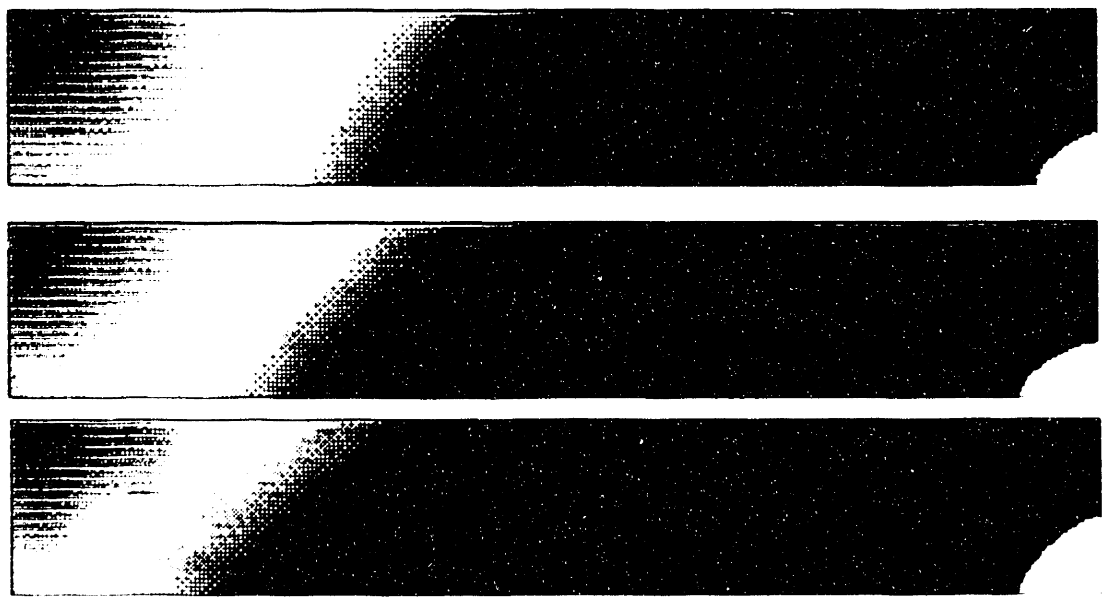
Results agreed with Hyman's data within the error tolerance specified for the ODE solver.

4.4. **Elliptic Equation** (Melgaard & Sincovec, 1981)

This example is an elliptic PDE that represents a parallel plate heated by a nearly oblong object. The PDE is,



Test Problem 4.4 : Elliptic Equation



Test Problem 4.5 : Burger's Equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - 6\,x\,y\,e^x\,e^y\,(x\,y + x + y - 3)$$

on the grid $0 \le x \le 1, 0 \le y \le 1$

Boundary and Initial Conditions : $u = 0$.

For this problem the steady-state solution is $3\,e^x\,e^y\,(x - x^2)\,(y - y^2)$, which can be approximated by integrating the original problem until the time derivative is nearly zero. We choose a uniformly spaced 10 x 10 mesh and relative and absolute error tolerances of 10^{-6} and 10^{-8} respectively. Another run with a 20 x 20 mesh reduced the error by approximately 1/4th, indicating that the errors were mainly due to the spatial discretization as expected.

4.5. Burger's Equation (Melgaard & Sincovec, 1981)

The second two-dimensional test case is the familiar Burger's Equation,

$$\frac{\partial u}{\partial t} = -\,u\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) + 0.01\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

Initial and Boundary Conditions : $u = (1.0 + e^{x+y-t})^{-1}$.

With these conditions the solution is a straight-line wave (u is constant along $x = -\,y$) moving in the direction $\theta = \pi / 4$

Because of this straight-line nature of the solution, we may obtain the numerical solution of this problem by determining the solution over a rectangular strip. Therefore, we choose uniformly spaced rectangular meshes with NX = 5, NY = 31 and NX = 10, NY = 61 defined over $0 \le y \le 1$ and $0 \le x \le NX/(NY - 1)$ and relative and absolute error tolerances of 10^{-5} and 10^{-7} respectively. As before, the errors were mainly due to the spatial discretization.

4.6. Coupled System of PDEs (Melgaard & Sincovec, 1981)

The third two-dimensional case consists of two coupled, nonlinear PDEs with all three types of boundary conditions,

$$\frac{\partial u}{\partial t} = v\left(\frac{\partial}{\partial y}\left(v\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial y}\left(u\frac{\partial v}{\partial y}\right)\right) - 3\,v\frac{\partial v}{\partial x} + 4\frac{\partial u}{\partial y} - \frac{\partial v}{\partial y}$$

$$\frac{\partial v}{\partial t} = \frac{\partial}{\partial x}\left(v\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial x}\left(u\frac{\partial v}{\partial x}\right) - \frac{\partial v}{\partial y}$$

on a uniform grid in the region, $0 \le x \le 1, 0 \le y \le 1$.

Initial Conditions : $u = x + y$, $v = 2\,x + 3\,y$.

Boundary Conditions :

At $x = 0$,	$u = t + y$,	$v = t + 3\,y$.
At $x = 1$,	$u = t + y + 1$,	$\partial v/\partial x = 2$.
At $y = 0$,	$\partial u/\partial y = 0$,	$v + \partial v/\partial y = t + 2\,x + 3$.
At $y = 1$,	$v\,\partial u/\partial y = t + 2\,x + 3$,	$v = t + 2\,x + 3$.

Analytical Solution : $u = t + x + y$, $v = t + 2\,x + 3\,y$.

Results obtained were consistent with the analytical solution.

4.7. Anisotropic Diffusion

This problem is a simple three-dimensional anisotropic diffusion equation,

$$\frac{\partial u}{\partial t} = D_x \frac{\partial^2 u}{\partial x^2} + D_y \frac{\partial^2 u}{\partial y^2} + D_z \frac{\partial^2 u}{\partial z^2}$$

on a unit cube. The diffusion coefficients may be given different values and the profiles judged accordingly.

Initial Conditions : $u = 0$ except at the center where $u = 5$.

Boundary Conditions : Homogeneous Neumann boundary conditions are imposed. At all boundaries, the outward normal derivative is zero (impermeable walls).

Results were consistent with what would be qualitatively expected.

4.8. Lotka - Volterra Model in 3-D (Brown & Hindmarsh, 1988)

This problem is based on a reaction-diffusion system arising from a Lotka-Volterra predator-prey model, with diffusion effects in three space dimensions. There are two species variables, c^1 and c^2 representing respectively the prey and predator species densities over a spatial habitat consisting of a unit cube, and time t ranging from 0 to 10 seconds. The equations are,

$$\frac{\partial c^i}{\partial t} = d_i \, \Delta c^i + f(c^1, c^2) \qquad (i = 1, 2)$$

$$d_1 = 0.05, \qquad d_2 = 1.0$$

$$f^1(c^1, c^2) = c^1(b_1 - a_{11} c^1 - a_{12} c^2), \qquad f^2(c^1, c^2) = c^2(b_2 - a_{21} c^1 - a_{22} c^2),$$

$$a_{11} = 10^6, \quad a_{12} = 1, \qquad a_{21} = 10^6 - 1, \; a_{22} = 10^6 ,$$

$$b_1 = b_2 = (1 + \alpha x y z)(10^6 - 1 + 10^{-6}).$$

Initial Conditions : $c^1(x, y, z, 0) = 500 + 250 \cos(\pi x) \cos(3 \pi y) \cos(10 \pi z)$
$c^2(x, y, z, 0) = 200 + 150 \cos(10 \pi x) \cos(\pi y) \cos(3 \pi z)$.

Boundary Conditions : Homogeneous Neumann boundary conditions at all boundaries. The outward normal derivatives are zero.

The coefficients have been chosen so that as t —> infinity, the solution of the system approaches a steady state which is (intentionally) not flat in space. This steady state is given roughly by the asymptotic solution of the problem without diffusion, namely:

$$\underline{} \; c^1 = (1 - 10^6)(1 + \alpha x y z), \qquad c^2 = 10^{-6}(1 + \alpha x y z).$$

The two PDEs are discretized on a regular J by K by L grid. We consider $\alpha = 0$ and $\alpha = 0.2$ and vary J = K = L from 6 to 20. Tolerance parameters are taken to be RTOL $= 10^{-6}$ and ATOL $= 10^{-8}$. Results were consistent with the asymptotic solution but run-times were higher since Brown and Hindmarsh tested a problem-specific code.

REFERENCES

Saltelli, A., Avogadro, A., and Bidoglio, G., "Americium Filtration in Glauconitic Sand Columns," Nuclear Technology, 67, 245, 1984.

Fried, S. M., et al., Argonne National Laboratory report ANL-76-127, 1976.

Champ, D. R., Merritt, W. F., and Young, J. L., "Potential for the Rapid Transport of Plutonium in Groundwater as Demonstrated by Core Column Studies," Scientific Basis for Nuclear Waste Management, Werner Lutze (Ed), North-Holland, NY, 5, 745-754, June,1982.

Travis, B. J. and Nuttall, H. E., "A Transport Code for Radiocolloid Migration: With an Assessment of an Actual Low-Level Waste Site, Scientific Basis for Nuclear Waste Management VIII," edited by C. M. Jantzen, J. A. Stone, R. C. Ewing, (Materials Research Society, Pittsburgh, Pennsylvania), 44, 969-976, 1985.

Fried, S. M., Friedman, A. M., Hines, J. J., and Quarterman, L. A., "Annual Report in DWMT Project AN0115A, FY 1975," Argonne National Laboratory report ANL-75-64, 1975.

Ho, C. H. and Miller, N. H., J. Colloid Interface Sci., 1986, 113, 222-240.

Means, J. L., Crerar, D. A., and Duguid, J. O., Science, 1978, 200, 1477-81.

Champ, D. R. et al., Water Pollut. Res. J. Can., 1984, 19, 35-54.

Gschwend, P. M., and Reynolds, M. D., J. Contam. Hydrol., 1987, 1, 309-27.

McDowell-Boyer, L. M., Hunt, James R., and Sitar, Nicholas, "Particle Transport Through Porous Media," Water Resources Research, 22, (13), 1901-1921, 1986.

Randolph, A. D., "A Population Balance for Countable Entities," Can. J. Chem Eng., 42, 280-81, 1962.

Hulburt, H. M., and Katz, S., "Some Problems in Particle Technology," Chem. Engineering Science, 19, 555-74, 1964.

Randolph, A. D., and Larson, M. A., Theory of Particulate Processes, 2nd ed. Academic Press, 1988.

Gelbard, F., and Seinfeld, John H., "Numerical Solution of the Dynamic Equation for Particulate Systems," Journal of Computational Physics, 28, 357-375, 1978

Marchal, P., David, R., Klein, J. P., and Villermaux, J., "Crystallization and Precipitation Engineering-I. An Efficient Method for Solving Population Balance in Crystallization with Agglomeration," Chemical Engineering Science, 43(1), 59-67, 1988.

Madsen, Niel K., and Sincovec, Richard F., "Software for Nonlinear Partial Differential Equations," ACM Transactons on Mathematical Software, 1(3), 232-260, 1975.

Loeb, A. M., "User's Guide to a New User-Oriented Subroutine for the Automatic Solution of One-Dimensional Partial Differential equations," Advances in Computer Methods for Partial Differential Equations, R. Vichnevetsky (ed.), Publ. AICA-1975, 167-175.

Hyman, J. M., "The Method of Lines Solution of Partial Differential Equations," NYU Report COO-3077-139, 1976.

Carver, M. B., "FORSIM: A Fortran Oriented Simulatin Package for the Automated Solution of Partial and Ordinary Differential Equation Systems," Report AECL-4608, Atomic Energy of Canada, Limited, Chalk River, Ontario, 1973.

Liskovets, O. A., "The Method of Lines (Review)," Differentsial'nye Uraveniya, 1(12), 1662-78, 1965.

Melgaard, David K., and Sincovec, Richard F., "General Software for Two-Dimensional Nonlinear Partial Differential Equations," ACM Transactons on Mathematical Software, 7(1), 106-125, 1981.

Brown, Peter N., and Hindmarsh, Alan C., "Matrix-Free Methods for Stiff Systems of ODE'S," SIAM J. Numer. Anal., 23(3), 610-638, 1986.

Hindmarsh, Alan C., "ODE Solvers for Time-Dependent PDE Software," PDE Software: Modules, Interfaces and Systems, B. Engquist and T. Smedsas (ed [illegible] 325-341, Elsevier Science Publishers B. V. (North-Holland), IFIP, 1984.

Machura, M., and Sweet, R. A., "A Survey of Software for Partial Differential Equations," ACM-Trans. Math. Softw., 6, 1980, 461-488.

Scheisser, W. E., DSS/2, Differential Systems Simulator, an introduction to the numerical method of lines integration of partial differential equations, Lehigh University, Bethlehem, PA, 1977

Sharma, M. M. and Yortsos, Y. C., "Transport of Particulate Suspensions in Porous Media : Model Formulation," AIChE Journal, 33(10), 1636-1643, 1987.

van Olphen, H., An Introduction to Clay Colloid Chemistry, 2d ed., J. Wiley & Sons, NY, 1977.

Jantzen, C. M. and Bibler, N. E., "The Role of Groundwater Oxidation Potential and Radiolysis on Waste Glass Performance in Crystalline Repository Environments," Proceedings of the Material Science Society, Stockholm, Sweden, Sept. 8 - 11, 1985.

Hidy, G. M. and Brock, J. R., "Topics in Current Aerosol Research," Pergamon Press, NY, 1971.

Hurd, A. J., "Diffusion Limited Aggregation of Silica Microspheres in Two Dimensions," Physics of Complex and Supermolecular Fluids, S. Safran & N. Clark ed.s, Wiley, 1986.