UNIVERSITY OF MINNESOTA
ST. ANTHONY FALLS LABORATORY
Engineering, Environmental and Geophysical Fluid Dynamics

Project Report No. 412

Two-Dimensional Water Quality Model for Unsteady Advection-Diffusion of Nonconservative Substances

by

Dragoslav L. Stefanovic and Heinz G. Stefan

Prepared for

U. S. ENVIRONMENTAL PROTECTION AGENCY

Office of Research and Development
Washington, D. C.
and
Mid-Continent Ecology Division
Duluth, Minnesota

December 1997
Minneapolis, Minnesota

UNIVERSITY OF MINNESOTA
ST. ANTHONY FALLS LABORATORY
Engineering, Environmental and Geophysical Fluid Dynamics

Project Report No. 412

Two-Dimensional Water Quality Model for Unsteady Advection-Diffusion of Nonconservative Substances

by

Dragoslav L. Stefanovic and Heinz G. Stefan

Prepared for

U. S. ENVIRONMENTAL PROTECTION AGENCY

Office of Research and Development
Washington, D. C.
and
Mid-Continent Ecology Division
Duluth, Minnesota

December 1997
Minneapolis, Minnesota

Table of Contents

ABSTRACT

A numerical model for calculation of advective-diffusive transport of nonconservative substances in two-dimensional environments was developed. The numerical method is based on the splitting-operator approach, in which the advection, the diffusion and the chemical/biological kinetic processes are calculated separately in one time step. Special attention was paid to the advection operator, which introduces essential difficulty in many numerical methods, and to the linearized source term which, in many cases, has proven to cause instability problems. The model calculates pure advection by the explicit Holly-Preissmann method of characteristics, and diffusion plus source/sink terms by an extended implicit alternate-direction (ADI) method. By comparison with analytical results for fronts and discrete mass releases it is established that numerical separation of differential operators does not induce significant errors in the solution or the physical realism of the results. The numerical scheme is accurate, stable and efficient because it eliminates the need to solve a pentadiagonal algebraic systems, replacing it with two tridiagonal ones. The computational method is intended for further use in the study of a two-dimensional lake hydrodynamic and transport field, driven either by forced (wind induced) or natural (buoyancy induced) convection.

ACKNOWLEDGMENTS

The investigation described herein was conducted for the U.S. Environmental Protection Agency as part of a project to anticipate the possible effects of projected climate change on water resources and ecosystems. The methodology developed and described in this report is intended for use in lake water quality modeling. Barbara M. Levinson (Office of Research and Development, Washington, D. C.) and John G. Eaton and Virginia M. Snarski (Mid-Continent Ecology Division, Duluth, MN) were project officers.

LIST OF FIGURES

I. INTRODUCTION

Deterministic, unsteady, one-dimensional, lake water quality and fish habitat simulation models were developed in the recent past at the St. Anthony Falls Laboratory, University of Minnesota, and are being expanded, validated and applied. These models simulate daily water temperature and dissolved oxygen distributions in various classes of lakes, assuming that horizontal layers of water are well mixed, so that a one-dimensional description of the processes is appropriate. In order to add convective exchange between littoral and profundal waters of lakes to the simulations, an extension of the existing one-dimensional transport model into another dimension is required, taking into account the variability of water quality parameters with distance from a lake's shore. It becomes inevitable to simulate comprehensive lake hydrodynamics, along with transport processes, in a two-dimensional flow field, driven either by forced (wind induced) or natural (buoyancy induced) convection. It is expected that inclusion of littoral-profundal water exchange in the simulations will give valuable new information on the distribution of water quality parameters in lakes, for both open-water and ice-cover season.

Herein we are concerned with transport of momentum and contaminants subjected to turbulent diffusion and differential advection in a two-dimensional, time-dependent velocity field that is to be calculated. A significant number of computational models have been developed for that same purpose but none of them is ideal or universal. Many numerical solutions have proved to be inaccurate and physically unrealistic for particular problems, due to introduction of artificial dissipation or dispersion (Anderson et. al., 1984) in a model, thus confounding the original specification of the problem. This is especially true for calculation of the advection term because of its hyperbolic nature. If turbulent diffusion is strong enough (as in turbulent river flow), then its smoothing effect tends to mask errors in the advective portion of a numerical calculation (Holly and Preissmann, 1977), and we are not principally concerned with large numerical diffusion that could result from our computation. But the problem arises when the advection transport dominates over or is of the same order of magnitude as turbulent diffusion. This is frequently encountered in highly unsteady regimes (spills) or in a flow region whith larger variations in the magnitude and/or direction of ambient current (coastal areas and lakes).

II. CONCEPTUAL BASIS OF THE NUMERICAL METHOD

For the above mentioned reasons it is desirable to develop a numerical method which, if not completely free of numerical errors, is reliable and efficient enough for a wide range of hydrodynamic and transport scenarios. The approach taken here is based on the fractional-step (splitting) technique that enables independent calculation of intrinsic processes in the problem. By doing so, it is possible to capitalize on numerical methods which are accurate and suitable in simulating the unique physical nature of the processes involved in two-dimensional dispersion. The model proposes use of the explicit Holly-Preissmann method (Holly and Preissmann, 1977) for calculation of pure advection and the factorized implicit ADI method based on the central space discretizations (Hirsch, 1988) for calculation of pure diffusion. The necessary attention is also paid to a first-order source/sink term which sometimes has proven to be a computational nuisance, causing numerical instabilities, as described by Patankar (1980).

Two-dimensional unsteady transport with a first-order source/sink term is described by the advection-diffusion equation

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} + v\frac{\partial C}{\partial y} = \frac{\partial}{\partial x}\left(\alpha_x \frac{\partial C}{\partial x}\right) + \frac{\partial}{\partial y}\left(\alpha_y \frac{\partial C}{\partial y}\right) + aC + b \qquad (1)$$

where

$C(x,y,t)$ = mass concentration

$u(x,y,t)$ = longitudinal velocity

$v(x,y,t)$ = lateral velocity

$\alpha_i(x,y,t)$ = mass diffusion coefficient in i direction

$a(x,y,t)$ = first-order reaction rate coefficient

$b(x,y,t)$ = zero-order reaction rate coefficient

t = time

x = longitudinal direction

y = lateral direction

Transport equation (1) takes into account first and zero-order source/sink terms which allows the extension to an arbitrary order reaction rate by a standard linearization techniques (Patankar, 1980).

The essence of the fractional-step method employed here is as follows: cast the transport equation (1) in the equivalent form

$$\frac{\partial C}{\partial t} + L_c(C) - L_d(C) = 0 \tag{2}$$

where

$$L_c(C) = u\frac{\partial C}{\partial x} + v\frac{\partial C}{\partial y} = \text{advection operator} \tag{3}$$

$$L_d(C) = \frac{\partial}{\partial x}\left(\alpha_x \frac{\partial C}{\partial x}\right) + \frac{\partial}{\partial y}\left(\alpha_y \frac{\partial C}{\partial y}\right) + aC + b = \text{diffusion operator with source} \tag{4}$$

Equation (2), written for the time level n, can be further modified by the use of a Taylor series expansion

$$\frac{C^{n+1} - C^n}{\Delta t} + L_c(C^n) - L_d(C^n) = \frac{\partial^2 C^n}{\partial t^2}\frac{\Delta t}{2} + \ldots = O(\Delta t) \tag{5}$$

Introduction of an auxiliary variable C', as well as separation of Eq. (5) into two distinct equations enables one to formulate the two-step method

$$\frac{C' - C^n}{\Delta t} + L_c(C^n) = \frac{\partial^2 C^n}{\partial t^2}\frac{\Delta t}{2} + \ldots = O(\Delta t) \tag{6}$$

$$\frac{C^{n+1} - C'}{\Delta t} - L_d(C^n) = 0 \tag{7}$$

Equation (6) is a standard advection (wave) equation which will be cast back into the differential form

$$\frac{\partial C^n}{\partial t} + u\frac{\partial C^n}{\partial x} + v\frac{\partial C^n}{\partial y} = 0 \tag{8}$$

and solved by a method for pure advection. On the other hand, equation (7) is a diffusion equation, combined with a source/sink term, which can be solved in an implicit form

$$\frac{C^{n+1} - C'}{\Delta t} - \theta L_d(C^{n+1}) - (1-\theta) L_d(C') = 0 \tag{9}$$

where θ is a weighting factor between zero and one.

In order to show the error induced by the splitting technique outlined above, we can sum equations (8) and (9) to recover the original advection-diffusion form. After

rearranging and keeping only the first-order terms in the Taylor series, it follows that

$$\frac{C^{n+1}-C^n}{\Delta t}+L_c(C^n)-L_d(C^n)=O(\Delta t)+\left\{\theta L_d\left[L_d(C^n)\right]-L_d\left[L_c(C^n)\right]\right\}\Delta t+\ldots \quad (10)$$

In comparison to the basic Eq. (5) there are additional terms on the right hand side of Eq. (10), two of them being of the order $O(\Delta t)$. They are the error caused by the separation of differential operators. The magnitude of this error will not significantly affect the overall accuracy of the method, because the accuracy of the diffusion equation alone would already be $O(\Delta t)$. The pure advection equation will be solved with an error $O(\Delta x^4)$, combined with an error $O(\Delta t, \Delta x^2)$ for the diffusion equation. The latter gives the overall accuracy of the method.

The aforementioned procedure is employed for each time level by alternate computation of advection and diffusion with source, where those two are coupled by the subsidiary variable C'. The result of pure advection is C' which is in turn the initial condition for diffusion with source. After diffusion has been calculated the final result is C^{n+1} and the algorithm is ready for the next time level.

III. SOLUTION OF THE ADVECTION EQUATION

The first step in the fractional-step approach is to solve the pure advection equation (8) (here cast in vector notation) in the previously calculated velocity field $\vec{u}(\vec{r},t)$

$$\frac{\partial C(\vec{r},t)}{\partial t}+\vec{u}(\vec{r},t)\cdot\nabla C(\vec{r},t)=0 \tag{11}$$

where $\vec{r}=(x,y)$ is a position vector.

Equation (11) is a hyperbolic partial differential equation requiring one initial condition and two boundary conditions, the latter being specified at the upstream ends of the computational domain.

The substantial derivative of the concentration C, along a fluid particle trajectory $\vec{r}_0(t)$ is

$$\frac{DC}{Dt}\equiv\frac{\partial C}{\partial t}+\vec{u}\cdot\nabla C=\frac{\partial C}{\partial t}+\frac{d\vec{r}_0}{dt}\cdot\nabla C \tag{12}$$

Equations (11) and (12) imply that $DC/Dt=0$ along the trajectory $\vec{u}(\vec{r}_0,t)=d\vec{r}_0(t)/dt$, or $C(\vec{r}_0,t)=const$. One can see that the partial differential equation (11) can be replaced by two ordinary differential equations

$$\frac{DC(\vec{r}_0,t)}{Dt}=0 \tag{13}$$

and

$$\vec{u}(\vec{r}_0,t)=\frac{d\vec{r}_0(t)}{dt} \tag{14}$$

The solution algorithm for Eq. (12) therefore consists of the following procedure (Fig. 1):

- For each computational grid point (i,j) determine the trajectory (also called characteristic line) $\vec{r}_0(t)$ which goes through that point, at every time level $n+1$, using Eq. (14)

- Integrate Eq. (13) along the characteristic line, i.e. equate $C(\vec{r}_0, t^{n+1}) = C(\vec{r}_0, t^n)$, where $C(\vec{r}_0, t^n)$ is the known concentration at the foot of the trajectory (where it intersects the domain at the previous time level n).

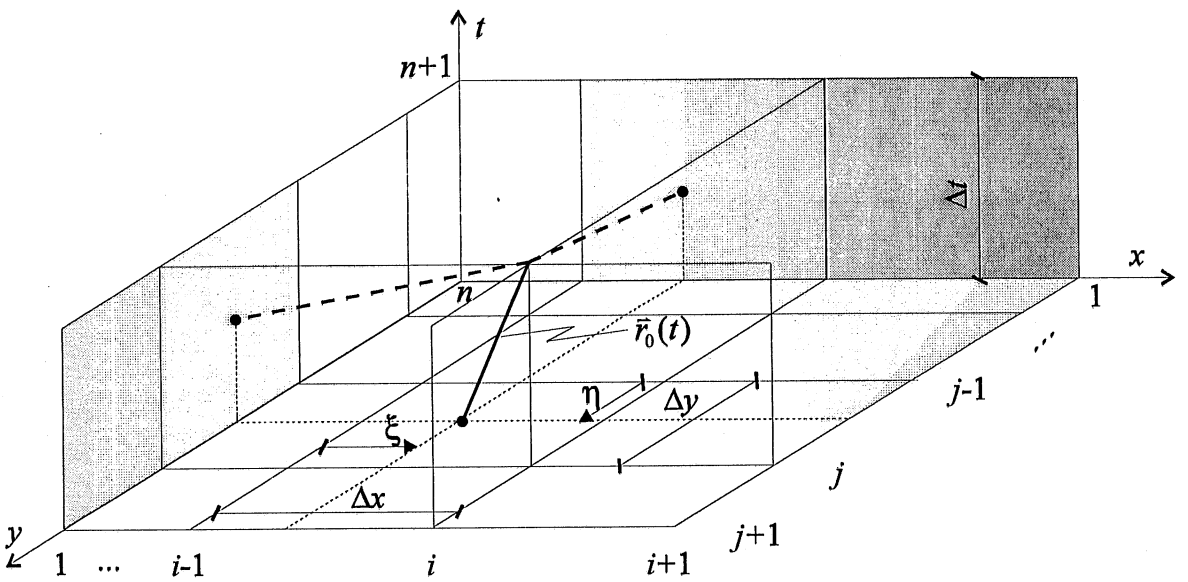

Figure 1. Two-dimensional convection computational grid

This procedure is carried out by marching through time for each computational point, starting from the initial condition specified in the problem. The method is thus explicit in nature, allowing the calculation of concentrations at time level $n+1$ using only the known concentration field at time level n.

The first step in the advection model is to determine the foot of a trajectory $\vec{r}_0(t)$ integrating Eq. (14). If the velocity field $\vec{u}(\vec{r}_0, t)$ is specified in the domain, the integration can be carried out by the modified Euler method

$$\vec{r}_0^{\,*}(t^n) = \vec{r}_0(t^{n+1}) - \vec{u}(\vec{r}_0, t^{n+1})\Delta t$$

$$\vec{r}_0(t^n) = \vec{r}_0(t^{n+1}) - \frac{\vec{u}(\vec{r}_0, t^{n+1}) + \vec{u}(\vec{r}_0^{\,*}, t^n)}{2}\Delta t \qquad (15)$$

where $\vec{u}(\vec{r}_0^{\,*}, t^n)$ is interpolated in the domain between the pertinent grid points.

After the foot of the characteristic line has been located the calculation proceeds with suitable interpolation of the concentration between grid points in the vicinity of the foot. This is where the power of the Holly-Preissmann method comes into play. If we assume a linear variation of $C(\vec{r}_0, t^n)$ between grid points, the result is a simple Upwind method which introduces severe numerical diffusion for small Courant numbers (Holly,

1975). The numerical damping can be reduced by using a more refined interpolation. One can increase the number of grid points, leading to polynomial interpolation, but this technique has proved to introduce numerical dispersion (phase shift). The Holly-Preissmann method, by contrast, is based on the use of the derivatives (gradients) of the concentration at the same grid points, rather than increasing their number. The resulting two-point higher order method uses information at only adjacent grid points. This method can be outlined as follows.

Consider the polynomial interpolation in the subsequent form

$$\tilde{f}(\xi,\eta)=\sum_{k=0}^{3}\sum_{l=0}^{3}a_{kl}\xi^{k}\eta^{l} \tag{16}$$

where ξ and η are the local foot coordinates of the characteristic line through point (i,j), as shown in Fig. 1. They are constrained as follows

$$0\le\xi=\frac{x-x_{i-1}}{\Delta x}\le 1,\quad 0\le\eta=\frac{y-y_{j-1}}{\Delta y}\le 1 \tag{17}$$

The coefficients a_{kl} of the interpolation function (16) can be determined from the known conditions at four bordering grid points

$$\begin{aligned}
&\tilde{f}(0,0)=C_{i-1,j-1},\quad \tilde{f}(1,0)=C_{i,j-1},\quad \tilde{f}(0,1)=C_{i-1,j},\quad \tilde{f}(1,1)=C_{i,j}\\
&\tilde{f}_{\xi}(0,0)=C_{\xi_{i-1,j-1}},\quad \tilde{f}_{\xi}(1,0)=C_{\xi_{i,j-1}},\quad \tilde{f}_{\xi}(0,1)=C_{\xi_{i-1,j}},\quad \tilde{f}_{\xi}(1,1)=C_{\xi_{i,j}}\\
&\tilde{f}_{\eta}(0,0)=C_{\eta_{i-1,j-1}},\quad \tilde{f}_{\eta}(1,0)=C_{\eta_{i,j-1}},\quad \tilde{f}_{\eta}(0,1)=C_{\eta_{i-1,j}},\quad \tilde{f}_{\eta}(1,1)=C_{\eta_{i,j}}\\
&\tilde{f}_{\xi\eta}(0,0)=C_{\xi\eta_{i-1,j-1}},\ \tilde{f}_{\xi\eta}(1,0)=C_{\xi\eta_{i,j-1}},\ \tilde{f}_{\xi\eta}(0,1)=C_{\xi\eta_{i-1,j}},\ \tilde{f}_{\xi\eta}(1,1)=C_{\xi\eta_{i,j}}
\end{aligned} \tag{18}$$

where

$$C_{\xi}=\frac{\partial C}{\partial x}\Delta x,\quad C_{\eta}=\frac{\partial C}{\partial y}\Delta y,\quad C_{\xi\eta}=\frac{\partial^{2}C}{\partial x\partial y}\Delta x\Delta y$$

After applying the conditions (18) and rearranging the polynomial function (16), it is possible to cast the interpolation expression in terms of the known concentration and its derivatives at adjacent grid points. The result is

$$\tilde{f}(\alpha_i) = \alpha_1 C_{i-1,j-1} + \alpha_2 C_{i-1,j} + \alpha_3 C_{i,j-1} + \alpha_4 C_{i,j} +$$

$$\alpha_5 C_{\xi_{i-1,j-1}} + \alpha_6 C_{\xi_{i-1,j}} + \alpha_7 C_{\xi_{i,j-1}} + \alpha_8 C_{\xi_{i,j}} + \tag{19}$$

$$\alpha_9 C_{\eta_{i-1,j-1}} + \alpha_{10} C_{\eta_{i-1,j}} + \alpha_{11} C_{\eta_{i,j-1}} + \alpha_{12} C_{\eta_{i,j}} +$$

where

$$\alpha_1 = F_1(\xi)F_1(\eta), \quad \alpha_2 = F_1(\xi)F_2(\eta), \quad \alpha_3 = F_2(\xi)F_1(\eta), \quad \alpha_4 = F_2(\xi)F_2(\eta)$$

$$\alpha_5 = F_3(\xi)F_1(\eta), \quad \alpha_6 = F_3(\xi)F_2(\eta), \quad \alpha_7 = F_4(\xi)F_1(\eta), \quad \alpha_8 = F_4(\xi)F_2(\eta) \tag{20}$$

$$\alpha_9 = F_1(\xi)F_3(\eta), \quad \alpha_{10} = F_1(\xi)F_4(\eta), \quad \alpha_{11} = F_2(\xi)F_3(\eta), \quad \alpha_{12} = F_2(\xi)F_4(\eta)$$

and

$$F_1(\lambda) = 1 - 3\lambda^2 + 2\lambda^3$$

$$F_2(\lambda) = \lambda^2(3 - 2\lambda)$$

$$F_3(\lambda) = \lambda(1 - \lambda)^2 \tag{21}$$

$$F_4(\lambda) = \lambda^2(\lambda - 1)$$

The number of relevant coefficients in Eq. (19) is reduced to 12, for it can be shown that the second derivatives with respect to x and y ($C_{\xi\eta}$) have little influence on the solution and may be neglected.

When the interpolation on the previous time level has been completed, the unknown concentration at point (i, j) is simply

$$C(\vec{r}_0, t^{n+1}) = C(\vec{r}_0, t^n) = \tilde{f}(\alpha_i) \tag{22}$$

To proceed with the method it is also necessary to know the gradients of the concentration at each grid point. These gradients are propagated along the trajectory $\vec{r}_0(t)$ in accordance with their substantial derivative

$$\frac{D}{Dt}(\nabla C) \equiv \frac{\partial}{\partial t}(\nabla C) + (\vec{u} \cdot \nabla)\nabla C = -\nabla \vec{u} \cdot \nabla C \tag{23}$$

which is obtained by differentiating the advection equation (11) with respect to $\vec{r}$.

Equation (23) is then integrated along the characteristic line $\vec{r}_0(t)$ by the modified Euler method to yield

$$\nabla C(\vec{r}_0^{\,*}, t^{n+1}) = \nabla C(\vec{r}_0, t^n) - \nabla \vec{u}(\vec{r}_0, t^n) \cdot \nabla C(\vec{r}_0, t^n) \Delta t$$

$$\nabla C(\vec{r}_0, t^{n+1}) = \nabla C(\vec{r}_0, t^n) - \frac{\nabla \vec{u}(\vec{r}_0, t^n) \cdot \nabla C(\vec{r}_0, t^n) + \nabla \vec{u}(\vec{r}_0^{\,*}, t^{n+1}) \cdot \nabla C(\vec{r}_0^{\,*}, t^{n+1}) +}{2} \Delta t \tag{24}$$

The velocity gradients $\nabla \vec{u}(\vec{r}_0, t^n)$ at the foot of the characteristic line are calculated by suitable finite differences between the adjacent grid points. It is also necessary to know the concentration gradients $\nabla C(\vec{r}_0, t^n)$ which will be calculated by differentiation of the interpolation function (19)

$$\frac{\partial C(\vec{r}_0, t^n)}{\partial x} = \frac{1}{\Delta x} \frac{\partial \tilde{f}(\alpha_i)}{\partial \xi} = \frac{\tilde{f}(\beta_i)}{\Delta x} \tag{25}$$

$$\frac{\partial C(\vec{r}_0, t^n)}{\partial y} = \frac{1}{\Delta y} \frac{\partial \tilde{f}(\alpha_i)}{\partial \eta} = \frac{\tilde{f}(\gamma_i)}{\Delta y} \tag{26}$$

where

$$\begin{aligned} &\beta_1 = G_1(\xi)F_1(\eta), \quad \beta_2 = G_1(\xi)F_2(\eta), \quad \beta_3 = -\beta_1 \quad \beta_4 = -\beta_2 \\ &\beta_5 = G_2(\xi)F_1(\eta), \quad \beta_6 = G_2(\xi)F_2(\eta), \quad \beta_7 = G_3(\xi)F_1(\eta), \quad \beta_8 = G_3(\xi)F_2(\eta) \\ &\beta_9 = G_1(\xi)F_3(\eta), \quad \beta_{10} = G_1(\xi)F_4(\eta), \quad \beta_{11} = -\beta_9, \quad \beta_{12} = -\beta_{10} \end{aligned} \tag{27}$$

$$\begin{aligned} &\gamma_1 = F_1(\xi)G_1(\eta), \ \gamma_2 = -\gamma_1, \quad \gamma_3 = F_2(\xi)G_1(\eta), \ \gamma_4 = -\gamma_3 \\ &\gamma_5 = F_3(\xi)G_1(\eta), \ \gamma_6 = -\gamma_5, \quad \gamma_7 = F_4(\xi)G_1(\eta), \ \gamma_8 = -\gamma_7 \\ &\gamma_9 = F_1(\xi)G_2(\eta), \ \gamma_{10} = F_1(\xi)G_3(\eta), \ \gamma_{11} = F_2(\xi)G_2(\eta), \ \gamma_{12} = F_2(\xi)G_3(\eta) \end{aligned} \tag{28}$$

$$\begin{aligned} &G_1(\lambda) = 6\lambda(\lambda - 1) \\ &G_2(\lambda) = (\lambda - 1)(3\lambda - 1) \\ &G_3(\lambda) = \lambda(3\lambda - 2) \end{aligned} \tag{29}$$

The aforementioned procedure refers to a case when the foot of the characteristic line $\vec{r}_0(t)$ intersects the computational domain (at the previous time level) in the first quadrant with respect to grid point (i, j), i.e. when Courant numbers in both x and

y direction are less than 1 ($|u|\Delta t/\Delta x \le 1$, $|v|\Delta t/\Delta y \le 1$) and $u \ge 0, v \ge 0$. If this is not the case, nothing is principally changed in the algorithm, except that the local coordinates ξ and η are then evaluated in the quadrant that the foot of the trajectory belongs to. If the foot hits the boundary planes ($i = 1$ or $j = 1$, dashed lines in Fig. 1) the appropriate concentration and its gradients at the foot are evaluated by suitable interpolation and differentiation between the known values at the boundaries.

The method also requires as initial condition both the concentration and its gradients. Thus it is important to estimate the sensitivity of the scheme to errors in the initial specification of the concentration gradients. It was actually found (Holly and Preissmann, 1977) that the inconsistencies between the concentration and its gradients, introduced by any reasonably accurate technique for the estimation of initial derivatives from given concentrations, will have a minor influence on the results. Here the concentration gradients at time zero are estimated by the central difference approximation.

The above method has order of accuracy $O(\Delta x^4)$. It can be shown that it is unconditionally stable with the amplification and phase shift factor very close to one (Stefanovic, 1995). This implies that the numerical dissipation and dispersion are not introduced to the extent that they would compromise the final result. Furthermore, being explicit in nature, the procedure does not require solving a system of algebraic equations, so that most of the computational time is spent on polynomial interpolation. The evaluation of the coefficients in the interpolation function (19) may seem fairly time-consuming but a closer look at their form reveals that they consist of a few block functions (21) and (29), which are calculated at the beginning and subsequently used in expressions (20), (27) and (28). As a result, this is an accurate and sufficiently efficient computational procedure for solution to pure advection. It is superior to standard finite difference methods such as Upwind, Runge-Kutta, Lax-Wendroff or MacCormack (Stefanovic, 1995).

It is worth mentioning that the fractional-step formulation is not restricted to any particular numerical methods for the solution of the advection equation and the choice of the numerical scheme is a preference of the user. However, the method proposed here has proven to give superior results compared to other methods for advection dominated flows and will be chosen as a first computational step in the coupled advection-diffusion simulation.

IV. SOLUTION OF THE DIFFUSION EQUATION WITH COMBINED SOURCE/SINK TERM

In order to model a diffusion (conduction) process, with adequate accuracy and stability, it is usually not necessary to make a considerable numerical effort, owing to the parabolic nature of the diffusion equation, here extended by the source terms aC and b

$$\frac{\partial C}{\partial t} = \frac{\partial}{\partial x}\left(\alpha_x \frac{\partial C}{\partial x}\right) + \frac{\partial}{\partial y}\left(\alpha_y \frac{\partial C}{\partial y}\right) + aC + b \tag{30}$$

Therefore, the diffusion terms in Eq. (30) are customarily discretized by a class of implicit central space approximations. The approach adopted here is a form of alternating directions (ADI) scheme (Hirsch, 1988), which is further modified for the general case of variable diffusion coefficients, nonuniform computational grid and presence of the source term.

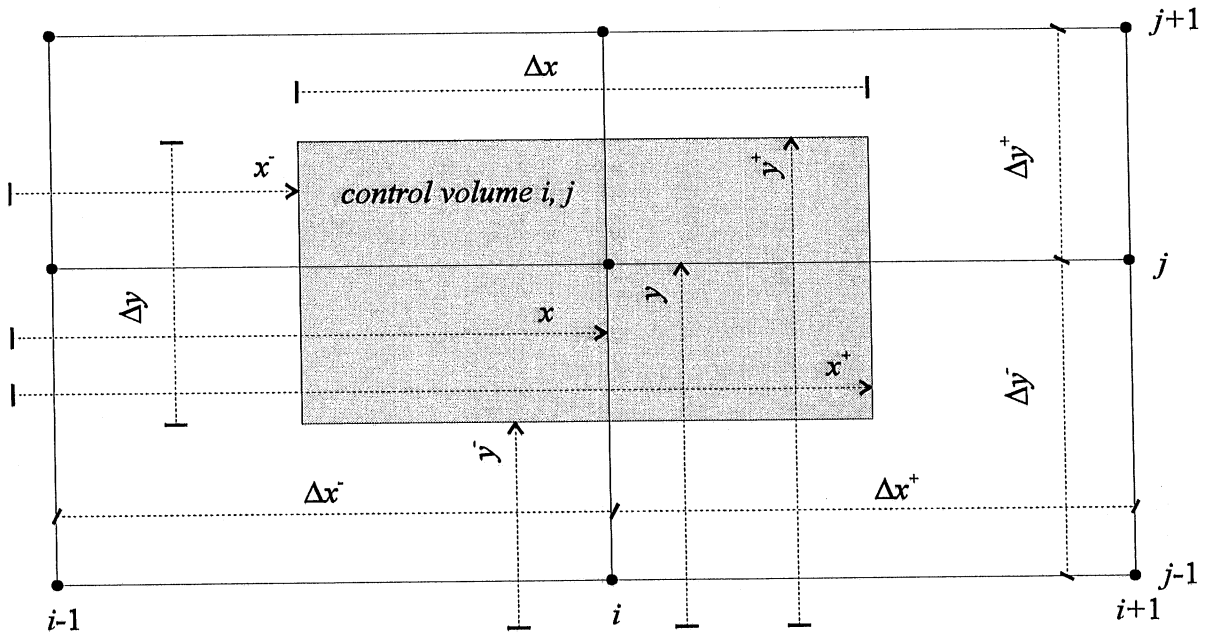

Figure 2. Two-dimensional diffusion computational grid

The numerical scheme for the computational grid in Fig. 2. will be constructed as follows

$$\dot{C} - C^n = \theta\Lambda_x(\dot{C}) + (1-\theta)\Lambda_y(C^n) + \theta S(\dot{C}) + (1-\theta)S(C^n)$$

$$C^{n+1} - \dot{C} = (1-\theta)\Lambda_x(\dot{C}) + \theta\Lambda_y(C^{n+1}) + \theta S(\dot{C}) + (1-\theta)S(C^{n+1}) \tag{31}$$

where

$\theta \in [0,1]$ = weighting factor between time levels n and $n+1$

$\vec{\Lambda}(C) \equiv (\Lambda_x, \Lambda_y) = (\vec{F}^+\vec{\delta}^+ - \vec{F}^-\vec{\delta}^-)C$ - diffusion operator vector

$S(C) = 0.5a\Delta tC + 0.5b\Delta t$ = source operator

$\vec{\delta}^+(C) \equiv (\delta_x^+, \delta_y^+) = C(\vec{r} + \Delta\vec{r}^+) - C(\vec{r})$ = forward difference operator vector

$\vec{\delta}^-(C) \equiv (\delta_x^-, \delta_y^-) = C(\vec{r}) - C(\vec{r} - \Delta\vec{r}^-)$ = backward difference operator vector

$\vec{F}^+ \equiv (F_x^+, F_y^+) = \dfrac{\vec{\alpha}^+\Delta t}{\Delta\vec{r}^+\Delta\vec{r}}$ = Fourier number vector at $\vec{r}^+$

$\vec{F}^- \equiv (F_x^-, F_y^-) = \dfrac{\vec{\alpha}^-\Delta t}{\Delta\vec{r}^-\Delta\vec{r}}$ = Fourier number vector at $\vec{r}^-$

$\vec{\alpha}^+ \equiv (\alpha_x^+, \alpha_y^+)$ = diffusion coefficient vector at $\vec{r}^+$

$\vec{\alpha}^- \equiv (\alpha_x^-, \alpha_y^-)$ = diffusion coefficient vector at $\vec{r}^-$

$\vec{r} \equiv (x, y)$, $\Delta\vec{r} \equiv (\Delta x, \Delta y)$, $\Delta\vec{r}^+ \equiv (\Delta x^+, \Delta y^+)$, $\Delta\vec{r}^- \equiv (\Delta x^-, \Delta y^-)$

The ADI scheme employed is composed of two steps, as specified in Eq. (31). In the first step, the operator Λ_x is calculated implicitly, and the operator Λ_y is calculated explicitly. In the second step the opposite is done. Subsequently, the whole procedure is repeated. In this way, the integration in each direction is carried out alternately by an explicit, then by an implicit scheme, and the error increase in the explicit phase is balanced by the error decrease in the implicit one, so that the stability is maintained (Yanenko, 1971). It can be shown by the Von Neumann stability analysis, that in the absence of the first-order source term ($a = 0$) or with the first-order sink ($a < 0$), the scheme is unconditionally stable in the range $0.5 \le \theta \le 1$. In the presence of a first-order source ($a > 0$) the method is stable for $a\Delta t < (a\Delta t)_{\max} = 8 \cdot \min\left(F_x, F_y\right)$. This condition is acceptable in most natural environments, with sufficiently large turbulent diffusion and the first-order rate coefficients being relatively small. If the requirement cannot be satisfied (by refining the space grid), the recommended practice is to establish an iterative

procedure by setting $a = 0$ and $b^* = aC^* + b$. Here b^* is a nominally zero-order source term in the current iteration cycle, corresponding to the previous iteration concentration C^*. The iteration technique is also employed in the presence of a nonlinear source term, which can be linearized by the Newton-Raphson or some other suitable technique and then solved in a linear framework. In the latter case, it is vital to keep a negative (if possible, if not resort to b^*) so that instabilities and physically unrealistic solutions do not arise (Patankar, 1980).

The accuracy of the method is $O(\Delta t, \Delta x^2)$, except for $\theta = 0.5$, when it is also second-order accurate in time. However, the last case is occasionally prone to physically unrealistic oscillations, so that the weighting factor $\theta > 0.5$ is introduced as a partial remedy for undesirable numerical phenomena (for larger θ the method is more diffusive). Along the same lines is the concern about magnitude of $(a\Delta t)_{\max}$, ensuring that the coefficients in the algebraic equations are all positive, which is a guarantee for physical realism of numerical results (Patankar, 1980).

The immediate advantage of the method presented is elimination of the pentadiagonal matrix sweep and its replacement by two tridiagonal sweeps (efficiently solved by a Thomas algorithm, Anderson et. al., 1984), which considerably reduces the extent of the calculations. The difference equation (31) leads to two tridiagonal algebraic systems

$$a_i' C'_{i-1,j} + b_i' C'_{i,j} + c_i' C'_{i+1,j} = f_i'(C''), \quad i = 2,\ldots,N-1$$

$$a_j^{n+1} C_{i,j-1}^{n+1} + b_j^{n+1} C_{i,j}^{n+1} + c_j^{n+1} C_{i,j+1}^{n+1} = f_j^{n+1}(C'), \quad j = 2,\ldots,M-1 \tag{32}$$

In addition to the previous set of equations one needs boundary conditions specified at the upstream and downstream end (in x and y direction), as well as the initial concentration field. Boundary conditions are supplied either as known concentrations or concentration gradients. In the former case, the boundary concentrations are directly used to close the system (32), while in the latter it is necessary to write additional balance equations at the boundaries (half-control volume approach, Patankar, 1980).

V. NUMERICAL EXAMPLES

The accuracy and stability of the method are tested for different advection/diffusion scenarios applicable to modeling of mass and heat transfer in natural environments. Particular analytical solutions of the advection-diffusion equation have been used in order to examine the quality and sensitivity of the method to typical process parameters (Courant, Fourier and Peclet numbers), thus anticipating the model behavior in natural systems with diversified transport characteristics. The first two examples are intended to illustrate the capabilities of the model under conditions where one individual transport mechanism prevails over others. The last example gives the model results in a case which encompasses all components of two-dimensional advective/dispersive mixing.

1. Advection of a conservative substance

As a first example, suppose that at initial time ($t = 0$) the concentration of a conservative substance (no sources/sinks due to physical, chemical or biochemical degradation processes) is zero everywhere in the computational domain. The concentration is then suddenly raised to C_0 at both upstream boundaries ($x = 0, y = 0$) and is held at that value thereafter. The homogeneous concentration distribution along the boundaries is accepted and retained over time as the concentration front advances downstream. This case may illustrate the discharge of a residual material into two-dimensional environments, with negligible mixing of fluid particles in the longitudinal and lateral direction. Each element of fluid and its associated quality flows downstream in a unique and descrete fashion, with insignificant spreading of the step discharge. This condition is customarily referred to as a plug flow or maximum gradient system and is useful as a first approach in water quality analyses, especially for one-dimensional rivers and streams (Thomann et. al., 1987).

Because the substance is conservative, there is no change in concentration downstream from the discharge source. The numerical method should reflect this circumstance without introduction of significant numerical diffusion, which is otherwise not inherent to the process.

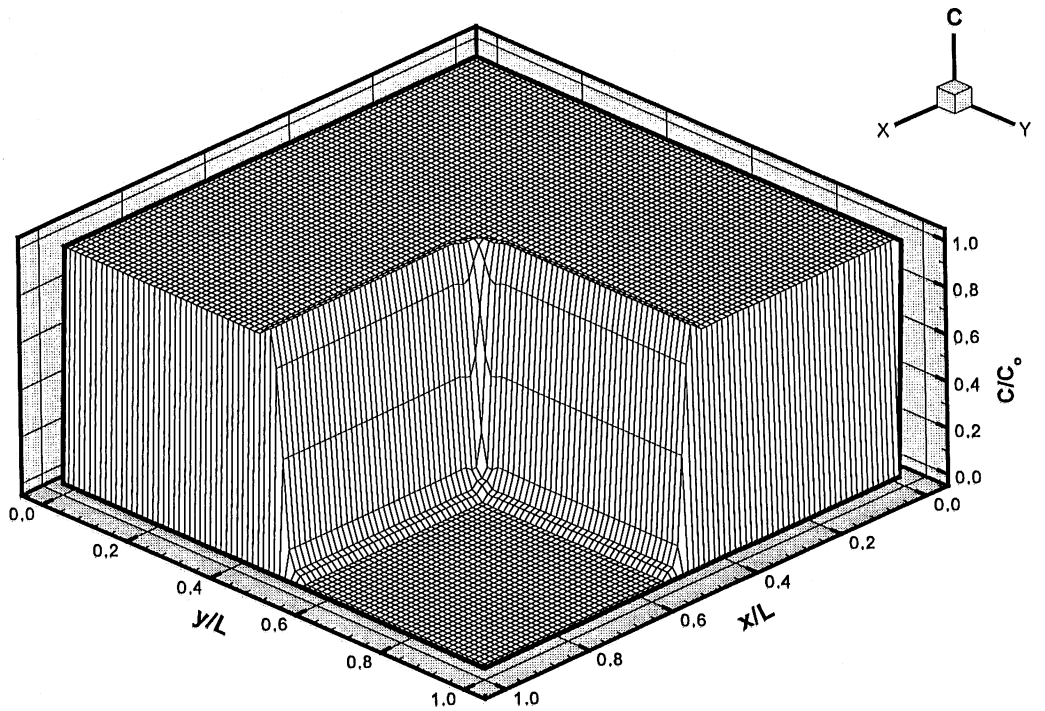

Figure 3. Advection of a conservative substance

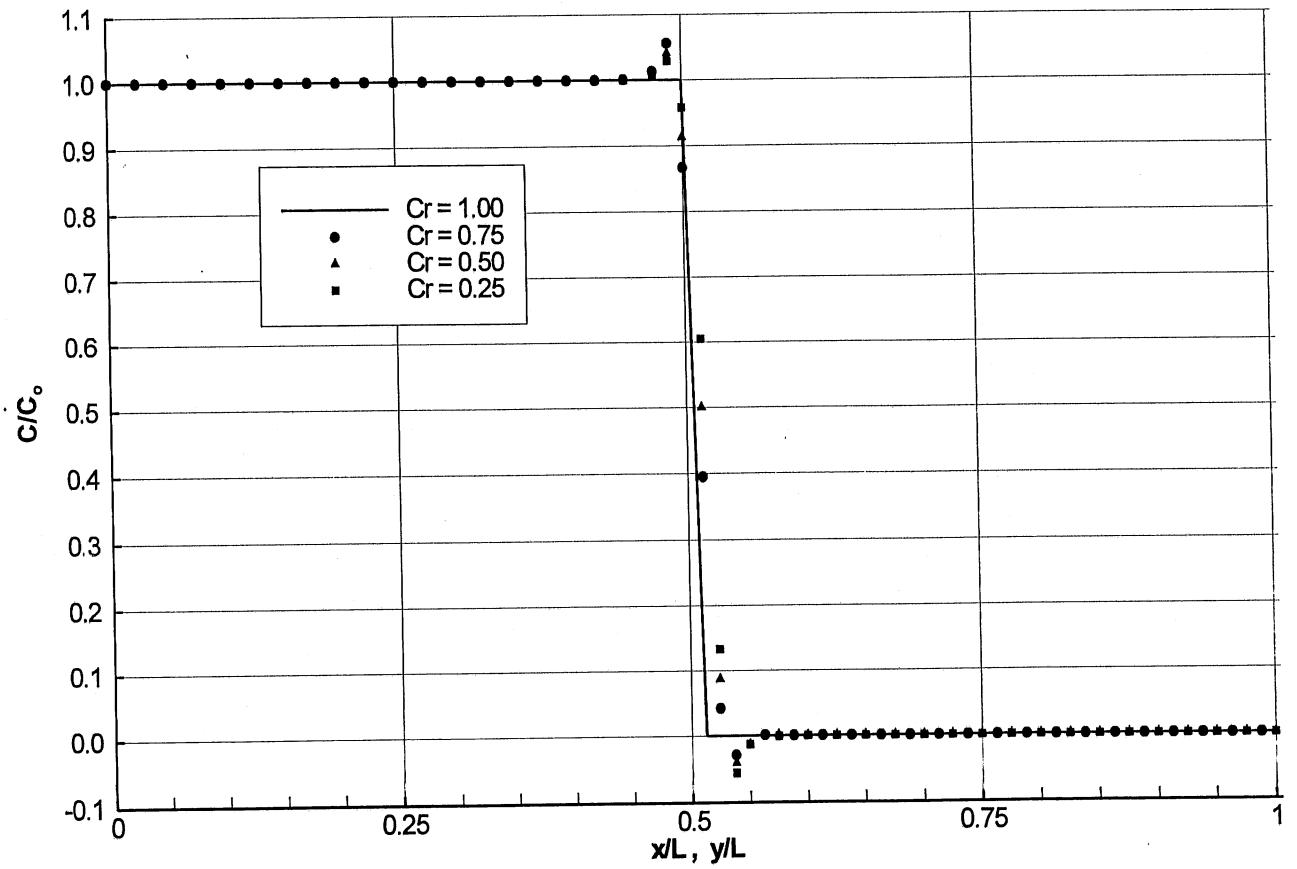

Figure 4. Advection of a conservative substance - cross sections

The simulation is carried out in a two-dimensional constant velocity field $u = v = 0.5$ m/s on an uniform grid of $\Delta x = \Delta y = 100$ m (Fig. 3). The boundary concentration is transported downstream for 2.8 hr ($50\Delta t$), which is a distance of 5 km in both space directions. The numerical results for a range of Courant numbers, along with the exact solution (Cr=1) are shown in Fig. 4, for two representative cross sections of Fig. 3.

The calculation results corroborate the formal accuracy of the Holly-Preissmann method for the entire range of Courant numbers. It should be noted that the exact solution is recovered for Cr=1 (or any integer number). Identical results are obtained for any other corresponding range of Courant numbers greater than one, as was previously explained.

The error in phase (celerity) is very small, while the error in amplitudes does not exceed 6%. These negative (as well as positive) oscillations, being the consequence of numerical dispersion, cannot be arbitrarily suppressed, for this would introduce additional mass into the model, which otherwise preserves the overall mass balance (Holly and Preissmann, 1977).

Similar results are obtained for anisotropic geometry and flow conditions in longitudinal and lateral direction, as well as for relatively large time steps Δt, implying that the two-dimensional advection model is reliable and can be used with engineering confidence under diversified circumstances that may be encountered in aquatic ecosystems.

2. Diffusion of a nonconservative substance

The hypothetical example outlined below is designed to verify the results of the second stage in the computational algorithm. It is assumed that the advection terms in the transport equation (1) are negligible, resulting in a purely dispersive system. This condition can illustrate a coastal regime of estuaries and bays with large estuary number ($a\alpha/u$), so that the transport of the nonconservative substance is mainly due to tidal mixing with no net flow (Thomann et. al., 1987).

The initial and upstream boundary conditions in a domain 2 x 2 km are the same ones used in the previous example. In contrast to pure advection, the diffusion equation (30) requires additional boundary conditions, specified at both downstream ends of the computational domain. Those are held at zero in the example under investigation. The first order reaction rate coefficient was set to $a = -2.5 \cdot 10^{-6}\ s^{-1}$, assuming the existence of a chemical reaction or biochemical degradation process in the system.

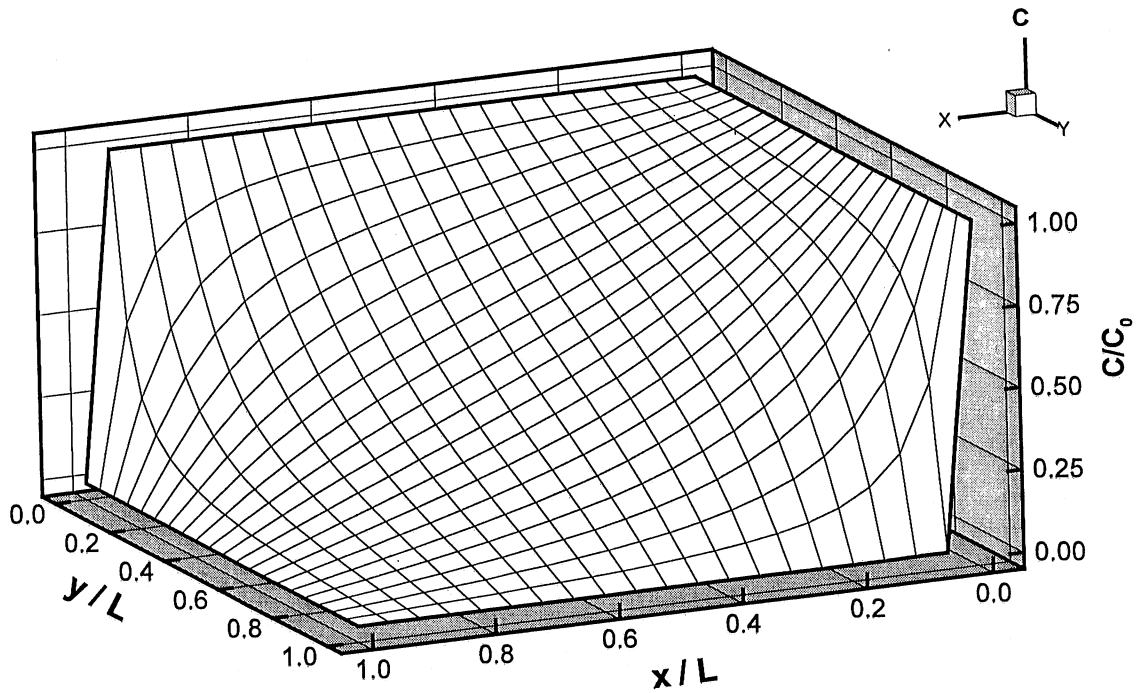

Figure 5. Diffusion of a nonconservative substance

The two-dimensional boundary value problem established above cannot be solved in a closed form, so the Fourier series technique (Tolstov, 1962) was employed to obtain the following exact solution

$$C(x,y,t)=\sum_{n=1}^{\infty}\sum_{m=1}^{\infty}A_{mn}\sin(\lambda_n x)\sin(\omega_m y)\left[1-e^{-(p_{mn}+a)t}\right] \tag{33}$$

where

$$A_{mn}=\frac{q_{mn}}{p_{mn}},$$

$$p_{mn}=\alpha_x\lambda_n^2+\alpha_y\omega_m^2,$$

$$q_{mn}=\frac{4C_o\alpha_x n}{l_x^2 m}\left[1-(-1)^m\right]+\frac{4C_o\alpha_y m}{l_y^2 n}\left[1-(-1)^n\right]$$

$$\lambda_n=n\pi/l_x,\ \ \omega_m=m\pi/l_y\ \ l_x=l_y=2000\ \text{m}$$

Numerical results compared to the previous exact solution are shown in Fig. 5 and 6. The characteristic nondimensional parameter for diffusion (conduction) problems is Fourier number, which was varied in the range from 0.1 to 10. The agreement between numerical results and exact solution is very good, with maximum error less than 2%, in the range of Fourier numbers typical for environmental studies.

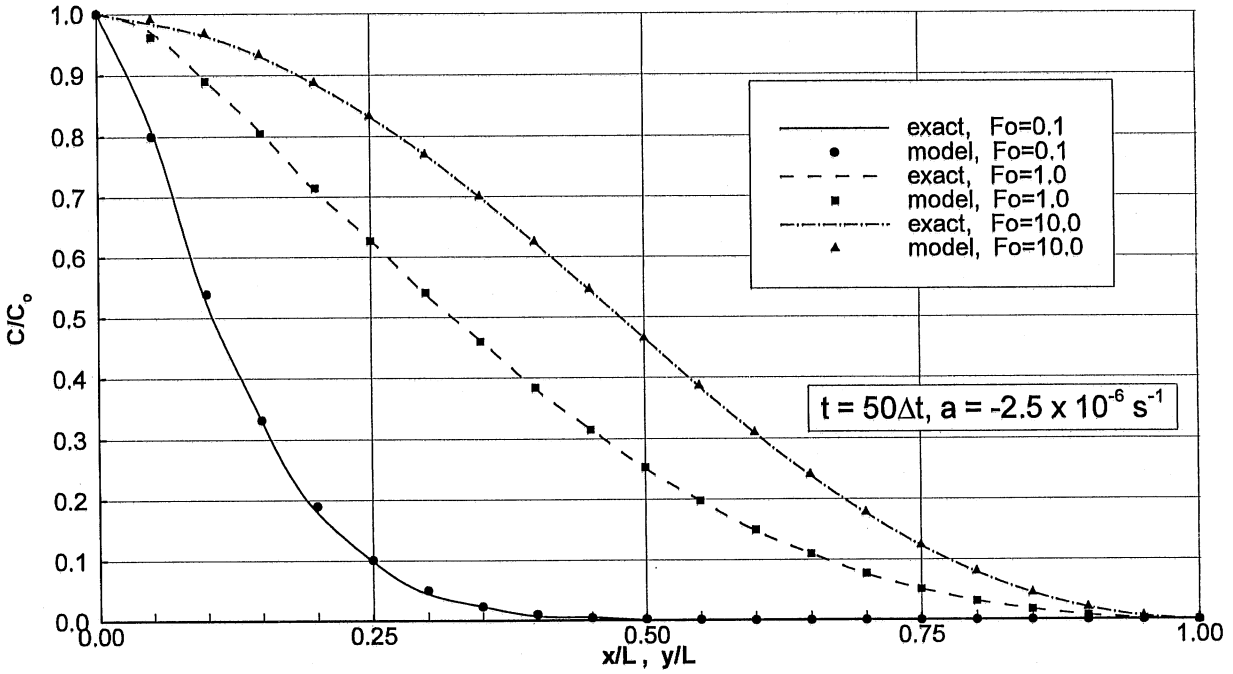

Figure 6. Diffusion of a nonconservative substance - diagonal cross section

It is interesting to note that, for this particular boundary value problem, the exact solution was converging very slowly, requiring a considerable number of double Fourier-series harmonics, which made the computational time an order of magnitude larger than the one used for numerical simulation. This clearly demonstrates the power of numerical methods.

3. Advection-diffusion of a nonconservative substance

This example represents a common situation in environmental flows. A nonconservative substance is being advected (by the mean motion) downstream from discharge, while at the same time there is mixing due to the existence of ambient velocity gradients and geometric irregularities.

We will examine an instantaneous spill of material with an initial mass M, released at some point in an infinite domain. The analytical solution is given by the Gaussian probability density function (Fischer et. al., 1979)

$$C(x,y,t) = \frac{M}{4\pi t\sqrt{\alpha_x \alpha_y}} \exp\left[-\frac{(x-ut)^2}{4\alpha_x t} - \frac{(y-vt)^2}{4\alpha_y t} + at \right] \tag{34}$$

Numerical model tests have been conducted for a first-order source strength $a = 2 \cdot 10^{-7}\ s^{-1}$ in a long and wide (8 x 8 km) constant velocity field of $u = v = 0.1$ m/s, with a regular grid spacing $\Delta x = \Delta y = 100$ m. The initial Gaussian distribution (34) is defined by the spread $\sigma = \sqrt{2\alpha t}$ of 179, 253, 358, 800 m, corresponding to Peclet numbers of 2000, 1000, 500, 100 respectively. These initial distributions are advected downstream for 8.3 hrs ($50\,\Delta t$), covering a distance of 3 km in both x and y direction (Fig. 7).

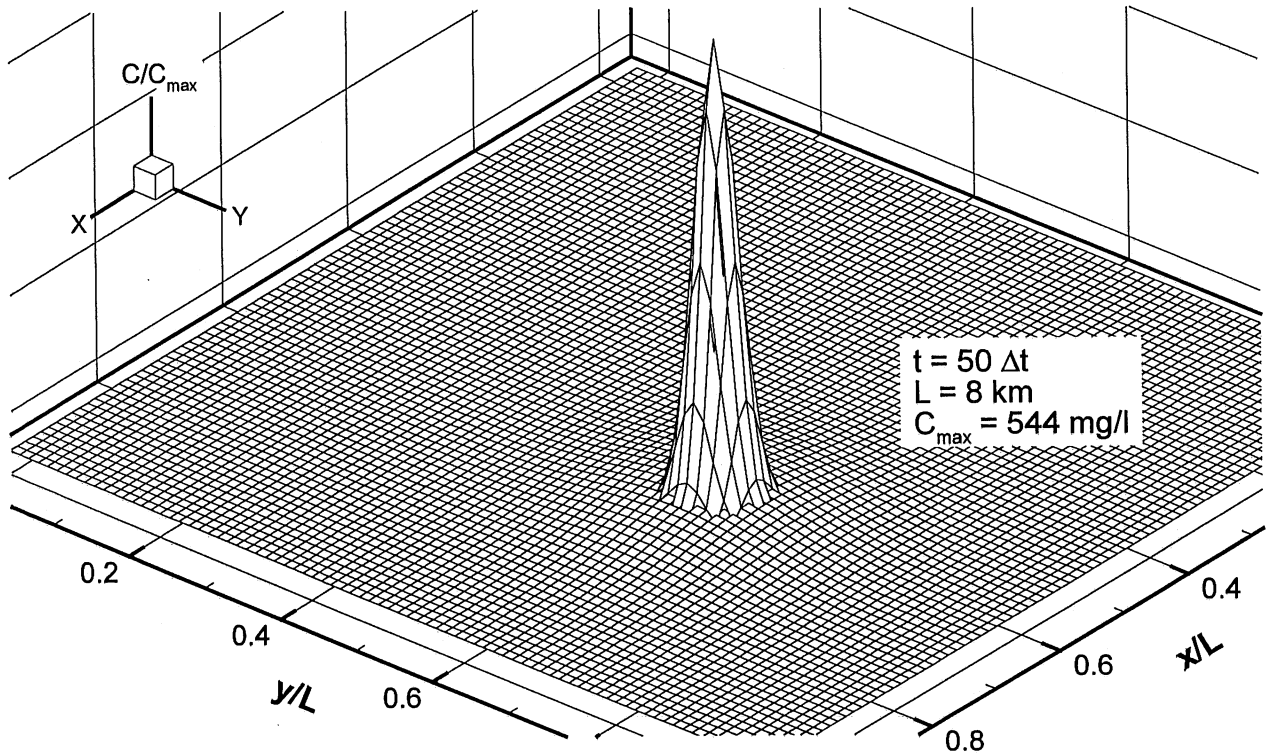

Figure 7. Advection-diffusion of a nonconservative substance

The tests are carried out for a Courant number $= 0.6$, since the method is not paricularly sensitive to the advection parameter. This is also true if Courant numbers are different in x and y directions.

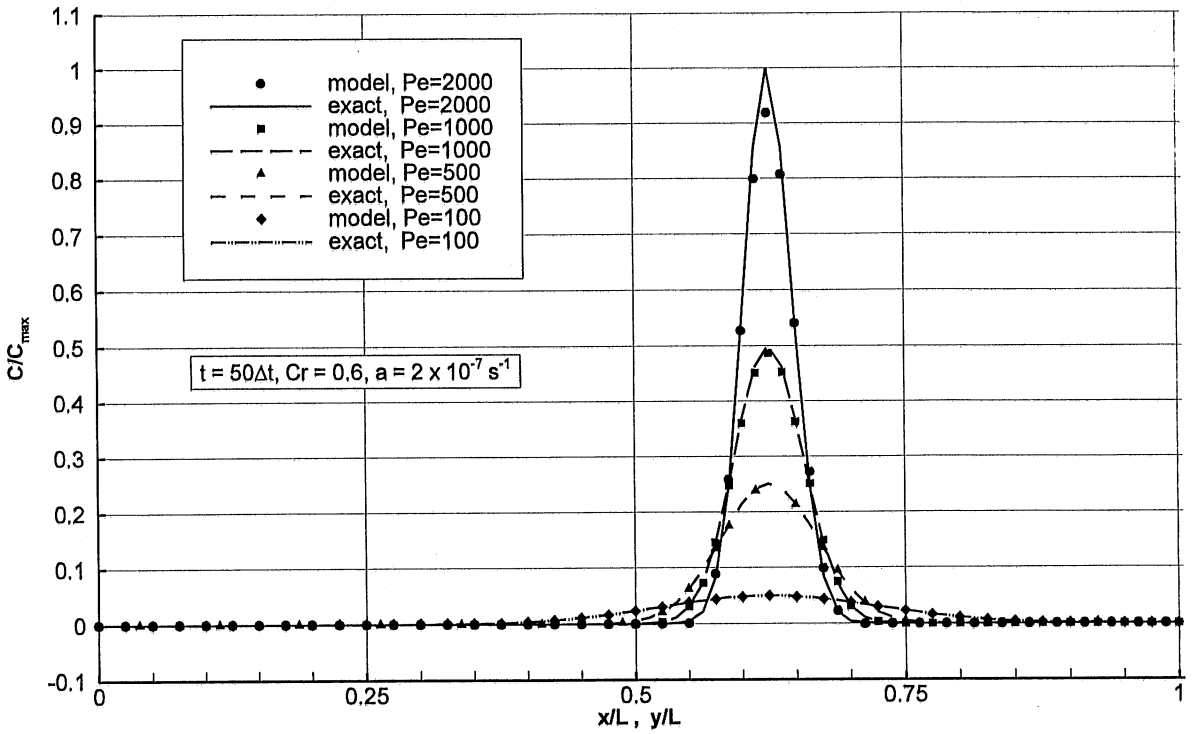

Figure 8. Advection-diffusion - cross sections of normalized concentration

Numerical results, along with the exact solution (34), are displayed in a two-dimensional plot (Fig. 8), representing two (symmetrical) x and y cross-sections in Fig. 7. It can be seen that the agreement between numerical and analytical results is fairly good, even for the most advection-dominated case corresponding to $\mathrm{Pe} = 2000$. The maximal error in amplitude (at the peak) is less then 8% for this drastic case, while the error in phase is negligible. Slight asymmetry in the results for the left and right branch of the Gaussian distribution are noticed (mainly for very high Peclet numbers), since Courant number used here suggests that the foot of the characteristic line is closer to the upstream grid points.

It is also useful to study the behavior of the numerical method for anisotropic conditions in longitudinal and lateral directions, representative of natural systems where advection/diffusion processes in one spatial direction are more pronounced than in the other. This is done be setting different Pe numbers in x and y direction, with more spreading in the y direction (Fig. 9). Numerical results for two representative x and y cross-sections are shown in Figure **10**. One can again see fairly good agreement with the exact solution.

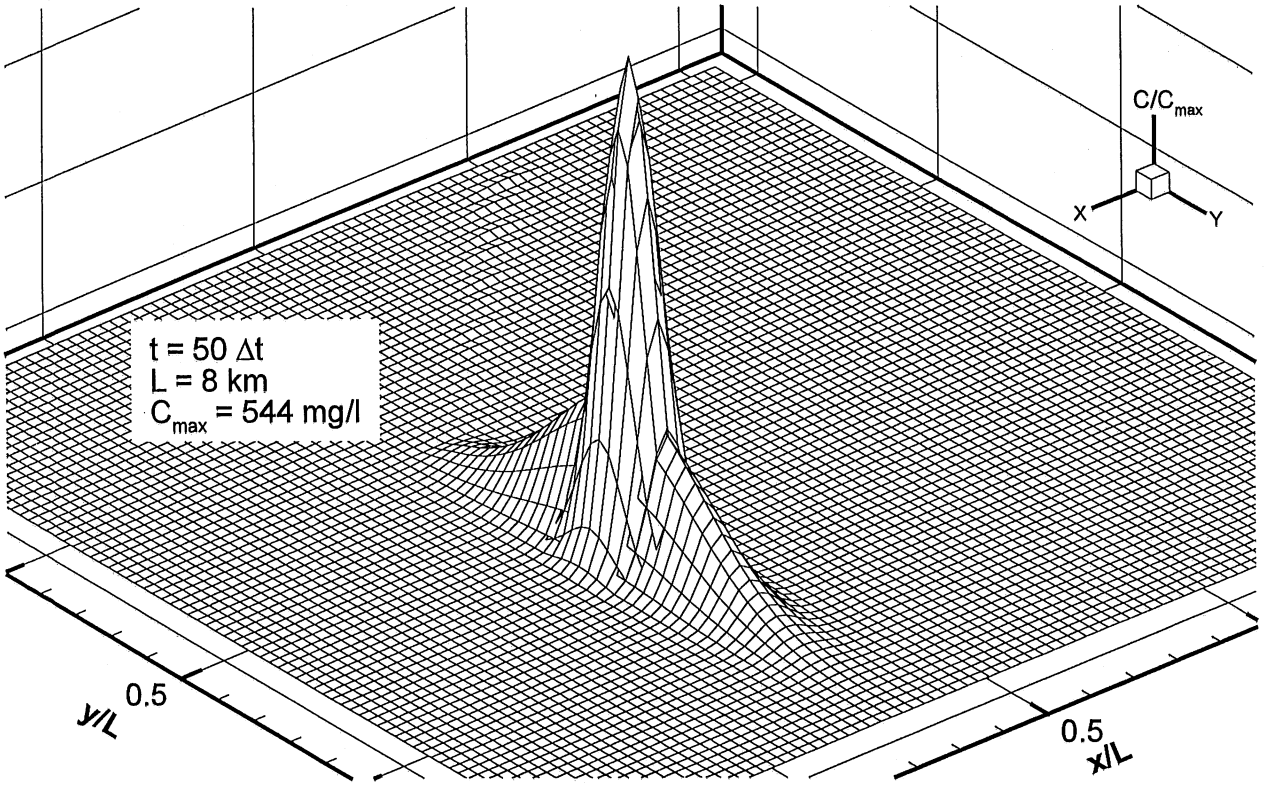

Figure 9. Anisotropic advection-diffusion of a nonconservative substance

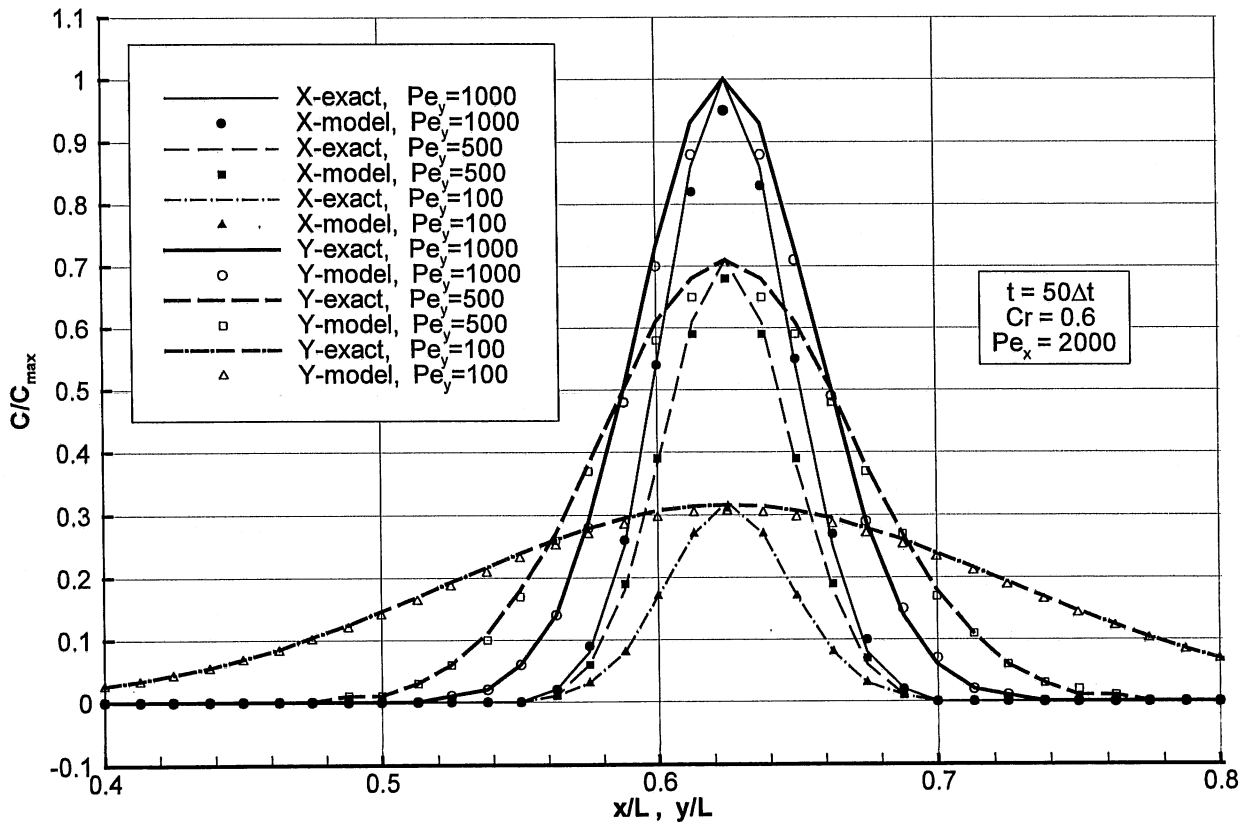

Figure 10. Anisotropic advection-diffusion - cross sections of concentration

VI. CONCLUSIONS

The purpose of this paper was to present a numerical model for the calculation of two-dimensional advection, diffusion and kinetic transformation processes, in the general framework of nonconstant velocities, variable diffusion coefficients and nonuniform computational grid. The model belongs to the class of finite difference methods and is based on the fractional step approach, which separates advection and diffusion/source processes at each time level.

As a first computational step in the algorithm, the model calculates pure advection by the Holly-Preissmann (1977) method. This method has proven to be accurate and stable, with considerably less numerical diffusion than other standard finite difference methods, over the entire range of Courant numbers. Being explicit, it is also not particularly time consuming.

The second step in the algorithm deals with diffusion and kinetic transformation processes. In this paper a modification and extension of the implicit alternate-direction method for calculation of pure diffusion is proposed. It takes into account first and zero-order source/sink terms, as a nominally linear framework for more complex nonlinear sources or sinks. The numerical scheme is accurate, stable and very efficient, for it eliminates the need to solve a pentadiagonal algebraic system, replacing it with two tridiagonal ones.

Comprehensive testing of the method for constant process parameters shows that numerical separation of differential operators does not induce significant errors and that physical realism of the results, for coupled two-dimensional transport, is ensured. This implies that the advantages of each particular scheme are preserved in the overall algorithm, for a wide range of Peclet numbers. Consequently, the numerical model proposed in this paper gives enough confidence for implementation in natural systems with nonconstant transport and geometric characteristics. It is especially appropriate for use in the study of coastal processes and incidental spills in aquatic environments with relatively large natural advection.

REFERENCES

Anderson, D. A., Tannehill, J. C., Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Corporation, 1984.

Fischer, H. B., List, E. J., Koh, R. C. Y., Imberger, J., and Brooks, N. H., *Mixing in Inland and Coastal Waters,* Academic Press, 1979.

Hirsch, C., *Numerical Computation of Internal and External Flows*, John Wiley & Sons, 1988.

Holly, F. M., Jr., and Preissmann A., Accurate Calculation of Transport in Two Dimensions, *J. Hydraul. Div. Proc. Am. Soc. Civ. Eng.*, Vol. 103, 1259-1277, 1977.

Holly, F. M., Jr., Two-Dimensional Mass Dispersion in Rivers, *Hydrology Paper No 78,* Colorado State University, Fort Collins, Colo., 1975.

Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publishing Co., 1980.

Stefanovic, D. L., One-Dimensional Heat Transfer Model in Open Channel Flow, Master thesis, School of Civil Engineering, University of Belgrade, 1995.

Thomann, R. V., and Mueller, J. A., *Principles of Surface Water Quality Modeling and Control*, HarperCollinsPublishers, NY, 1987.

Tolstov, G. P., *Fourier Series*, Englewood Cliffs, N.J., Prentice-Hall, 1962.

Yanenko, N. N., *The Method of Fractional Steps*, Springler-Verlag, Berlin, NY, 1971.

APPENDIX

List of main codes and subroutines written in FORTRAN 90:

T2.FOR - main code

ADV.FOR - calculation of the advection step using the method of characteristics with the Holly-Preissmann bicubic interpolation

CHAR.FOR - determination of characteristic lines by numerical integration over the known velocity field

HP4.FOR - bicubic interpolation between four internal grid points

BOUND.FOR - searching procedure to inspect whether particular characteristic line intersects one of four boundary planes

BGRAD.FOR - calculation of boundary gradients by the use of pure advection equation

CROSSDER.FOR - calculation of concentration cross-derivatives

FDER2.FOR - central difference computation of an 2-D array's first derivatives

FLINT.FOR - 1-D linear interpolation between two points

FLINT2.FOR - 2-D linear interpolation between four internal grid points

FLINT22.FOR - 2-D linear interpolation between four boundary points

DIFF.FOR - calculation of the diffusion-reaction step by the alternate direction implicit method (with first and/or zero-order reaction)

TDMA.FOR - Thomas algorithm for solving the three-diagonal algebraic systems of linear equations

FLUX.FOR - computation of boundary net fluxes for the overall mass (heat) balance check

MOD1.FOR - declaration module for common allocatable arrays

```
C*************************T2.FOR*********************************
C****************************************************************
C       Numerical Model for 2D Unsteady Convection-Diffusion-Reaction
C       Transport Solving, based on splitting technique:
C
C       - Holly-Preissmann Method (Holly, 1978) for advection
C       - Implicit Alternating Directions (Yanenko, 1971) for
C         diffusion, extended by reaction
C       - Source (sink) can represent first or zero-order reactions
C
C                                - January 1998 -
C****************************************************************
C****************************************************************
      PROGRAM MAIN
      use PORTLIB
      use ARRAY_ALLOCATION
      include 'param.for'
      character tit*50,str*1
      common nx,ny,i1,i2,j1,j2,rksi,reta
      allocatable tux(:),tdx(:),tiux(:),tidx(:),tuy(:),tdy(:),sp(:,:),
     +            tiuy(:),tidy(:),bc(:,:),tx(:,:),ty(:,:),sc(:,:),
     +            t(:,:),tp(:,:),txp(:,:),typ(:,:),alx(:,:),aly(:,:),
     +            txy(:,:)

      open(1,FILE='t2.in')
      open(2,FILE='t2.res')
      open(3,FILE='t2.tpl')
      open(4,FILE='t2.ana')
      open(5,FILE='t2.dat')
c-----------------------------------------------------------------------
c       Input data
c-----------------------------------------------------------------------
      read(1,'(A50)') tit
      read(1,*) timebeg,timend,dt,theta
      read(1,*) nx,ny   ! # of sect. in x,y direction
      nt=max(nx,ny)

      allocate (t(nx,ny),tp(nx,ny),tx(nx,ny),txp(nx,ny),ty(nx,ny),
     +          typ(nx,ny),x(nx),y(ny),sx(nx),sy(ny),alx(nx,ny),
     +          aly(nx,ny),u(nx,ny),v(nx,ny),up(nx,ny),vp(nx,ny),
     +          ux(nx,ny),vx(nx,ny),uxp(nx,ny),vxp(nx,ny),uy(nx,ny),
     +          vy(nx,ny),uyp(nx,ny),vyp(nx,ny),sp(nx,ny),sc(nx,ny),
     +          bc(nx,ny),txy(nx,ny))
```

```
      read(1,*) (sx(i),i=1,nx)     ! x-space increments
      read(1,*) (sy(i),i=1,ny)     ! y-space increments

      read(1,*) nubx,nuby          ! upstream boundary cond.
      allocate (tux(nubx),tuy(nuby),tiux(nubx),tiuy(nuby))
      read(1,*) (tiux(i),i=1,nubx) ! time for  x-U.B.C.
      read(1,*) (tux(i),i=1,nubx)  ! temp. for x-U.B.C.
      read(1,*) (tiuy(i),i=1,nuby) ! time for  y-U.B.C.
      read(1,*) (tuy(i),i=1,nuby)  ! temp. for y-U.B.C.

      read(1,*) ndbx,ndby          ! downstream boundary cond.
      allocate (tdx(ndbx),tdy(ndby),tidx(ndbx),tidy(ndby))
      read(1,*) (tidx(i),i=1,ndbx) ! time for D.B.C.
      read(1,*) (tdx(i),i=1,ndbx)  ! temp. for D.B.C.
      read(1,*) (tidy(i),i=1,ndby) ! time for D.B.C.
      read(1,*) (tdy(i),i=1,ndby)  ! temp. for D.B.C.

      read(1,*) nprint             ! time frequency of print
      read(1,*) nint               ! interpolation (0/1/2 - lin/FHP/HP)
      close(1)

      read(5,*) str
      read(5,*) ((tp(i,j),i=1,nx),j=1,ny)  ! initial temperatures
      read(5,*) str
      read(5,*) ((up(i,j),i=1,nx),j=1,ny)  ! init. x-dir. velocities
      read(5,*) str
      read(5,*) ((vp(i,j),i=1,nx),j=1,ny)  ! init. y-dir. velocities
      read(5,*) str
      read(5,*) ((alx(i,j),i=1,nx),j=1,ny) ! init. x diff. coeff.
      read(5,*) str
      read(5,*) ((aly(i,j),i=1,nx),j=1,ny) ! init. y diff. coeff.
      read(5,*) str
      read(5,*) ((sc(i,j),i=1,nx),j=1,ny)  ! init. zero-order source
      read(5,*) str
      read(5,*) ((sp(i,j),i=1,nx),j=1,ny)  ! init. first-order source
      close(5)
c-----------------------------------------------------------------------------
c     Initiation
c-----------------------------------------------------------------------------
      sbal=0.
      ap0=1./dt
      theta1=1.-theta
      dx=sx(2)
      dy=sy(2)
      alxc=alx(1,1)
```

```
      alyc=aly(1,1)
      x(1)=sx(1)
      y(1)=sy(1)
      timer=timebeg
      do 40 i=2,nx
40    x(i)=x(i-1)+sx(i)
      do 45 i=2,ny
45    y(i)=y(i-1)+sy(i)
55    write(2,1010) tit,dt,dx,dy,alxc,alyc,sc(1,1),sp(1,1)
      write(3,1000) tit
c-----------------------------------------------------------------------
c     Initial derivatives tx, ty, ux, uy, vx, vy
c-----------------------------------------------------------------------
      call fder2(tp,txp,typ)
      call fder2(up,uxp,uyp)
      call fder2(vp,vxp,vyp)
c-----------------------------------------------------------------------
c     Initial concentration content and boundary flux
c-----------------------------------------------------------------------
      scci=fnint2(sx,sy,tp)
C***********************************************************************
C     TIME MARCHING PROCEDURE
C***********************************************************************
      l=0
      jtime=time()
      LOOP TIME: do
      write(*,1040) timer
      if(MOD(l,nprint).eq.0) then
      write(2,1030) timer
      write(3,1020) timer,nx,ny
      do 20 j=1,ny
      do 20 i=1,nx
      write(2,1050) i,j,x(i),y(j),tp(i,j),up(i,j),vp(i,j)
20    write(3,1060) x(i),y(j),tp(i,j)
      end if
      if(timer.ge.timend) exit LOOP TIME
      timer=timer+dt
      l=l+1
c-----------------------------------------------------------------
c     Time-dependent variables updating
c     - velocity
c     - velocity derivatives
c     - source terms
c     - diffusion coefficients
c-----------------------------------------------------------------
```

```
      u=up
      v=vp
      call fder2(u,ux,uy)
      call fder2(v,vx,vy)
c-----------------------------------------------------------------------------
c     Boundary conditions
c-----------------------------------------------------------------------------
      bc(1,:)=flint(tiux,tux,nubx,timer)  ! x-U.B.C.
      bc(nx,:)=flint(tidx,tdx,ndbx,timer) ! x-D.B.C.
      bc(:,1)=flint(tiuy,tuy,nuby,timer)  ! y-U.B.C.
      bc(1:nx,ny)=flint(tidy,tdy,ndby,timer) ! y-D.B.C.
c-----------------------------------------------------------------------------
c     Advection-diffusion-reaction transport
c-----------------------------------------------------------------------------
      if(nint>0) call bgrad(dt,bc,tp,tx,ty)          ! bound. gradients
      if (nint==2) call crossder(txp,typ,txy)        ! Txy for orig. HP
      call adv(tp,txp,typ,txy,bc,nint,dt,t,tx,ty)    ! advection
      sbfc=flux(1,tp,t,sc,sp,alx,aly,dt)             ! net convec. bound. influx
      call diff(tp,t,bc,sc,sp,alx,aly,dt,theta,1)    ! x-dir diffusion
      sbfd=flux(0,tp,t,sc,sp,alx,aly,dt)             ! net diff. bound. influx
      sps=fnint2(sx,sy,sp*(theta1*t+theta*tp))/2.  !1-order source
      call diff(t,tp,bc,sc,sp,alx,aly,dt,theta,2)    ! y-dir diffusion
c-----------------------------------------------------------------------------
c     Total conservation terms for the current time
c-----------------------------------------------------------------------------
      sbf=sbfc+sbfd+flux(0,tp,t,sc,sp,alx,aly,dt)    ! net bound. influx
      scs=fnint2(sx,sy,sc)                           ! 0-order source
      sps=sps+fnint2(sx,sy,sp*(theta1*t+theta*tp))/2. ! 1-order s.
      sbal=sbal+sbf+scs+sps                          ! total net influx
c-----------------------------------------------------------------------------
c     Updating for the next time step
c-----------------------------------------------------------------------------
      tp=t
      txp=tx
      typ=ty
      up=u
      vp=v
      uxp=ux
      vxp=vx
      uyp=uy
      vyp=vy
      end do LOOP TIME
```

```
C-------------------------------------------------------------------------------
C       Final conservation check
C-------------------------------------------------------------------------------
        sccf=fnint2(sx,sy,t)                    ! final concentration content
        sccm=amax1(scci,sccf)                   ! maximum concentration content
        write(4,*) scci,sccf,sbal*dt
        e=abs(sccf-scci-sbal*dt)/sccm*100.  ! conservation error
        jtime=time()-jtime                      ! execution time (sec)
        print*,' '
        write(*,900) e
        write(*,910) jtime/60.
        print*,' '

900     format(2x,'Conservation error (%) = ',f6.3)
910     format(2x,'Execution time (min)   = ',f6.3)
1000    format('TITLE="',A50,'"'/
     +  'VARIABLES= X, Y, T')
1010    format(2x,a50//5x,'dt = ',F8.3/
     +  5x,'dx = ',F8.3/
     +  5x,'dy = ',F8.3/
     +  5x,'alx= ',F8.3/
     +  5x,'aly= ',F8.3/
     +  5x,'Sc = ',F8.3/
     +  5x,'Sp = ',F8.3//
     +  '    i    j     ',
     +  'X       Y       T       U       V'/2x,63(1H-))
1020    format('ZONE T="',f8.0,'"',' I=',I3,' J=',I3,' F=POINT')
1030    format(/2x,'Time = ',f9.0/)
1040    format(2x,'Time = ',f9.0)
1050    format(2i7,6f10.3)
1060    format(3f10.3)
        end PROGRAM MAIN
```

```
C*************************ADV.FOR*********************************
C*****************************************************************
C       2D transport model for pure advection
C       based on the characteristcs method with HP bicubic interpolation
C*****************************************************************
C*****************************************************************
        subroutine adv(tp,txp,typ,txy,bc,nint,dt,t,tx,ty)
        use ARRAY_ALLOCATION
        include 'param.for'
        common nx,ny,i1,i2,j1,j2,rksi,reta
        dimension t(nx,ny),tx(nx,ny),ty(nx,ny),tp(nx,ny),txp(nx,ny),
     +  typ(nx,ny),bc(nx,ny),txy(nx,ny)

        LOOP ADVEC: do j=1,ny
        do 280 i=1,nx
c-------------------------------------------------------------------------------
c       Foot of the characteristics (modified Euler)
c-------------------------------------------------------------------------------
        uc=u(i,j)
        vc=v(i,j)
        iu=sign(1.,uc)
        iv=sign(1.,vc)
        call bound(x,y,i,j,uc*dt,vc*dt,xp,yp,ss,reta,ifoot)

        if(ifoot<0) then                    ! foot in the field
        call char(x,xp,nx,i,iu,i1,i2,rksi)
        call char(y,yp,ny,j,iv,j1,j2,reta)
        us=flint2(up)
        vs=flint2(vp)
        else
        if(ifoot<3) then                    ! foot hits x-planes
        call char(y,ss,ny,j,iv,j1,j2,rksi)
        else                                ! foot hits y-planes
        call char(x,ss,nx,i,iu,i1,i2,rksi)
        endif
        us=flint22(up,u,ifoot)
        vs=flint22(vp,v,ifoot)
        endif

        uc=(uc+us)/2.
        vc=(vc+vs)/2.
        iu=sign(1.,uc)
        iv=sign(1.,vc)

        call bound(x,y,i,j,uc*dt,vc*dt,xp,yp,ss,reta,ifoot)
```

```
c-------------------------------------------------------------------------------
c       Boundary conditions / HP (linear) interpolation
c-------------------------------------------------------------------------------
        if(ifoot<0) then                        ! foot in the field
        call char(x,xp,nx,i,iu,i1,i2,rksi)
        call char(y,yp,ny,j,iv,j1,j2,reta)
        if(nint>0) then                         ! HP interpolation
        dx=x(i2)-x(i1)
        dy=y(j2)-y(j1)
        call hp4(tp,txp,typ,txy,dx,dy,nint,t0,tx0,ty0)
        else                                    ! linear interpolation
        t0=flint2(tp)
        end if
        else
        if(ifoot<3) then                        ! foot hits x-planes
        call char(y,ss,ny,j,iv,j1,j2,rksi)
        else                                    ! foot hits y-planes
        call char(x,ss,nx,i,iu,i1,i2,rksi)
        endif
        t0=flint22(tp,bc,ifoot)
        if(nint>0) then
        tx0=flint22(txp,tx,ifoot)
        ty0=flint22(typ,ty,ifoot)
        endif
        endif
c-------------------------------------------------------------------------------
c       Implicit propagation along the characteristic line
c-------------------------------------------------------------------------------
        t(i,j)=t0
        if(nint==0.or.(i==1.and.ifoot==1).or.
     +  (i==nx.and.ifoot==2).or.(j==1.and.ifoot==3).or.
     +  (j==ny.and.ifoot==4)) cycle ! gradients known from the B.C.
        a=ux(i,j)*dt
        b=vx(i,j)*dt
        c=uy(i,j)*dt
        d=vy(i,j)*dt
        det=(1.+a)*(1.+d)-b*c
        tx(i,j)=(tx0*(1.+d)-ty0*b)/det
        ty(i,j)=(ty0*(1.+a)-tx0*c)/det
280     continue
        END DO LOOP ADVEC
        end
```

```
C*************************CHAR.FOR*********************************
C******************************************************************
C       Computation of the foot of a characteristic line
C******************************************************************
C******************************************************************
        subroutine char(x,x0,nx,i,iu,i1,i2,rksi)
        include 'param.for'
        dimension x(1)

        if((i==1.and.iu==1).or.(i==nx.and.iu==-1)) then
        i1=i
        i2=i
        rksi=1.
        else
        i2=isearch(x,nx,x0,i,iu)
        i1=i2-iu
        dl=x(i2)-x(i1)
        rksi=(x0-x(i1))/dl
        endif
        end
c-------------------------------------------------------------------------------------------------
c       1-D array search
c-------------------------------------------------------------------------------------------------
        function isearch(x,nx,x0,j,k)
        include 'param.for'
        dimension x(1)
        nb=j
        ne=1
        if(k==-1) ne=nx

        do i=nb,ne,-k
        x1=x(i)
        x2=x(i-k)
        if((x0.ge.x1.and.x0.le.x2).or.(x0.ge.x2.and.x0.le.x1)) then
        isearch=i
        return
        end if
        end do
        end
c-------------------------------------------------------------------------
c       - nb is the begining index for serching
c       - ne is the ending index for searching
c       - k is the step (+1 or -1, depending on the search direction)
c-------------------------------------------------------------------------
```

```
C*****************************HP4.FOR*********************************
C*************************************************************************
C       Holly-Preissmann 4 interpolation procedure
C       for 2-D advection equation
C*************************************************************************
C*************************************************************************
        subroutine hp4(tp,txp,typ,txy,dx,dy,nint,t0,tx0,ty0)
        use ARRAY_ALLOCATION
        include 'param.for'
        common nx,ny,i1,i2,j1,j2,rksi,reta
        dimension tp(nx,ny),txp(nx,ny),typ(nx,ny),txy(nx,ny)

        fin()=c1*t1+c2*t2+c3*t3+c4*t4+cx1*tx1+cx2*tx2+cx3*tx3+cx4*tx4+
     +  cy1*ty1+cy2*ty2+cy3*ty3+cy4*ty4

        fin1()=cxy1*txy1+cxy2*txy2+cxy3*txy3+cxy4*txy4

        f1(p)=1.-3.*p*p+2.*p*p*p
        f2(p)=1.-f1(p)
        f3(p,dl)=p*(1.-p)*(1.-p)*dl
        f4(p,dl)=p*p*(p-1.)*dl
        g1(p,dl)=6.*p*(p-1.)/dl
        g3(p)=(p-1.)*(3.*p-1.)
        g4(p)=p*(3.*p-2.)
        t1=tp(i1,j1)
        t2=tp(i1,j2)
        t3=tp(i2,j1)
        t4=tp(i2,j2)
        tx1=txp(i1,j1)
        tx2=txp(i1,j2)
        tx3=txp(i2,j1)
        tx4=txp(i2,j2)
        ty1=typ(i1,j1)
        ty2=typ(i1,j2)
        ty3=typ(i2,j1)
        ty4=typ(i2,j2)
        if (nint==2) then ! original HP
        txy1=txy(i1,j1)
        txy2=txy(i1,j2)
        txy3=txy(i2,j1)
        txy4=txy(i2,j2)
        endif
        r1x=f1(rksi)
        r2x=f2(rksi)
        r3x=f3(rksi,dx)
```

```
      r4x=f4(rksi,dx)
      r1y=f1(reta)
      r2y=f2(reta)
      r3y=f3(reta,dy)
      r4y=f4(reta,dy)
      p1x=g1(rksi,dx)
      p3x=g3(rksi)
      p4x=g4(rksi)
      p1y=g1(reta,dy)
      p3y=g3(reta)
      p4y=g4(reta)
c-----------------------------------------------------------------
c     Interpolation of the concentration
c-----------------------------------------------------------------
      c1=r1x*r1y
      c2=r1x*r2y
      c3=r2x*r1y
      c4=r2x*r2y
      cx1=r3x*r1y
      cx2=r3x*r2y
      cx3=r4x*r1y
      cx4=r4x*r2y
      cy1=r1x*r3y
      cy2=r1x*r4y
      cy3=r2x*r3y
      cy4=r2x*r4y
      t0=fin()
      if(nint==2) then
      cxy1=r3x*r3y
      cxy2=r3x*r4y
      cxy3=r4x*r3y
      cxy4=r4x*r4y
      t0=t0+fin1()
      endif
c-----------------------------------------------------------------
c     Interpolation of the concentration x derivative
c-----------------------------------------------------------------
      c1=p1x*r1y
      c2=p1x*r2y
      c3=-c1
      c4=-c2
      cx1=p3x*r1y
      cx2=p3x*r2y
      cx3=p4x*r1y
      cx4=p4x*r2y
```

```
        cy1=p1x*r3y
        cy2=p1x*r4y
        cy3=-cy1
        cy4=-cy2
        tx0=fin()
        if(nint==2) then
        cxy1=p3x*r3y
        cxy2=p3x*r4y
        cxy3=p4x*r3y
        cxy4=p4x*r4y
        tx0=tx0+fin1()
        endif
c-------------------------------------------------------------------
c       Interpolation of the concentration y derivative
c-------------------------------------------------------------------
        c1=r1x*p1y
        c2=-c1
        c3=r2x*p1y
        c4=-c3
        cx1=r3x*p1y
        cx2=-cx1
        cx3=r4x*p1y
        cx4=-cx3
        cy1=r1x*p3y
        cy2=r1x*p4y
        cy3=r2x*p3y
        cy4=r2x*p4y
        ty0=fin()
        if(nint==2) then
        cxy1=r3x*p3y
        cxy2=r3x*p4y
        cxy3=r4x*p3y
        cxy4=r4x*p4y
        ty0=ty0+fin1()
        endif
        end
```

```
C************************BOUND.FOR*********************************
C******************************************************************
C       Subroutine for determination
C       if the foot of a characteristic line
C       intersects a boundary plane and which
C******************************************************************
C******************************************************************
        subroutine bound(x,y,i,j,udt,vdt,xp,yp,ss,rdtp,ifoot)
        include 'param.for'
        dimension x(1),y(1)
        common nx,ny

        xc(p,q)=p+udt/vdt*(yj-q)
        xs(p)=xi-udt/vdt*(yj-p)
        ys(p)=yj-vdt/udt*(xi-p)
        ftp(p,u)=p/abs(u)

        xi=x(i)
        yj=y(j)
        xp=xi-udt
        yp=yj-vdt
        x1=x(1)
        xn=x(nx)
        y1=y(1)
        yn=y(ny)

        if(xp>=x1.and.xp<=xn.and.yp>=y1.and.yp<=yn) then
        ifoot=-1
        elseif(yp<y1.and.((xp>xn.and.xi<=xc(xn,y1)).        ! case 1
     +  or.(xp>=x1.and.xp<=xn).                             ! case 1'
     +  or.(xp<x1.and.xi>xc(x1,y1)))) then                  ! case 1"
        ifoot=3
        ss=xs(y1)
        rdtp=ftp(y1-yp,vdt)
        elseif(xp<x1.and.((yp<y1.and.xi<=xc(x1,y1)).        ! case 2
     +  or.(yp>=y1.and.yp<=yn).                             ! case 2'
     +  or.(yp>yn.and.xi<xc(x1,yn)))) then                  ! case 2"
        ifoot=1
        ss=ys(x1)
        rdtp=ftp(x1-xp,udt)
        elseif(yp>yn.and.((xp<x1.and.xi>=xc(x1,yn)).        ! case 3
     +  or.(xp>=x1.and.xp<=xn).                             ! case 3'
     +  or.(xp>xn.and.xi<xc(xn,yn)))) then                  ! case 3"
        ifoot=4
        ss=xs(yn)
```

```
      rdtp=ftp(yp-yn,vdt)
      else                                                ! case 4, 4', 4"
      ifoot=2
      ss=ys(xn)
      rdtp=ftp(xp-xn,udt)
      endif
      end

C************************BGRAD.FOR*********************************
C******************************************************************
C     Calculation of boundary gradients using the pure advection equation
C******************************************************************
C******************************************************************
      subroutine bgrad(dt,bc,tp,tx,ty)
      use ARRAY_ALLOCATION
      include 'param.for'
      common nx,ny
      dimension bc(nx,ny),tp(nx,ny),tx(nx,ny),ty(nx,ny)

      nx1=nx-1
      ny1=ny-1

      do 10 i=2,nx1
      dx=x(i+1)-x(i-1)
      tx(i,1)=(bc(i+1,1)-bc(i-1,1))/dx
10    tx(i,ny)=(bc(i+1,ny)-bc(i-1,ny))/dx

      do 20 j=2,ny1
      dy=y(j+1)-y(j-1)
      ty(1,j)=(bc(1,j+1)-bc(1,j-1))/dy
20    ty(nx,j)=(bc(nx,j+1)-bc(nx,j-1))/dy

      dx1=x(2)-x(1)
      dxn=x(nx)-x(nx1)
      dy1=y(2)-y(1)
      dyn=y(ny)-y(ny1)

      tx(1,1)=(bc(2,1)-bc(1,1))/dx1
      tx(nx,1)=(bc(nx,1)-bc(nx1,1))/dxn
      tx(1,ny)=(bc(2,ny)-bc(1,ny))/dx1
      tx(nx,ny)=(bc(nx,ny)-bc(nx1,ny))/dxn
      ty(1,1)=(bc(1,2)-bc(1,1))/dy1
      ty(1,ny)=(bc(1,ny)-bc(1,ny1))/dyn
      ty(nx,1)=(bc(nx,2)-bc(nx,1))/dy1
      ty(nx,ny)=(bc(nx,ny)-bc(nx,ny1))/dyn
```

```
      do 30 i=2,nx1
      ty(i,1)=((tp(i,1)-bc(i,1))/dt-u(i,1)*tx(i,1))/v(i,1)
30    ty(i,ny)=((tp(i,ny)-bc(i,ny))/dt-u(i,ny)*tx(i,ny))/v(i,ny)

      do 40 j=2,ny1
      tx(1,j)=((tp(1,j)-bc(1,j))/dt-v(1,j)*ty(1,j))/u(1,j)
40    tx(nx,j)=((tp(nx,j)-bc(nx,j))/dt-v(nx,j)*ty(nx,j))/u(nx,j)
      end

C**********CROSSDER.FOR**********
C*****************************************************************
C     Calculation of the concentration cross-derivatives
C*****************************************************************
C*****************************************************************
      subroutine crossder(tx,ty,txy)
      use ARRAY_ALLOCATION
      include 'param.for'
      common nx,ny
      dimension tx(nx,ny),ty(nx,ny),txy(nx,ny)

      nx1=nx-1
      ny1=ny-1
      dx1=x(2)-x(1)
      dxn=x(nx)-x(nx1)
      dy1=y(2)-y(1)
      dyn=y(ny)-y(ny1)

      do 10 i=2,nx1
      dx=x(i+1)-x(i-1)
      do 10 j=2,ny1
      dy=y(j+1)-y(j-1)
10    txy(i,j)=0.5*((tx(i,j+1)-tx(i,j-1))/dy+(ty(i+1,j)-ty(i-1,j))/dx)

      do 20 j=2,ny1
      dy=y(j+1)-y(j-1)
      txy(1,j)=0.5*((tx(1,j+1)-tx(1,j-1))/dy+(ty(2,j)-ty(1,j))/dx1)
20    txy(nx,j)=0.5*((tx(nx,j+1)-tx(nx,j-1))/dy+(ty(nx,j)-ty(nx1,j))/dxn)

      do 30 i=2,nx1
      dx=x(i+1)-x(i-1)
      txy(i,1)=0.5*((tx(i,2)-tx(i,1))/dy1+(ty(i+1,1)-ty(i-1,1))/dx)
30    txy(i,ny)=0.5*((tx(i,ny)-tx(i,ny1))/dyn+(ty(i+1,ny)-ty(i-1,ny))/dx)

      txy(1,1)=0.5*((tx(1,2)-tx(1,1))/dy1+(ty(2,1)-ty(1,1))/dx1)
```

```
      txy(nx,1)=0.5*((tx(nx,2)-tx(nx,1))/dy1+(ty(nx,1)-ty(nx1,1))/dxn)
      txy(1,ny)=0.5*((tx(1,ny)-tx(1,ny1))/dyn+(ty(2,ny)-ty(1,ny))/dx1)
      txy(nx,ny)=0.5*((tx(nx,ny)-tx(nx,ny1))/dyn+(ty(nx,ny)-ty(nx1,ny))/dxn)
      end

C*************************FDER2.FOR*********************************
C*******************************************************************
C     Calculation of the 2-D array's 1st derivatives
C     by central difference formula (except for boundary nodes)
C*******************************************************************
C*******************************************************************
      subroutine fder2(ax,xx,xy)
      use ARRAY_ALLOCATION
      include 'param.for'
      common nx,ny
      dimension ax(nx,ny),xx(nx,ny),xy(nx,ny)

      fx(i,j,k,dx)=(-ax(i+2*k,j)+4.*ax(i+k,j)-3.*ax(i,j))/2./dx
      fy(i,j,k,dy)=(-ax(i,j+2*k)+4.*ax(i,j+k)-3.*ax(i,j))/2./dy

      nx1=nx-1
      ny1=ny-1

      do 5 j=1,ny
      dy=y(j+1)-y(j-1)
      do 5 i=1,nx
      if(i==1.or.i==nx) goto 6
      dx=x(i+1)-x(i-1)
      xx(i,j)=(ax(i+1,j)-ax(i-1,j))/dx        ! x - derivatives
6     if(j==1.or.j==ny) cycle
5     xy(i,j)=(ax(i,j+1)-ax(i,j-1))/dy        ! y - derivatives

      dx1=x(2)-x(1)
      dxn=x(nx1)-x(nx)
      do 10 j=1,ny
      xx(1,j)=fx(1,j,1,dx1)                   ! first column
10    xx(nx,j)=fx(nx,j,-1,dxn)                ! last  column
      dy1=y(2)-y(1)
      dyn=y(ny1)-y(ny)
      do 20 i=1,nx
      xy(i,1)=fy(i,1,1,dy1)                   ! first row
20    xy(i,ny)=fy(i,ny,-1,dyn)                ! last  row
      end
```

```
C************************FLINT.FOR*********************************
C******************************************************************
C       1-D linear interpolation
C******************************************************************
C******************************************************************
        function flint(x,y,m,x0)
        include 'param.for'
        dimension x(1),y(1)
        if(x0.gt.x(1)) go to 1
        flint=y(1)
        return
1       if(x0.lt.x(m)) go to 2
        flint=y(m)
        return
2       m1=m-1
        do i=1,m1
        if(x0.lt.x(i).or.x0.ge.x(i+1)) cycle
        flint=y(i)+(y(i+1)-y(i))/(x(i+1)-x(i))*(x0-x(i))
        return
        end do
        end

C************************FLINT2.FOR********************************
C******************************************************************
C       2-D linear interpolation between 4 nodal values
C******************************************************************
C******************************************************************
        function flint2(x)
        include 'param.for'
        common nx,ny,i1,i2,j1,j2,rksi,reta
        dimension x(nx,ny)
        f1=x(i1,j1)
        f2=x(i2,j1)
        f3=x(i2,j2)
        f4=x(i1,j2)
        flint2=(f1*(1.-rksi)+f2*rksi)*(1.-reta)+(f4*(1.-rksi)+f3*rksi)*reta
        end
```

```
C*************************FLINT22.FOR*******************************
C*******************************************************************
C       2-D linear interpolation between 4 nodal values at the boundaries
C*******************************************************************
C*******************************************************************
        function flint22(x,y,ifoot)
        include 'param.for'
        common nx,ny,i1,i2,j1,j2,rksi,reta
        dimension x(nx,ny),y(nx,ny)

        if(ifoot<3) then
        if(ifoot==1) then i=1
        else
        i=nx
        endif
        f1=x(i,j1)
        f2=x(i,j2)
        f3=y(i,j1)
        f4=y(i,j2)
        else
        if(ifoot==3) then j=1
        else
        j=ny
        endif
        f1=x(i1,j)
        f2=x(i2,j)
        f3=y(i1,j)
        f4=y(i2,j)
        endif
        flint22=(f1*(1.-rksi)+f2*rksi)*(1.-reta)+(f4*(1.-rksi)+f3*rksi)*reta
        end
```

```
C*************************DIFF.FOR*********************************
C******************************************************************
C       2D transport model for diffusion with first/zero-order reaction
C       based on alternate direction implicit  method (two TDMA procedures)
C******************************************************************
C******************************************************************
        subroutine diff(tp,t,bc,sc,sp,alx,aly,dt,theta,idir)
        use ARRAY_ALLOCATION
        include 'param.for'
        common nx,ny
        dimension tp(nx,ny),t(nx,ny),bc(nx,ny),sc(nx,ny),sp(nx,ny),
     +                  alx(nx,ny),aly(nx,ny)
        allocatable a(:),b(:),c(:),d(:),z(:)
        nt=max(nx,ny)
        allocate (a(nt),b(nt),c(nt),d(nt),z(nt))

        ap0=1./dt
        theta1=1.-theta
        if(idir==1) then                ! diffusion in x-direction
        mx=nx
        my=ny
        else                            ! diffusion in y-direction
        mx=ny
        my=nx
        endif

        LOOP DIFFUSION: do j=2,my-1
        a(1)=1.
        b(1)=0.
        c(1)=0.
        if(idir==1) then
       d(1)=bc(1,j)
        dyw=sy(j)
        dye=sy(j+1)
        else
        d(1)=bc(j,1)
        dyw=sx(j)
        dye=sx(j+1)
        endif
        dyp=dye+dyw

        LOOP DCOEFF: do i=2,mx-1
        if(idir==1) then
       alye=aly(i,j+1)
        alyw=aly(i,j-1)
```

```
 alyp=aly(i,j)
alxe=alx(i+1,j)
 alxw=alx(i-1,j)
 alxp=alx(i,j)
 te=t(i,j+1)
 tw=t(i,j-1)
 tpp=t(i,j)
 scij=sc(i,j)
 spij=sp(i,j)
 dxw=sx(i)
 dxe=sx(i+1)
 else
alye=alx(j+1,i)
 alyw=alx(j-1,i)
 alyp=alx(j,i)
alxe=aly(j,i+1)
 alxw=aly(j,i-1)
 alxp=aly(j,i)
 te=t(j+1,i)
 tw=t(j-1,i)
 tpp=t(j,i)
 scij=sc(j,i)
 spij=sp(j,i)
 dxw=sy(i)
 dxe=sy(i+1)
 endif
 aye=theta1*(alyp+alye)/dye/dyp
 ayw=theta1*(alyp+alyw)/dyw/dyp
 ayp=aye+ayw
 dxp=dxe+dxw
 axe=theta*(alxp+alxe)/dxe/dxp
 axw=theta*(alxp+alxw)/dxw/dxp
 a(i)=axe+axw+ap0-.5*spij*theta
b(i)=axe
c(i)=axw
d(i)=(ap0+.5*spij*theta1-ayp)*tpp+ayw*tw+aye*te+.5*scij
 END DO LOOP DCOEFF

 a(mx)=1.
 b(mx)=0.
 c(mx)=0.
 if(idir==1) then
d(mx)=bc(mx,j)
 else
 d(mx)=bc(j,mx)
```

```
      endif
      CALL TDMA(mx,a,b,c,d,z)
      if(idir==1) then
      tp(:,j)=(/(z(i),i=1,mx)/)
      else
      tp(j,:)=(/(z(i),i=1,mx)/)
      endif
      END DO LOOP DIFFUSION

      if(idir==1) then              ! diffusion in x-direction
      tp(:,1)=bc(:,1)
      tp(:,ny)=bc(:,ny)
      else                          ! diffusion in y-direction
      tp(1,:)=bc(1,:)
      tp(nx,:)=bc(nx,:)
      endif
      deallocate (a,b,c,d,z)
      end

C**********TDMA.FOR*****************************
C************************************************
C     Thomas algorithm for tridiagonal systems of linear algebraic equations
C************************************************
C************************************************
      subroutine TDMA(n,a,b,c,d,t)
      include 'param.for'
      dimension a(1),b(1),c(1),d(1),t(1),P(n),Q(n)

      pi=0.
      qi=0.
      c(1)=0.
      b(n)=0.
      ti=0.
      do i=1,n
      pp=a(i)-c(i)*pi
      pi=b(i)/pp
      qi=(d(i)+c(i)*qi)/pp
      p(i)=pi
      q(i)=qi
      end do
      do 20 i=n,1,-1
      ti=p(i)*ti+q(i)
20    t(i)=ti
      end
```

```
C*************************FLUX.FOR*********************************
C******************************************************************
C       Boundary net influx
C******************************************************************
C******************************************************************
        function flux(iflux,tp,t,sc,sp,alx,aly,dt)
        use ARRAY_ALLOCATION
        include 'param.for'
        common nx,ny
        dimension t(nx,ny),tp(nx,ny),sc(nx,ny),sp(nx,ny),alx(nx,ny),aly(nx,ny)
        allocatable xs(:)

        uta(k,l)=(u(k,l)*t(k,l)+up(k,l)*tp(k,l))/2.
        vta(k,l)=(v(k,l)*t(k,l)+vp(k,l)*tp(k,l))/2.

        allocate (xs(max(nx,ny)))
        do 10 j=1,ny
        if(iflux==0) then                                  ! x-diffusive influx
        xd1=-fgrad(nx,j,1,sx(nx),dt,sy,alx,aly,tp,t,sc,sp)  ! outflux
        xd2=-fgrad(1,j,1,sx(2),dt,sy,alx,aly,tp,t,sc,sp)    ! influx
        xflux=xd1+xd2
        else                                               ! x-convective influx
        xflux=uta(1,j)-uta(nx,j)
        endif
10      xs(j)=xflux
        sfx=fsum(ny,sy,xs)

        do 20 i=1,nx
        if(iflux==0) then                                  ! y-diffusive influx
        xd1=-fgrad(ny,i,2,sy(ny),dt,sx,aly,alx,tp,t,sc,sp)  ! outflux
        xd2=-fgrad(1,i,2,sy(2),dt,sx,aly,alx,tp,t,sc,sp)    ! influx
        xflux=xd1+xd2
        else                                               ! y-convective influx
        xflux=vta(i,1)-vta(i,ny)
        endif
20      xs(i)=xflux
        sfy=fsum(nx,sx,xs)

        flux=sfx+sfy
        deallocate (xs)
        end
```

```
c-------------------------------------------------------------------------------------------
c       Boundary gradients
c-------------------------------------------------------------------------------------------
        function fgrad(n,j,k,se,dt,s,alfx,alfy,tp,t,sc,sp)
        include 'param.for'
        common nx,ny
        dimension s(1),alfx(nx,ny),alfy(nx,ny),tp(nx,ny),t(nx,ny),sc(nx,ny),sp(nx,ny)
        m=1
        cc=0.
        if(n.ne.1) m=-1
        dy=(s(j)+s(j+1))/2.
        ip=n
        ie=n+m
        is=n
        in=n
        jp=j
        je=j
        js=j-1
        jn=j+1
        nend=ny
        if(k.eq.2) then         ! y-dir. gradients
        call swap(ip,jp)
        call swap(ie,je)
        call swap(is,js)
        call swap(in,jn)
        nend=nx
        cc=1.
        endif
        cc1=1.-cc
        tpp=t(ip,jp)
        te=t(ie,je)
        tpe=tp(ie,je)
        tppp=tp(ip,jp)
        ae=(alfx(ie,je)+alfx(ip,jp))/4.
        re=(cc*(te-tpp)+cc1*(tpe-tppp))/se
        rdt=(tppp-tpp)/dt
        rsc=sc(ip,jp)/2.
        rsp=sp(ip,jp)*(tpp+tppp)/4.
        if(j.eq.1.or.j.eq.nend) then  ! first or last point
        an=0.
        as=0.
        rdt=rdt/2.
        rsc=rsc/2.
        else
        ts=t(is,js)
```

```
      tn=t(in,jn)
      tps=tp(is,js)
      tpn=tp(in,jn)
      apy=alfy(ip,jp)
      as=(alfy(is,js)+apy)/4.
      an=(alfy(in,jn)+apy)/4.
      rs=(cc1*(tpp-ts)+cc*(tppp-tps))/s(j)
      rn=(cc1*(tn-tpp)+cc*(tpn-tppp))/s(j+1)
      endif
      fgrad=ae*re+se/2.*((an*rn-as*rs)/dy+rsc+rsp-rdt)
      end
c-----------------------------------------------------------------------------------------
c     Swaping of indices
c-----------------------------------------------------------------------------------------
      subroutine swap(i,j)
      l=i
      i=j
      j=l
      end
c-----------------------------------------------------------------------------------------
c     1-D Numerical integration
c-----------------------------------------------------------------------------------------
      function fsum(n,st,x)
      include 'param.for'
      dimension st(1),x(1)

      fsum=x(1)*st(2)/2.
      do 10 i=2,n-1
10    fsum=fsum+x(i)*(st(i)+st(i+1))/2.
      fsum=fsum+x(n)*st(n)/2.
      end

C*****************************MOD1.FOR*********************************
C**********************************************************************
C     Declaration of common allocatable 2-D arrays
C**********************************************************************
C**********************************************************************
      module array_allocation
      include 'param.for'
      allocatable ::x(:),y(:),sx(:),sy(:),up(:,:),vp(:,:),u(:,:),v(:,:),
     +              ux(:,:),uy(:,:),uxp(:,:),uyp(:,:),
     +              vx(:,:),vy(:,:),vxp(:,:),vyp(:,:)
      end module
```