Applied and Computational Mathematics
2015; 4(3-1): 59-77
Published online April 7, 2015 (http://www.sciencepublishinggroup.com/j/acm)
doi: 10.11648/j.acm.s.2015040301.15
ISSN: 2328-5605 (Print); ISSN: 2328-5613 (Online)

Implementation of One and Two-Step Generalized Integral Representation Methods (GIRMs)

Hideyuki Niizato[1], Gantulga Tsedendorj[2], Hiroshi Isshiki[3]

[1]Hitachi Zosen Corporation, Osaka, Japan
[2]Department of Mathematics, National University of Mongolia, Ulaanbaatar, Mongolia
[3]IMA, Institute of Mathematical Analysis, Osaka, Japan

Email address:
niizato@hitachizosen.co.jp (H. Niizato), gantulga@smcs.num.edu.mn (G. Tsedendorj), isshiki@dab.hi-ho.ne.jp (H. Isshiki)

To cite this article:
Hideyuki Niizato, Gantulga Tsedendorj, Hiroshi Isshiki. Implementation of One and Two-Step Generalized Integral Representation Methods (GIRMs). *Applied and Computational Mathematics*. Special Issue: Integral Representation Method and its Generalization. Vol. 4, No. 3-1, 2015, pp. 59-77. doi: 10.11648/j.acm.s.2015040301.15

Abstract: In this study, we summarize and implement one- and two-step Generalized Integral Representation Methods (GIRMs). Although GIRM requires matrix inversion, the solution is stable and the accuracy is high. Moreover, it can be applied to an irregular mesh. In order to validate the theory, we apply one- and two-step GIRMs to the one-dimensional Initial and Boundary Value Problem for advective diffusion. The numerical experiments are conducted and the approximate solutions coincide with the exact ones in both cases. The corresponding computer codes implemented in most popular computational languages are also given.

Keywords: Initial and Boundary Value Problem (IBVP), Generalized Fundamental Solution, Generalized Integral Representation Method (GIRM), Implementation of GIRM, Computer Codes

1. Introduction

Integral Representation Method (IRM) and its generalization - Generalized Integral Representation Method (GIRM) are one of the most convenient methods to numerically solve Initial and Boundary Value Problems (IBVP) such as advection-diffusion type equations. Both methods can be applied to an irregular mesh, and the solution is stable and accurate. In Ref. [1], generalization from IRM to GIRM is discussed not only from the theoretical viewpoint, but also from the computational aspects. Comparison and relationship with other numerical methods are also given. As an example, GIRM is applied to one- and two-dimensional diffusion problems and two-dimensional Burgers' equation. Moreover, it is also shown that the determination of a fundamental solution for GIRM is always possible in advance.

As another demonstration, GIRM is applied to fluid dynamic motion of gas or particles to obtain the accurate numerical solutions [2]. The numerical results by the GIRM are compared with the solutions by Finite Difference Method (FDM). The GIRM produces reasonable and accurate numerical solutions.

In the present study, a brief summary of the theory of One- and Two-step Generalized Integral Representation Methods (GIRMs) are presented. To validate the theory, numerical experiments of the one-dimensional IBVP for advective diffusion are conducted. Highly accurate numerical results are obtained in both cases of One- and Two-step GIRMs in admissible time periods. The corresponding computer codes implemented in widely used programming languages such as Matlab, C and FORTRAN are also given.

2. One-Step Generalized Integral Representation Method (1-Step GIRM)

2.1. Summary of Theory

We discuss numerical solutions of the IBVP for advective diffusion. For simplicity, we apply 1-step GIRM to the one-dimensional problem with constant advection velocity and constant diffusion coefficient.

If $C(x,t)$ is substance density in the region $-L < x < L$, the corresponding IBVP is given by

Differential equation:

$$\frac{\partial C}{\partial t}+U\frac{\partial C}{\partial x}=\kappa\frac{\partial^2 C}{\partial x^2}+\sigma,\ -L<x<L,\ t>0 \quad (1)$$

Boundary condition:

$$C(-L,t)=g_{-L}(t),\ C(+L,t)=g_{+L}(t)\ ,\ t>0 \quad (2)$$

Initial condition:

$$C(x,0)=f(x),\ -L<x<L \quad (3)$$

where x is space coordinate, t is time, U is advection velocity, κ is diffusion constant and $\sigma(x,t)$ is source of the substance, respectively.

Multiplying both sides of Eq. (1) by a function $\tilde{G}(x,\xi)$ of x and ξ and integrating over $-L<x<L$, we obtain

$$0=\int_{-L}^{+L}\left(\frac{\partial C(x,t)}{\partial t}+U\frac{\partial C(x,t)}{\partial x}-\kappa\frac{\partial^2 C(x,t)}{\partial x^2}-\sigma(x)\right)\tilde{G}(x,\xi)dx=\int_{-L}^{+L}\frac{\partial C(x,t)}{\partial t}\tilde{G}(x,\xi)\,dx-U\int_{-L}^{+L}C(x,t)\frac{\partial\tilde{G}(x,\xi)}{\partial x}dx$$
$$-\kappa\int_{-L}^{+L}C(x,t)\frac{\partial^2\tilde{G}(x,\xi)}{\partial x^2}dx+U\left[C(x,t)\tilde{G}(x,\xi)\right]_{x=-L}^{x=+L}-\kappa\left[\frac{\partial C(x,t)}{\partial x}\tilde{G}(x,\xi)-C(x,t)\frac{\partial\tilde{G}(x,\xi)}{\partial x}\right]_{x=-L}^{x=+L}-\int_{-L}^{+L}\sigma(x,t)\tilde{G}(x,\xi)\,dx \quad (4)$$

If we rewrite Eq. (4) and exchange x and ξ, we have

$$\int_{-L}^{+L}\frac{\partial C(\xi,t)}{\partial t}\tilde{G}(\xi,x)\,d\xi=U\int_{-L}^{+L}C(\xi,t)\frac{\partial\tilde{G}(\xi,x)}{\partial\xi}d\xi+\kappa\int_{-L}^{+L}C(\xi,t)\frac{\partial^2\tilde{G}(\xi,x)}{\partial\xi^2}d\xi-U\left[C(\xi,t)\tilde{G}(\xi,x)\right]_{\xi=-L}^{\xi=+L}+\int_{-L}^{L}\sigma(\xi,t)\tilde{G}(\xi,x)\,d\xi$$
$$+\kappa\left[\frac{\partial C(\xi,t)}{\partial\xi}\tilde{G}(\xi,x)-C(\xi,t)\frac{\partial\tilde{G}(\xi,x)}{\partial\xi}\right]_{\xi=-L}^{\xi=+L} \quad (5)$$

where $\tilde{G}(x,\xi)$ is a Generalized Fundamental Solution (GFS) chosen properly. For instance, we take Gaussian GFS:

$$\tilde{G}(x,\xi)=\frac{1}{\sqrt{2\pi}\,\gamma}\exp\left(-\frac{(x-\xi)^2}{2\gamma^2}\right) \quad (6)$$

Eq. (5) is a generalized integral representation of Eq. (1). This integral representation is applied to numerical solution. If $C(x,t)$ in $-L<x<L$, $C(-L,t)$ and $C(L,t)$ are known, then Eq. (5) is an integral equation with unknowns $\partial C(x,t)/\partial t$ in $-L<x<L$, $C_x(-L,t)$ and $C_x(L,t)$, where $\tilde{G}(x,\xi)$ is the kernel function. Namely, we are able to obtain $C(x,t)$ numerically, if we use for instance, the following procedure:

Let $C(x,t)$ be known at time t $\rightarrow$

Obtain $\partial C(x,t)/\partial t$ from Eq. (5) $\rightarrow$

Apply $C(x,t+dt)=C(x,t)+dt\,\partial C(x,t)/\partial t$ $\rightarrow$

Increment time by dt $\rightarrow$ Repeat process. (7)

2.2. Numerical Experiment

To begin with, we introduce for example, a regular mesh:

$$dx=d\xi=2L/N,$$

$$x_i=\xi_i=-L+(i+0.5)dx,\ i=0,1,\ldots,N-1 \quad (8)$$

$$t_n=ndt,\ n=0,1,\ldots \quad (9)$$

and denote

$$C_i^{(n)}=C(x_i,t_n),\ \sigma_i^{(n)}=\sigma(x_i,t_n),\ \left[\frac{\partial C}{\partial t}\right]_j^{(n)}=\frac{\partial C(\xi_j,t_n)}{\partial t} \quad (10)$$

We prepare the following approximations for discretization of Eq. (5)

$$\int_{-L}^{+L}\frac{\partial C(\xi,t_n)}{\partial t}\tilde{G}(x,\xi)\,d\xi=\sum_{j=0}^{N-1}\int_{\xi_j-d\xi/2}^{\xi_j+d\xi/2}\frac{\partial C(\xi,t_n)}{\partial t}\tilde{G}(x,\xi)d\xi=\sum_{j=0}^{N-1}\left[\frac{\partial C}{\partial t}\right]_j^{(n)}\int_{\xi_j-d\xi/2}^{\xi_j+d\xi/2}\tilde{G}(x,\xi)d\xi \quad (11a)$$

$$\left[\frac{\partial C(\xi,t_n)}{\partial\xi}\tilde{G}(x,\xi)\right]_{\xi=-L}^{\xi=+L}=\frac{\partial C(+L,t_n)}{\partial\xi}\tilde{G}(x,+L)-\frac{\partial C(-L,t_n)}{\partial\xi}\tilde{G}(x,-L) \quad (11b)$$

$$\int_{-L}^{+L}\sigma(\xi,t_n)\tilde{G}(x,\xi)\,d\xi=\sum_{j=0}^{N-1}\int_{\xi_j-d\xi/2}^{\xi_j+d\xi/2}\sigma(\xi,t_n)\tilde{G}(x,\xi)d\xi=\sum_{j=0}^{N-1}\sigma_j^{(n)}\int_{\xi_j-d\xi/2}^{\xi_j+d\xi/2}\tilde{G}(x,\xi)d\xi \quad (11c)$$

$$\left[C(\xi,t_n)\frac{\partial\tilde{G}(x,\xi)}{\partial\xi}\right]_{\xi=-L}^{\xi=+L}=g_{+L}(t_n)\frac{\partial\tilde{G}(x,+L)}{\partial\xi}-\,-g_{-L}(t_n)\frac{\partial\tilde{G}(x,-L)}{\partial\xi} \quad (11d)$$

where $\partial C(\xi_j,t_n)/\partial t = C_\xi(\xi_j,t_n)$ etc.

$$\sum_{j=0}^{N-1}\left[\frac{\partial C}{\partial t}\right]_j^{(n)}\Gamma_j(x)-\kappa\left[\frac{\partial C(L,t_n)}{\partial\xi}\tilde{G}(L,x)-\frac{\partial C(-L,t_n)}{\partial\xi}\tilde{G}(-L,x)\right]=\kappa\sum_{j=0}^{N-1}C_j^{(n)}\Xi_j(x)+U\sum_{j=0}^{N-1}C_j^{(n)}\Lambda_j(x)+\sum_{j=0}^{N-1}\sigma_j^{(n)}\Gamma_j(x)$$
$$-U\left[g_{+L}(t_n)\tilde{G}(L,x)-g_{-L}(t_n)\tilde{G}(-L,x)\right]-\kappa\left[g_{+L}(t_n)\frac{\partial\tilde{G}(L,x)}{\partial\xi}-g_{-L}(t_n)\frac{\partial\tilde{G}(-L,x)}{\partial\xi}\right] \quad (12)$$

where

$$\Gamma_j(x)=\int_{\xi_j-d\xi/2}^{\xi_j+d\xi/2}\tilde{G}(x,\xi)d\xi$$
$$\Lambda_j(x)=\int_{\xi_j-d\xi/2}^{\xi_j+d\xi/2}\frac{\partial\tilde{G}(\xi,x)}{\partial\xi}d\xi \quad (13)$$
$$\Xi_j(x)=\int_{\xi_j-d\xi/2}^{\xi_j+d\xi/2}\frac{\partial^2\tilde{G}(\xi,x)}{\partial\xi^2}d\xi$$

The unknowns in Eq. (12) are $\left[\partial C/\partial t\right]_j^{(n)}$, $j=0,1,\ldots,N-1$, $\partial C(-L,t_n)/\partial\xi$ and $\partial C(L,t_n)/\partial\xi$. Eq. (12) is satisfied at the interior points $x=x_0,x_1,\ldots,x_{N-1}$ as well as at the boundary points $x=\pm L$. Hence, we have $N+2$ equations for $N+2$ unknowns. Furthermore, if we use approximations

$$\frac{\partial C(-L,t_n)}{\partial\xi}=\frac{2}{d\xi}\left(C(x_0,t_n)-C(-L,t_n)\right),$$

Thus Eq. (5) can be discretized

$$\frac{\partial C(+L,t_n)}{\partial\xi}=\frac{2}{d\xi}\left(C(+L,t_n)-C(x_{N-1},t_n)\right) \quad (14)$$

and satisfy Eq. (12) at the interior points, we have N equations for N unknowns.

Although GIRM is computationally complex and requires matrix inversion, its accuracy of the numerical results is high. It can be applied to an irregular mesh. If a computer code is properly composed and implemented, the computational load might be comparable with the Finite element Method (FEM).

Numerical examples of the GIRM are given below. The initial condition is exponential and given by

$$C(x,0)=\exp\left(-\frac{1}{2}\left(\frac{x}{L/8}\right)^2\right) \quad (15)$$

We assume that L is large enough, and the boundary condition is specified as

$$C(\pm L,t)=0 \quad (16)$$

The exact solution is given by

$$C(x,t)=\frac{1}{2\sqrt{\pi\kappa t}}\int_{-\infty}^{+\infty}\exp\left(-\frac{1}{2}\left(\frac{\xi}{L/8}\right)^2\right)\exp\left(-\frac{(x-\xi-Ut)^2}{4\kappa t}\right)d\xi \quad (17)$$

In order to reduce spurious oscillation in numerical solutions, it is effective to use a finer mesh, but it invites serious increase of computational resource. Therefore, if necessary, numerical damping

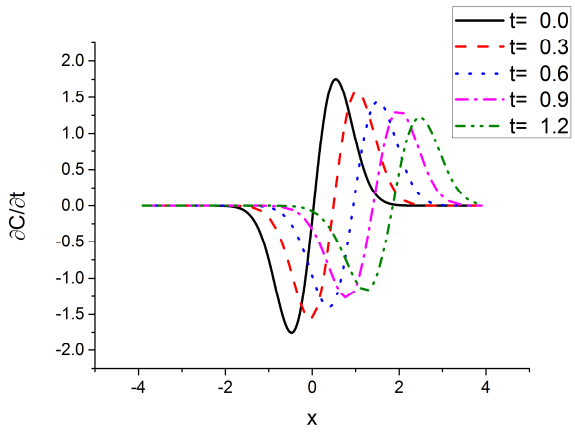
$$-\frac{4\alpha}{dx^2}\left[C_i^{(n)}-\frac{1}{4}\left(C_{i+1}^{(n)}+2C_i^{(n)}+C_{i-1}^{(n)}\right)\right] \quad (18)$$

is added to $C_i^{(n)}$ at every time step, where α is a damping constant. Moreover, if the discontinuity of the initial density causes serious numerical errors, it is effective to replace $C_i^{(0)}$ with a filtered value such as

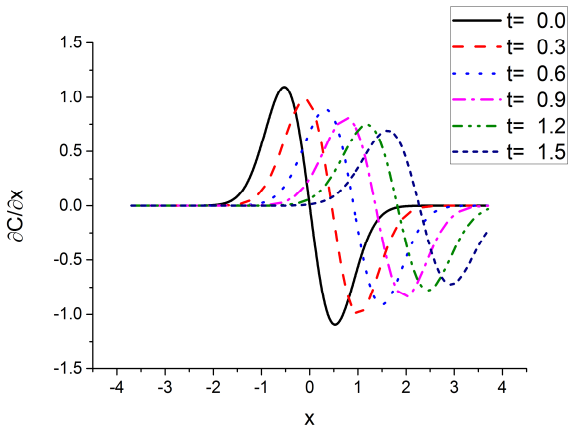
$$\frac{1}{4}\left(C_{i+1}^{(0)}+2C_i^{(0)}+C_{i-1}^{(0)}\right) \quad (19)$$

Finally, for the reduction of computation time, numerical integrals that include function G and its derivatives w.r.t ξ in the right hand side of Eq. (5) are calculated in the neighborhood of x_i only:

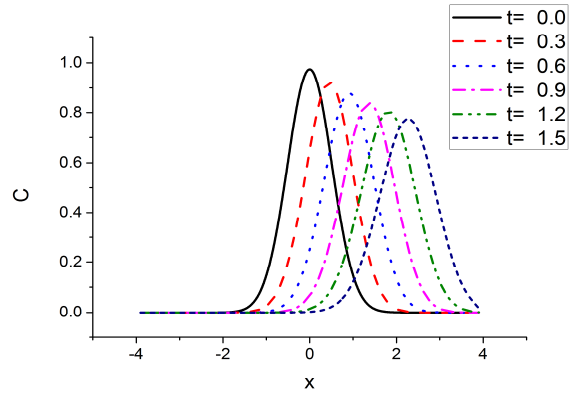
$$\sum_{j=0}^{N-1}I(i,j)\approx\sum_{|i-j|\le bdw}I(i,j) \quad (20)$$



(a) ∂C/∂t



(b) ∂C/∂x



(c) C

Figure 1. *Numerical solutions by 1-Step GIRM: (a) Time derivative ∂C/∂t, (b) Space derivative ∂C/∂x and (c) Solution C itself, where used exponential initial density distribution.*

Numerical results are shown in Fig. 1. The accuracy of the numerical results is very high and it coincides with the exact ones. Values of the parameters used in the numerical experiment are:

$$L=4,\ N=40,\ dx=2L/8=0.2,\ dt=0.0005,\ U=1.5,\ T=3000dt,\ \gamma=0.75dx,\ \kappa=0.05,\ NItg=3. \quad (21)$$

The computer codes are given in Appendix A.

3. Two-Step Generalized Integral Representation Method (2-Step GIRM)

3.1. Summary of Theory

In order to derive two-step GIRM for Eq. (1), we rewrite it as follows:

$$\text{Non-uniformity equation: } \theta=\frac{\partial C}{\partial x} \quad (22)$$

$$\text{Constitutive equation: } q=-\kappa\theta \quad (23)$$

$$\text{Equilibrium equation: } \frac{\partial C}{\partial t}+U\frac{\partial C}{\partial x}=-\frac{\partial q}{\partial x}+\sigma \quad (24)$$

Multiplying both sides of Eq. (22) by function $\tilde{G}(x,\xi)$ and integrating over $-L<x<L$, we obtain

$$0=\int_{-L}^{+L}\left[\theta(x,t)-\frac{\partial C(x,t)}{\partial x}\right]\tilde{G}(x,\xi)\,dx=\int_{-L}^{+L}\left[\tilde{G}(x,\xi)\theta(x,t)+C(x,t)\tilde{\delta}(x,\xi)\right]dx-\left[C(+L,t)\tilde{G}(+L,\xi)-C(-L,t)\tilde{G}(-L,\xi)\right] \quad (25)$$

where

$$\frac{\partial\tilde{G}(x,\xi)}{\partial x}=\tilde{\delta}(x,\xi) \quad (26)$$

Rewriting Eq. (25) and exchanging x and ξ, we obtain a generalized integral representation for Eq. (22):

$$\int_{-L}^{+L}\tilde{G}(\xi,x)\theta(\xi,t)\,d\xi=-\int_{-L}^{+L}C(\xi,t)\tilde{\delta}(\xi,x)\,d\xi+\left[\tilde{G}(+L,x)C(+L,t)-\tilde{G}(-L,x)C(-L,t)\right] \quad (27)$$

A generalized integral representation of Eq. (24) is obtained similarly:

$$0=\int_{-L}^{+L}\tilde{G}(x,\xi)\left[\frac{\partial C(x,t)}{\partial t}+U\frac{\partial C(x,t)}{\partial x}+\frac{\partial q(x,t)}{\partial x}-\sigma(x,t)\right]dx=\int_{-L}^{+L}\tilde{G}(x,\xi)\frac{\partial C(x,t)}{\partial t}dx-U\int_{-L}^{+L}C(x,t)\delta(x,\xi)\,dx$$
$$+U\left[\tilde{G}(+L,\xi)C(+L,t)-\tilde{G}(-L,\xi)C(-L,t)\right]+\left[\tilde{G}(+L,\xi)q(+L,t)-\tilde{G}(-L,\xi)q(-L,t)\right]-\int_{-L}^{+L}q(x,t)\tilde{\delta}(x,\xi)\,dx-\int_{-L}^{+L}\sigma(x,t)\tilde{G}(x,\xi)\,dx \quad (28)$$

Rewriting Eq. (28) and exchanging x and ξ, we obtain a generalized integral representation for Eq. (24):

$$\int_{-L}^{+L}\tilde{G}(\xi,x)\frac{\partial C(\xi,t)}{\partial t}d\xi=\int_{-L}^{+L}\left[UC(\xi,t)+q(\xi,t)\right]\tilde{\delta}(\xi,x)\,d\xi+\int_{-L}^{+L}\sigma(x,t)\tilde{G}(x,\xi)\,dx-\left[\tilde{G}(+L,x)q(+L,t)-\tilde{G}(-L,x)q(-L,t)\right]$$
$$-U\left[\tilde{G}(+L,x)C(+L,t)-\tilde{G}(-L,x)C(-L,t)\right] \quad (29)$$

Hence, we are able to solve $C(x,t)$ numerically, if we use for example, the following procedure:

$$\text{Let } C(x,t) \text{ be known } \rightarrow \text{ Obtain } \theta(x,t) \text{ from (27)}$$

$$\text{Obtain } q(x,t) \text{ from (23) } \rightarrow \text{ Obtain } \partial C(x,t)/\partial t \text{ from (29)}$$

$$\text{Then } C(x,t+dt) = C(x,t) + dt\,\partial C(x,t)/\partial t \rightarrow$$

$$\text{Add } dt \text{ to } t \rightarrow \text{Repeat process.} \qquad (30)$$

3.2. Numerical Experiment

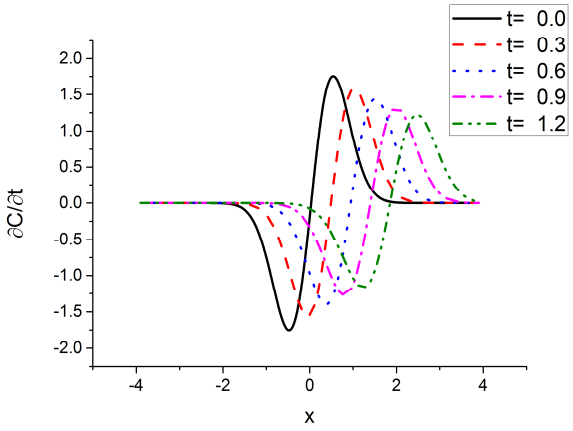
Numerical examples are given for the same problem as in case of the 1-Step GIRM. The initial and boundary conditions are given by Eq. (3) and Eq. (16), respectively. The exact solution is given by Eq. (17). Values of the parameters used in the numerical experiment are:

$$L = 4,\ N = 40,\ dx = 2L/8 = 0.2,\ dt = 0.0005,\ U = 1.5,$$
$$T = 3000dt,\ \gamma = 0.75dx,\ \kappa = 0.05,\ NItg = 3. \qquad (31)$$

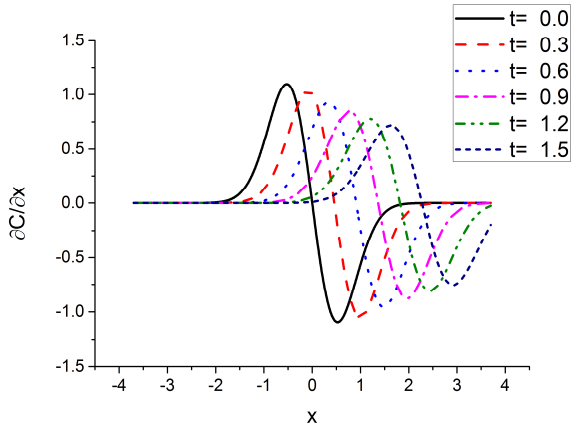
Numerical results are shown Fig. 2. The accuracy of the numerical results is very high and it coincides with the exact ones. The corresponding computer codes are given in Appendix B.

4. Characteristics of Generalized Fundamental Solutions (GFS)

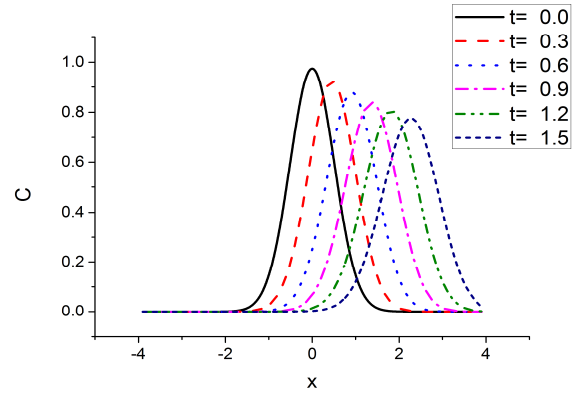
In the numerical experiments in 2.2 and 3.2, Gaussian GFS was used. However, Gaussian GFS causes some problems, when the boundary value is not zero [3,4]. The remedies are given in Ref. [3,4]. For example, C^0 GFS such as harmonic and exponential GFSs works well. If the non-zero boundary value problem is transformed into a zero boundary value problem, Gaussian GFS is very effective.



(a) ∂C/∂t



(b) ∂C/∂x



(c) C

Figure 2. *Numerical solutions by 2-Step GIRM: (a) Time derivative ∂C/∂t, (b) Space derivative ∂C/∂x and (c) Solution C itself, where used exponential initial density distribution.*

5. Conclusion

GIRM is a convenient alternative to numerically solve the IBVP such as advective diffusion. Numerical solutions obtained by GIRM in particular, by 1- and 2-step GIRMs are stable and accurate.

In the present paper, we summarize and implement 1- and 2-step GIRMs. As implementation demonstration, we provide computer codes written in widely used computational languages including low-level programming languages such as C and FORTRAN as well as high-level language such as Matlab.

Appendix A. Computer codes of 1-Step GIRM

A.1. Matlab code

```
function GIRM_1_step ( )
% Solves 1D IBVP for advective diffusion using 1-step GIRM.
```

```
   N = 40;                          % number of element
   L = 4.;                          % half length of region
   dt = 0.0005;                     % time step
   iend = 3000;                     % end of calculation
   U = 1.5;                         % advection velocity
   kpp = 0.05;                      % diffusion constant
   C0 = 0.;                         % C0 = C(-L,t)
   C1 = 0.;                         % C1 = C(+L,t)
   gam = 0.15;                      % scale of generalized
fundamental solution Gchld
   alp = 0.0;                       % artificial damping
coefficient
   NItg = 3;                        % number of division for
numerical integration

        dx = 2. * L / (N + 0);
        for i = 1:N
            x(i) = -L+0.5*dx+(i-1)*dx;
            C(1,i) = f_ini(x(i), L);
        end

        % Set matrix A
        for i = 1:N
            for j = 1:N;
                AMAT(i,j) = 0.;
                for p = 1:NItg
                    AMAT(i,j) = AMAT(i,j) +
Gchld(x(j)-dx/2.+(p-0.5)*dx/(NItg+0.),x(i),gam)*dx/(NItg+0
.);
                end
            end
        end

        for i = 1:N
            for j = 1:N
                if (i == j)
                    AMAT(i,N+j) = 1.;
                else
                    AMAT(i,N+j) = 0.;
                end
            end
        end

        A1 = inverse(AMAT, N); % Inverse of matrix A

        % Loop over time
        for it = 1:iend
            for i = 1:N
                vec1(i) = 0.;
                for j = 1:N
                    vec1(i) = vec1(i) +
C(it,j)*(U*Gchld_x(x(j),x(i),gam)+kpp*Gchld_xx(x(j),x(i),g
am))*dx;
                end

                vec1(i) = vec1(i) + U*( -
C(it,N)*Gchld(L,x(i),gam) + C(it,1)*Gchld(-L,x(i),gam) );
                vec1(i) = vec1(i) -
kpp*C1*Gchld_x(+L,x(i),gam) +
kpp*C0*Gchld_x(-L,x(i),gam);
                vec1(i) = vec1(i) +
kpp*(C1-C(it,N))/(0.5*dx)*Gchld(+L,x(i),gam) -
kpp*(C(it,1)-C0)/(0.5*dx)*Gchld(-L,x(i),gam);
            end

            for i = 1:N
                Dt_C(it,i) = 0.;
                for j = 1:N
                    Dt_C(it,i) = Dt_C(it,i) +
A1(i,N+j)*vec1(j);
                end
            end

            for i = 1:N
                C(it+1,i) = C(it,i)+Dt_C(it,i)*dt;
            end

            for i = 2:N-1
                tmp(i) =
C(it+1,i)+alp*( (C(it+1,i+1)-2.0*C(it+1,i)+C(it+1,i-1))/dx/dx
 ); % Reduce spurious oscillation
            end

            for i = 2:N-1
                C(it+1,i) = tmp(i); % Copy solution
            end

        end
        plot(x,C(1:itvl:iend,:)); xlabel ( 'x' ); ylabel ( 'C(x,t)');
ylabel ( 'C(x,t)');
    end

    function A1 = inverse( AMAT, N)
        i = 0; j = 0; k = 0; N1 = 0; N1 = N + N;

        for k=1:N
            AKK = AMAT(k,k);
            for j=k:N1
                AMAT(k,j) = AMAT(k,j) / AKK;
            end

            for i=1:N
                if (i == k)
                    continue;
                end
                AIK = AMAT(i,k);
                for j=k:N1
                    AMAT(i,j) = AMAT(i,j) -
AIK*AMAT(k,j);
                end
            end
        end
        A1 = AMAT;
    end
```

```
function G = Gchld( x, xsi, gam)
    G = 1./sqrt(2.*pi)/gam*exp(-(x-xsi)^2/2./gam/gam);
end

function Gx = Gchld_x( x, xsi, gam)
    Gx = -(x-xsi)/sqrt(2.*pi)/(gam^3)*exp(-(x-xsi)^2/2./gam/gam);
end

function Gxx = Gchld_xx( x, xsi, gam)
    Gxx = -1.0/sqrt(2.0*pi)/(gam^3)*exp(-(x-xsi)*(x-xsi)/2.0/gam/gam)
...

+(x-xsi)*(x-xsi)/sqrt(2.0*pi)/(gam^5)*exp(-(x-xsi)*(x-xsi)/2.
0/gam/gam);
end

function f = f_ini( x, L)
     f = exp(-(x^2/(L/8.)^2)/2.);
end
```

A.2. C code

```
/* ---------------------------------------------------------------- */
/*  Solves the IBVP for the advective diffusion using
1-step GIRM  */
/* ---------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define PI      3.14159265

void main();
void inverse(int);                          // obtain inverse
matrix of A
double Gchld(double,double,double);         // Generalized
fundamental solution
double Gchld_x(double,double,double);       // dG/dx
double f_ini(double);                       // initial
condition

int N;                              // number of element
double L;                           // half length of region
double x[1001];                     // x-coordinate
double dx;                          // length of element
double t;                           // time
double dt;                          // time step
int it;                             // time step number
int iend;                           // end of calculation
int itvl;                           // interval for saving data
double U;                           // advection velocity
double kpp;                         // diffusion constant
double C[30001][1001];              // density of substance
double Dt_C[30001][1001];           // dC/dt
double C0;                          // C0 = C(-L,t)
double C1;                          // C1 = C(+L,t)
double gam;                         // scale of generalized
fundamental solution Gchld
double alp;                         // artificial damping
coeff
int NItg;                           // number of division for
numerical integration
double AMAT[1001][1001];            // matrix
FILE *fp_inp;                       // pointer of input file
FILE *fp_out;                       // pointer of output file
char InputDataFile[80];             // input file name
char OutputDataFile[80];            // output file name
char buf[500];                      // buffer for texts
double vec[1001];                   // temporary vector
double vec1[1001];                  // temporary vector
double tmp[401];                    // temporary array

void main()
{
    int i, j, p;

    sprintf(InputDataFile, "GIRM1Step_inp.dat"); // Input
file
    if ((fp_inp = fopen(InputDataFile, "r")) == NULL) {
        printf("Failed in Reading Input Data File! ... %s\n",
InputDataFile);
        exit(1);
    }

    fscanf(fp_inp, "%s %lf", buf, &kpp);
    fscanf(fp_inp, "%s %lf", buf, &L);
    fscanf(fp_inp, "%s %d", buf, &N);
    fscanf(fp_inp, "%s %lf", buf, &C0);
    fscanf(fp_inp, "%s %lf", buf, &C1);
    fscanf(fp_inp, "%s %lf", buf, &dt);
    fscanf(fp_inp, "%s %d", buf, &iend);
    fscanf(fp_inp, "%s %d", buf, &itvl);
    fscanf(fp_inp, "%s %lf", buf, &U);
    fscanf(fp_inp, "%s %lf", buf, &gam);
    fscanf(fp_inp, "%s %lf", buf, &alp);
    fscanf(fp_inp, "%s %d", buf, &NItg);
    fscanf(fp_inp, "%s %d", buf, &ini_no);

    fclose(fp_inp);

    dx = 2*L/(N+0.0);

    printf("kpp   = %12.6f\n", kpp);
    printf("L     = %12.6f\n", L);
    printf("N     = %d\n", N);
    printf("C0    = %12.6f\n", C0);
    printf("C1    = %12.6f\n", C1);
    printf("dx    = %12.6f\n", dx);
    printf("dt    = %12.6f\n", dt);
    printf("iend = %d\n", iend);
    printf("itvl = %d\n", itvl);
```

```
        printf("U      = %12.6f\n", U);
        printf("gam    = %12.6f\n", gam);
        printf("alp      = %12.6g\n", alp);
        printf("NItg     = %d\n", NItg);
        printf("ini_no = %d\n", ini_no);

        // Output file
        sprintf(OutputDataFile, "GIRM1Step_out.csv");

        if ((fp_out = fopen(OutputDataFile, "w")) == NULL) {
            printf("Failed in Reading Output Data File! ... %s\n", OutputDataFile);
            exit(1);
        }

        fprintf(fp_out, "kpp =, %12.6f\n", kpp);
        fprintf(fp_out, "L =, %12.6f, N =, %d, dx =, %12.6f\n", L, N, dx);
        fprintf(fp_out, "C0 =, %12.6f, C1 =, %12.6f\n", C0, C1);
        fprintf(fp_out, "dt =, %12.6f, iend =, %d, itvl =, %d\n", dt, iend, itvl);
        fprintf(fp_out, "U =, %12.6f\n", U);
        fprintf(fp_out, "gam =, %12.6f\n", gam);
        fprintf(fp_out, "alp =, %12.6g\n", alp);
        printf("NItg     = %d\n", NItg);
        fprintf(fp_out, "ini_no =, %d\n", ini_no);
        fprintf(fp_out, "\n");

        for (i = 0; i < N; i++) {
            x[i] = -L+0.5*dx+(i+0.0)*dx;
            C[0][i] = f_ini(x[i]);
        }

        // inverse of matrix A
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++) {
                AMAT[i][j] = 0.0;
                for (p = 0; p < NItg; p++)
                    AMAT[i][j]                                    += Gchld(x[j]-dx/2.0+(p+0.5)*dx/(NItg+0.0),x[i],gam)*dx/(NItg+0.0);
            }

        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++) {
                if (i == j)
                    AMAT[i][N+j] = 1.0;
                else
                    AMAT[i][N+j] = 0.0;
            }

        inverse(N);   // inverse of matrix A

        for (it = 0; it < iend; it++) {
            for (i = 0; i < N; i++) {
                vec1[i] = 0.0;
                for (j = 0; j < N; j++)
                    vec1[i]                                    += C[it][j]*(U*Gchld_x(x[j],x[i],gam)+kpp*Gchld_xx(x[j],x[i],gam))*dx;

                vec1[i] += U*( - C[it][N-1]*Gchld(L,x[i],gam) + C[it][0]*Gchld(-L,x[i],gam) );
                vec1[i] += -kpp*C1*Gchld_x(+L,x[i],gam) + kpp*C0*Gchld_x(-L,x[i],gam);
                vec1[i]                                    += +kpp*(C1-C[it][N-1])/(0.5*dx)*Gchld(+L,x[i],gam)          - kpp*(C[it][0]-C0)/(0.5*dx)*Gchld(-L,x[i],gam);
            }

            for (i = 0; i < N; i++) {
                Dt_C[it][i] = 0.0;
                for (j = 0; j < N; j++)
                    Dt_C[it][i] += AMAT[i][N+j]*vec1[j];
            }

            if (it % 20 == 0) {
                printf("Solution it = %d\n", it);
                for (i = 0; i < N; i++)
                    printf("it          =          %4d,          i = %4d, %12.6f, %12.6f\n", it, i, Dt_C[it][i], C[it][i]);
            }

            for (i = 0; i < N; i++)
                C[it+1][i] = C[it][i]+Dt_C[it][i]*dt;

            // reduction of spurious oscillation
            for (i = 1; i < N-1; i++)
                tmp[i]                                    = C[it+1][i]+alp*(  (C[it+1][i+1]-2.0*C[it+1][i]+C[it+1][i-1])/dx/dx );
            for (i = 1; i < N-1; i++)
                C[it+1][i] = tmp[i];
        }

        fprintf(fp_out, "(1) Dt_C\n");
        fprintf(fp_out, "i, x, ");
        for (it = 0; it < iend; it++)
            if (it % itvl == 0)
                fprintf(fp_out, "t=%8.4f, ", (it+0.0)*dt);
        fprintf(fp_out, "\n");
        for (i = 0; i < N; i++) {
            fprintf(fp_out, "%d, %12.6f, ", i, x[i]);
            for (it = 0; it < iend; it++)
                if (it % itvl == 0)
                    fprintf(fp_out, "%12.6f, ", Dt_C[it][i]);
            fprintf(fp_out, "\n");
        }

        fprintf(fp_out, "\n");
        fprintf(fp_out, "(2) dC/dx\n");
        fprintf(fp_out, "i, x, ");
        for (it = 0; it <= iend; it++)
```

```
            if (it % itvl == 0)
                fprintf(fp_out, "t=%8.4f, ", (it+0.0)*dt);
        fprintf(fp_out, "\n");
        for (i = 1; i < N-1; i++) {
            fprintf(fp_out, "%d, %12.6f, ", i, x[i]);
            for (it = 0; it <= iend; it++)
                if (it % itvl == 0)
                    fprintf(fp_out, "%12.6f, ",
(C[it][i+1]-C[it][i-1])/2.0/dx);
            fprintf(fp_out, "\n");
        }

        fprintf(fp_out, "\n");
        fprintf(fp_out, "(3) C\n");
        fprintf(fp_out, "i, x, ");
        for (it = 0; it <= iend; it++)
            if (it % itvl == 0)
                fprintf(fp_out, "t=%8.4f, ", (it+0.0)*dt);
        fprintf(fp_out, "\n");
        for (i = 0; i < N; i++) {
            fprintf(fp_out, "%d, %12.6f, ", i, x[i]);
            for (it = 0; it <= iend; it++)
                if (it % itvl == 0)
                    fprintf(fp_out, "%12.6f, ", C[it][i]);
            fprintf(fp_out, "\n");
        }

        fprintf(fp_out, "\n");
        fclose(fp_out);
    }

    /* ---------------------------------------------------------------- */
    void inverse(int N)
    {
        int i, j, k, N1; N1 = N + N;
        double AIK, AKK;

        for (k = 0; k < N; k++) {
            AKK = AMAT[k][k];
            for (j = k; j < N1; j++)
                AMAT[k][j] = AMAT[k][j] / AKK;

            for (i = 0; i < N; i++) {
                if (i == k)
                    continue;
                AIK = AMAT[i][k];
                for (j = k; j < N1; j++)
                    AMAT[i][j] = AMAT[i][j] -
AIK*AMAT[k][j];
            }
        }
    }

    double Gchld(double x, double xsi, double gam)
    {
        return
1.0/sqrt(2.0*PI)/gam*exp(-(x-xsi)*(x-xsi)/2.0/gam/gam);
    }

    double Gchld_x(double x, double xsi, double gam)
    {
        return
-(x-xsi)/sqrt(2.0*PI)/pow(gam,3.0)*exp(-(x-xsi)*(x-xsi)/2.0/g
am/gam);
    }

    double Gchld_xx(double x, double xsi, double gam)
    {
        return
-1.0/sqrt(2.0*PI)/pow(gam,3.0)*exp(-(x-xsi)*(x-xsi)/2.0/gam
/gam)

+(x-xsi)*(x-xsi)/sqrt(2.0*PI)/pow(gam,5.0)*exp(-(x-xsi)*(x-
xsi)/2.0/gam/gam);
    }

    double f_ini(double xx)
    {
        return exp(-pow(xx/(L/8.0),2.0)/2.0);
    }
```

A.3. FORTRAN code

```
      integer n, a(100), m, k
      real*8 L, tt(100)
      real*8 x(1001)
      real*8 dx
      real*8 t
      real*8 dt
      integer it, iend, itvl
      real*8 u
      real*8 kpp
      real*8 c(30001,1001)
      real*8 dtc(30001,1001)
      real*8 dcdx(30001,1001)
      real*8 c0, c1
      real*8 gam
      real*8 alp
      integer nitg
      integer i, j, p
      real*8 amat(1001,1001)
      real*8 vec(1001)
      real*8 vec1(1001)
      real*8 tmp(401)
      real*8 sum
c
      open(11,file='input2.dat')
      open(12,file='calcu2.csv')
c
      read(11,*) kpp
      read(11,*) L
      read(11,*) n
      read(11,*) c0
      read(11,*) c1
```

```
      read(11,*) dt
      read(11,*) iend
      read(11,*) itvl
      read(11,*) u
      read(11,*) gam
      read(11,*) alp
      read(11,*) nitg
c
      dx = 2.0d0*L/(n + 0.0d0)
c
      do 100 i = 1, n
        x(i) = -L + 0.5d0*dx + (i-1.0+0.0d0)*dx
        c(1,i) =   fini(x(i), L)
  100 continue
c
      do i = 1, n
        do j = 1, n
          amat(i,j) = 0.0d0
c          sum = 0.0d0
        end do
      end do
c
      do 300 i = 1, n
        do j = 1, n
          sum = 0.0d0
          do p = 1, nitg
            amat(i,j) = Gchld(x(j)-dx/2.0d0
     1
+(p-1.0+0.5)*dx/(nitg+0.0d0),x(i),gam)*dx/(nitg+0.0d0)
            sum = sum + amat(i,j)
          end do
          amat(i,j) = sum
        end do
  300 continue
c
      do 400 i = 1, n
        do j = 1, n
          if (i.eq.j) then
            amat(i,n+j) = 1.0d0
          else
            amat(i,n+j) = 0.0d0
          end if
        end do
  400 continue
c
      call inverse (n, amat)
c
      do 500 it = 1, iend
        do i = 1, n
          vec1(i) = 0.0d0
          do j = 1, n
            vec1(i) = vec1(i) +
c(it,j)*(u*Gchldx(x(j),x(i),gam)
     *          + kpp*Gchldxx(x(j),x(i),gam))*dx
          end do
            vec1(i) = vec1(i) +
u*(-c(it,n)*Gchld(L,x(i),gam)
     *          + c(it,1)*Gchld(-L,x(i),gam))
          vec1(i) = vec1(i) -
kpp*c1*Gchldx(L,x(i),gam)
     *          + kpp*c0*Gchldx(-L,x(i),gam)
          vec1(i) = vec1(i) + kpp*(c1-c(it,n))/(0.5*dx)
     1          * Gchld(L,x(i),gam)
     2                                              -
kpp*(c(it,1)-c0)/(0.5*dx)*Gchld(-L,x(i),gam)
        end do
c
        do i = 1, n
          dtc(it,i) = 0.0d0
          do j = 1, n
            dtc(it,i) = dtc(it,i) + amat(i,n+j)*vec1(j)
          end do
        end do
c
        do i = 1, n
          c(it+1,i) = c(it,i) + dtc(it,i)*dt
        end do
c
        do i = 2, n-1
          tmp(i) = c(it+1,i) + alp*((c(it+1,i+1)
     *          -2.0d0*c(it+1,i)+c(it+1,i-1))/dx/dx)
        end do
c
        do i = 2, n-1
          c(it+1,i) = tmp(i)
          dcdx(it,i) = (c(it,i+1) - c(it,i-1))/2.0/dx
        end do
c
  500 continue
c=========================
c     Output
c=========================
        m = iend/itvl
        a(1) = 1
        tt(1) = 0.0
c
        do k = 2, m+1
         a(k) = itvl*(k - 1)
         tt(k) = dt*a(k)
        end do
c--
        write(12,99)
   99 format(1x,'Dt_c')
        write(12,102) (tt(k),k=1,m)
  102 format(' ,','x',',','t =',f4.1,',','t =',f4.1,',','t =',f4.1,
     1          ',','t =',f4.1,',','t =',f4.1)
        do i = 1, n
          write(12,1111) i, x(i), dtc(a(1),i),
     1                        (dtc(a(k),i),k=2,m)
        end do
c--
        write(12,98)
   98 format(1x,'dC/dx')
        write(12,101) (tt(k),k=1,m+1)
```

```
     101 format(' ,','x',',','t =',f4.1,',','t =',f4.1,',','t =',f4.1,
        1                ',','t =',f4.1,',','t =',f4.1,',','t =',f4.1)
         do i = 1, n
           write(12,1111) i, x(i), dcdx(a(1),i),
        1                          (dcdx(a(k),i),k=2,m+1)
         end do
   c--
          write(12,97)
      97 format(1x,'C')
         write(12,101) (tt(k),k=1,m+1)
         do i = 1, n
           write(12,1111)      i,      x(i),      c(a(1),i),
(c(a(k),i),k=2,m+1)
         end do
   c
    1111 format(i5,',',f9.2,',',6(f15.6,','))
   c
         stop
         end
   c-------------------------------------------------------------
         subroutine inverse (n, mat)
   c-------------------------------------------------------------
         integer n, i, j, k, n1
         real*8 aik, akk
         real*8 mat(1001,1001)
   c
         n1 = n + n
   c
         do 100 k = 1, n
           akk = mat(k,k)
           do 200 j = k, n1
             mat(k,j) = mat(k,j)/akk
     200   continue
   c
           do 300 i = 1, n
             if (i.eq.k) go to 300
               aik = mat(i,k)
               do 400 j = k, n1
                 mat(i,j) = mat(i,j) - aik*mat(k,j)
     400       continue
     300   continue
     100 continue
   c
         return
         end
   c-------------------------------------------------------------
         function Gchld(x1, xsi1, gam1)
   c-------------------------------------------------------------
         real*8 x1, xsi1, gam1
         pi = 3.14159265
         Gchld = 1.0d0/sqrt(2.0d0*pi)/gam1
        1
*exp(-(x1-xsi1)*(x1-xsi1)/2.0d0/gam1/gam1)
         return
         end
   c-------------------------------------------------------------
         function Gchldx(x2, xsi2, gam2)
   c-------------------------------------------------------------
         real*8 x2, xsi2, gam2
         pi = 3.14159265
         Gchldx = -(x2-xsi2)/sqrt(2.0d0*pi)/gam2**3.0
        1
*exp(-(x2-xsi2)*(x2-xsi2)/2.0d0/gam2/gam2)
         return
         end
   c-------------------------------------------------------------
         function Gchldxx(x3, xsi3, gam3)
   c-------------------------------------------------------------
         real*8 x3, xsi3, gam3
         pi = 3.14159265
         Gchldxx = -1.0d0/sqrt(2.0d0*pi)/gam3**3.0
        1
*exp(-(x3-xsi3)*(x3-xsi3)/2.0d0/gam3/gam3)
         2
+(x3-xsi3)*(x3-xsi3)/sqrt(2.0d0*pi)/gam3**5.0
         3
*exp(-(x3-xsi3)*(x3-xsi3)/2.0d0/gam3/gam3)
         return
         end
   c-------------------------------------------------------------
         function fini(xx, L)
   c-------------------------------------------------------------
         real*8 xx, L
         real*8 aa1
            aa1   = ( xx/(L/8.0d0) )**2.0/2.0
            fini = exp( -aa1 )
         return
         end
   c-------------------------------------------------------------
-
```

Appendix B. Computer codes of 2-Step GIRM

B.1. Matlab code

```
   function GIRM_2_step ( )
   % Solves 1D IBVP for advective diffusion using 2-step
GIRM.
   L = 4.;                    % half length of region
   N = 40;                    % number of division in
x-direction
   itend = 3000;              % end of calculation
   itvl = 500;                % interval of print out
   dt = 0.0005;               % time indrement
   kpp = 0.05;                % diffusion coefficient
   U = 1.5;                   % velocity of flow
   C0 = 0.;                   % C at x = -L
   C1 = 0.;                   % C at x = +L
   NItg = 3;                  % number of division for
numerical integration
   gam = 0.15;                % scale of generalized
fundamental solution Gchld
   alp = 0.;                  % artificial damping coefft
```

```
dx = 2.0*L/(N+0.0);
for i = 1:N
    x(i) = -L + 0.5*dx + (i-1)*dx;
    C(1,i) = f_ini(x(i), L);
end

% Obtain inverse of kernel matrix A
for i = 1:N
    for n = 1:N
        AMAT(i,n) = 0.0;
        for p = 1:NItg
            AMAT(i,n) = AMAT(i,n) + Gchld(x(n)-dx/2.0+(p-0.5)*dx/(NItg+0.0),x(i),gam)*dx/(NItg+0.0);
        end
    end
end

for i = 1:N
    for j = 1:N
        if (i == j)
            AMAT(i,N+j) = 1.;
        else
            AMAT(i,N+j) = 0.;
        end
    end
end

% Inverse of matrix A
A1 = inverse(AMAT, N);

% Obtain solution at time t = it*dt
for it = 1:itend

    % Obtain theta(i) at it
    for i = 1:N
        vec0(i) = 0.;
        for n = 1:N
            vec0(i) = vec0(i) - C(it,n)*Gchld_x(x(n),x(i),gam)*dx;
        end
        vec0(i) = vec0(i) + C1*Gchld(+L,x(i),gam) - C0*Gchld(-L,x(i),gam);
    end

    for i = 1:N
        vec1(i) = 0.;
        for j = 1:N
            vec1(i) = vec1(i) + A1(i,N+j)*vec0(j);
        end
    end

    for i = 1:N
        theta(i) = vec1(i);
    end

    % Obtain qq(i) at it
    for i = 1:N
        qq(i) = -kpp*theta(i);
    end

    % Obtain DtC at it
    for i = 1:N
        vec0(i) = 0.;
        for n = 1:N
            vec0(i) = vec0(i) + qq(n)*Gchld_x(x(n),x(i),gam)*dx;
        end

        for n = 1:N
            vec0(i) = vec0(i) + U*C(it,n)*Gchld_x(x(n),x(i),gam)*dx;
        end

        vec0(i) = vec0(i) + 2.0*kpp*(C1-C(it,N))/dx*Gchld(+L,x(i),gam) - 2.0*kpp*(C(it,1)-C0)/dx*Gchld(-L,x(i),gam);
        vec0(i) = vec0(i) - U*C(it,N)*Gchld(+L,x(i),gam) + U*C(it,1)*Gchld(-L,x(i),gam);
    end

    for i = 1:N
        vec1(i) = 0.;
        for j = 1:N
            vec1(i) = vec1(i) + A1(i,N+j)*vec0(j);
        end
    end

    for i = 1:N
        DtC(i) = vec1(i);
    end

    % Obtain C at it = it+1
    for i = 1:N
        C(it+1,i) = C(it,i) + DtC(i)*dt;
    end

    % Reduction of spurious oscillation
    for i = 2:N-1
        tmp(i) = C(it+1,i)+alp*( (C(it+1,i+1)-2.0*C(it+1,i)+C(it+1,i-1))/dx/dx );
    end

    for i = 2:N-1
        C(it+1,i) = tmp(i);
    end
end

% Plotting solution
plot(x,C(1:itvl:itend,:)); xlabel ( 'x' ); ylabel ( 'C(x,t)'); grid on
```

```
    end

    function A1 = inverse( AMAT, N)
        N1 = 0; N1 = N + N;

        for k = 1:N
            AKK = AMAT(k,k);
            for j = k:N1
                AMAT(k,j) = AMAT(k,j) / AKK;
            end

            for i = 1:N
                if (i == k)
                    continue;
                end
                AIK = AMAT(i,k);
                for j = k:N1
                    AMAT(i,j) = AMAT(i,j) -
AIK*AMAT(k,j);
                end
            end
        end

        A1 = AMAT;

    end

    function G = Gchld( x, xsi, gam)
        G = 1./sqrt(2.*pi)/gam*exp(-(x-xsi)^2/2./gam/gam);
    end

    function Gx = Gchld_x( x, xsi, gam)
        Gx =
-(x-xsi)/sqrt(2.*pi)/(gam^3)*exp(-(x-xsi)^2/2./gam/gam);
    end

    function f = f_ini( x, L);
        f = exp(-(x^2/(L/8.)^2)/2.);
    end
```

B.2. C code

```
    /* ---------------------------------------------------------- */
    /*  Solves the IBVP for the advective diffusion using
2-step GIRM  */
    /* ---------------------------------------------------------- */
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <math.h>
    #define PI      3.14159265

    void main();
    void inverse(int);          // Obtain inverse matrix of A
    double                      Gchld(double,double,double);
// Generalized fundamental solution
    double                      Gchld_x(double,double,double);
// dG/dx
    double                                  f_ini(double);
// initial condition

    double L;                   // half length of region
    int N;                      // number of division in
x-direction
    double x[401];                  // x-coordinate of
element center -L<x<L
    double dx;                  // L/N
    int it;                     // time step
    int itend;                  // end of calculation
    int itvl;                   // interval of print out
    double t;                   // time
    double dt;                  // time indrement
    int itvl0;                  // interval of saving C etc.
    double kpp;                 // diffusion coefficient
    double U;                   // velocity of flow
    double C0;                  // C at x = -L
    double C1;                  // C at x = +L
    int NItg;                   // number of divion for
numerical integration
    double C[401];              // density of substance
    double DtC[401];            // dw/dt
    double C_itvl0[1000][401];  // C to be saved
    double DtC_itvl0[1000][401];    // DtC to be saved
    double theta[401];          // dC/dx
    double qq[401];             // qq[i] = -kpp*theta[i]
    double theta_itvl0[1000][401];  // theta to be saved
    double gam;                 // scale of generalized
fundamental solution Gchld
    double alp;                 // artificial damping
coefft
    double AMAT[4001][8001];        // matrix

    double vec0[2001];          // vector
    double vec1[2001];          // vector
    FILE *fp_inp;               // pointer of input file
    FILE *fp_out;               // pointer of output file
    char InputDataFile[80];     // input file name
    char OutputDataFile[80];    // output file name
    char buf[500];              // buffer for texts
    double tmp[401];            // temporary array

    void main()
    {
        int i, j, n, p;

        // Input file
        sprintf(InputDataFile, "GIRM2Step_inp.dat");

        if ((fp_inp = fopen(InputDataFile, "r")) == NULL) {
            printf("Failed in Reading Input Data File! ... %s\n",
InputDataFile);
            exit(1);
        }
```

```
        fscanf(fp_inp, "%s %lf", buf, &L);
        fscanf(fp_inp, "%s %d", buf, &N);
        fscanf(fp_inp, "%s %lf", buf, &kpp);
        fscanf(fp_inp, "%s %lf", buf, &U);
        fscanf(fp_inp, "%s %lf", buf, &C0);
        fscanf(fp_inp, "%s %lf", buf, &C1);
        fscanf(fp_inp, "%s %lf", buf, &dt);
        fscanf(fp_inp, "%s %d", buf, &itend);
        fscanf(fp_inp, "%s %d", buf, &itvl);
        fscanf(fp_inp, "%s %d", buf, &itvl0);
        fscanf(fp_inp, "%s %lf", buf, &gam);
        fscanf(fp_inp, "%s %lf", buf, &alp);
        fscanf(fp_inp, "%s %d", buf, &NItg);
        fscanf(fp_inp, "%s %d", buf, &ini_no);

        fclose(fp_inp);

        dx = 2.0*L/(N+0.0);
        for (i = 0; i < N; i++) {
            x[i] = -L+0.5*dx+(i+0.0)*dx;
            C[i] = f_ini(x[i], ini_no);
        }

        printf("L        = %12.6f\n", L);
        printf("N        = %d\n", N);
        printf("kpp      = %12.6f\n", kpp);
        printf("U        = %12.6f\n", U);
        printf("C0       = %12.6f\n", C0);
        printf("C1       = %12.6f\n", C1);
        printf("dx       = %12.6f\n", dx);
        printf("dt       = %12.6f\n", dt);
        printf("itend    = %d\n", itend);
        printf("itvl     = %d\n", itvl);
        printf("itvl0    = %d\n", itvl0);
        printf("gam      = %12.6f\n", gam);
        printf("alp      = %12.6g\n", alp);
        printf("NItg     = %d\n", NItg);
        printf("ini_no = %d\n", ini_no);

        // Output file
        sprintf(OutputDataFile,
"BasicProb_Gauss2Step_out.csv");

        if ((fp_out = fopen(OutputDataFile, "w")) == NULL) {
            printf("Failed in Reading Output Data
File! ... %s\n", OutputDataFile);
            exit(1);
        }

        // input data
        fprintf(fp_out, "L =, %12.6f\n", L);
        fprintf(fp_out, "N =, %d, dx =, %12.6f\n", N, dx);
        fprintf(fp_out, "kpp =, %12.6f\n", kpp);
        fprintf(fp_out, "U =, %12.6f\n", U);
        fprintf(fp_out, "C0 =, %12.6f, C1 =, %12.6f\n", C0,
C1);
        fprintf(fp_out, "dt =, %12.6f\n", dt);
        fprintf(fp_out, "itend =, %d, itvl =, %d, itvl0 =, %d\n",
itend, itvl, itvl0);
        fprintf(fp_out, "gam =, %12.6f\n", gam);
        fprintf(fp_out, "alp =, %12.6g\n", alp);
        fprintf(fp_out, "NItg =, %d\n", NItg);
        fprintf(fp_out, "ini_no =, %d\n", ini_no);
        fprintf(fp_out, "\n");

        // obtain inverse of kernel matrix A
        for (i = 0; i < N; i++)
            for (n = 0; n < N; n++) {
                AMAT[i][n] = 0.0;
                for (p = 0; p < NItg; p++)
                    AMAT[i][n]                                    +=
Gchld(x[n]-dx/2.0+(p+0.5)*dx/(NItg+0.0),x[i],gam)*dx/(NIt
g+0.0);
            }

        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++) {
                if (i == j)
                    AMAT[i][N+j] = 1.0;
                else
                    AMAT[i][N+j] = 0.0;
            }

        inverse(N);

        for (i = 1; i < N-1; i++)
            theta_itvl0[0][i] = (C[i+1]-C[i-1])/2.0/dx;

        for (i = 0; i < N; i++)
            C_itvl0[0][i] = C[i];

        // obtain solution at time t = it*dt
        for (it = 0; it < itend; it++) {

            // obtain theta[i] at it
            for (i = 0; i < N; i++) {
                vec0[i] = 0.0;
                for (n = 0; n < N; n++)
                    vec0[i]                                       +=
-C[n]*Gchld_x(x[n],x[i],gam)*dx;

                vec0[i]                                           +=
+C1*Gchld(+L,x[i],gam)-C0*Gchld(-L,x[i],gam);
            }

            for (i = 0; i < N; i++) {
                vec1[i] = 0.0;
                for (j = 0; j < N; j++)
                    vec1[i] += AMAT[i][N+j]*vec0[j];
            }

            for (i = 0; i < N; i++)
                theta[i] = vec1[i];
```

```
            // obtain qq[i] at it
            for (i = 0; i < N; i++)
                qq[i] = -kpp*theta[i];

            // obtain DtC at it
            for (i = 0; i < N; i++) {
                vec0[i] = 0.0;
                for (n = 0; n < N; n++)
                    vec0[i]                                    +=
qq[n]*Gchld_x(x[n],x[i],gam)*dx;

                for (n = 0; n < N; n++)
                    vec0[i]                                    +=
U*C[n]*Gchld_x(x[n],x[i],gam)*dx;

                vec0[i]                                        +=
2.0*kpp*(C1-C[N-1])/dx*Gchld(+L,x[i],gam)                      -
2.0*kpp*(C[0]-C0)/dx*Gchld(-L,x[i],gam);

                vec0[i]  +=  -U*C[N-1]*Gchld(+L,x[i],gam)  +
U*C[0]*Gchld(-L,x[i],gam);
            }

            for (i = 0; i < N; i++) {
                vec1[i] = 0.0;
                for (j = 0; j < N; j++)
                    vec1[i] += AMAT[i][N+j]*vec0[j];
            }

            for (i = 0; i < N; i++)
                DtC[i] = vec1[i];

            // obtain C at it = it+1
            for (i = 0; i < N; i++)
                C[i] += DtC[i]*dt;

            // Reduction of spurious oscillation

            for (i = 1; i < N-1; i++)
                tmp[i]                                         =
C[i]+alp*( (C[i+1]-2.0*C[i]+C[i-1])/dx/dx );
            for (i = 1; i < N-1; i++)
                C[i] = tmp[i];

            if ((it+0) % itvl0 == 0)
                for (i = 1; i < N-1; i++)
                    DtC_itvl0[(it+0)/itvl0][i] = DtC[i];

            if ((it+1) % itvl0 == 0) {
                for (i = 1; i < N-1; i++)
                    theta_itvl0[(it+1)/itvl0][i] = theta[i];
                for (i = 0; i < N; i++)
                    C_itvl0[(it+1)/itvl0][i] = C[i];
            }
        }

        // DtC
        t = (itend-1.0)*dt;
        fprintf(fp_out, "DtC at it = %d (%12.6f\n", itend-1, t);
        fprintf(fp_out, "i, x, DtC\n");
        for (i = 0; i < N; i++)
            fprintf(fp_out, "%d, %12.6f, %12.6f\n", i, x[i],
DtC[i]);
        fprintf(fp_out, "\n");

        // theta
        t = (itend-1.0)*dt;
        fprintf(fp_out, "theta at it = %d (%12.6f\n", itend-1, t);
        fprintf(fp_out, "i, x, theta\n");
        for (i = 0; i < N; i++)
            fprintf(fp_out, "%d, %12.6f, %12.6f\n", i, x[i],
theta[i]);
        fprintf(fp_out, "\n");

        // C
        t = (itend+0.0)*dt;
        fprintf(fp_out, "C at it = %d (t = %12.6f)\n", itend, t);
        fprintf(fp_out, "i, x, C\n");
        for (i = 0; i < N; i++)
            fprintf(fp_out, "%d, %12.6f, %12.6f\n", i, x[i],
C[i]);
        fprintf(fp_out, "\n");

        // dC/dt
        fprintf(fp_out, "DtC\n");
        fprintf(fp_out, "i, x, ");
        for (j = 0; j < itend/itvl0; j++)
            fprintf(fp_out, "t=%12.6f, ", ((j+0.0)*itvl0)*dt);
        fprintf(fp_out, "\n");
        for (i = 0; i < N-1; i++) {
            fprintf(fp_out, "%d, %12.6f, ", i, x[i]);
            for (j = 0; j < itend/itvl0; j++)
                fprintf(fp_out, "%12.6f, ", DtC_itvl0[j][i]);
            fprintf(fp_out, "\n");
        }

        fprintf(fp_out, "\n");

        // theta
        fprintf(fp_out, "theta\n");
        fprintf(fp_out, "x, ");
        for (j = 0; j <= itend/itvl0; j++)
            fprintf(fp_out, "t=%12.6f, ", ((j+0.0)*itvl0)*dt);
        fprintf(fp_out, "\n");
        for (i = 1; i < N-1; i++) {
            fprintf(fp_out, "%12.6f, ", x[i]);
            for (j = 0; j <= itend/itvl0; j++)
                fprintf(fp_out, "%12.6f, ", theta_itvl0[j][i]);
            fprintf(fp_out, "\n");
        }
        fprintf(fp_out, "\n");

        // C
        fprintf(fp_out, "C\n");
```

```
        fprintf(fp_out, "x, ");
        for (j = 0; j <= itend/itvl0; j++)
            fprintf(fp_out, "t=%12.6f, ", ((j+0.0)*itvl0)*dt);
        fprintf(fp_out, "\n");
        for (i = 0; i < N; i++) {
            fprintf(fp_out, "%12.6f, ", x[i]);
            for (j = 0; j <= itend/itvl0; j++)
                fprintf(fp_out, "%12.6f, ", C_itvl0[j][i]);
            fprintf(fp_out, "\n");
        }
        fprintf(fp_out, "\n");
        fclose(fp_out);
    }

    void inverse(int N)
    {
        int i, j, k, N1;
        double AIK, AKK;

        N1 = N + N;

        for (k = 0; k < N; k++) {
            AKK = AMAT[k][k];
            for (j = k; j < N1; j++)
                AMAT[k][j] = AMAT[k][j] / AKK;

            for (i = 0; i < N; i++) {
                if (i == k)
                    continue;
                AIK = AMAT[i][k];
                for (j = k; j < N1; j++)
                    AMAT[i][j] = AMAT[i][j] -
AIK*AMAT[k][j];
            }
        }

    }

    double Gchld(double x, double xsi, double gam)
    {
        double r2;
        r2 = (x-xsi)*(x-xsi);
        return 1.0/sqrt(2.0*PI)/gam*exp(-r2/(2.0*gam*gam));
    }

    double Gchld_x(double x, double xsi, double gam)
    {
        double r2;
        r2 = (x-xsi)*(x-xsi);
        return
-(x-xsi)/sqrt(2.0*PI)/pow(gam,3.0)*exp(-r2/(2.0*gam*gam));
    }

    double f_ini(double xx)
    {
        return exp(-pow(xx/(L/8.0),2.0)/2.0);
    }
```

B.3. FORTRAN code

```
          integer n, a(100), m, k
          real*8 L, tt(100)
          real*8 x(401)
          real*8 dx
          real*8 t
          real*8 dt
          integer it, itend, itvl, itvl0
          real*8 u
          real*8 kpp
          real*8 c(401)
          real*8 dtc(401)
          real*8 citvl0(1000,401), dtcitvl0(1000,401)
          real*8 theta(401), qq(401)
          real*8 thetaitvl0(1000,401)
          real*8 c0, c1
          real*8 gam
          real*8 alp
          integer nitg, inino
          integer i, j, p
          real*8 amat(4001,8001)
          real*8 vec0(2001)
          real*8 vec1(2001)
          real*8 tmp(401)
    c
          open(11,file='input1.dat')
          open(12,file='calcu1.csv')
    c
          read(11,*) L
          read(11,*) n
          read(11,*) kpp
          read(11,*) u
          read(11,*) c0
          read(11,*) c1
          read(11,*) dt
          read(11,*) itend
          read(11,*) itvl
          read(11,*) itvl0
          read(11,*) gam
          read(11,*) alp
          read(11,*) nitg
    c
          dx = 2.0d0*L/(n + 0.0d0)
    c
          do 100 i = 1, n
            x(i) = -L + 0.5d0*dx + (i-1.0+0.0d0)*dx
            c(i) =   fini(x(i), L)
      100 continue
    c
          do 300 i = 1, n
            do j = 1, n
              amat(i,j) = 0.0d0
              do p = 1, nitg
                amat(i,j) = amat(i,j) + Gchld(x(j)-dx/2.0d0
         1
    +(p-1.0+0.5)*dx/(nitg+0.0d0),x(i),gam)*dx/(nitg+0.0d0)
```

```
            end do
          end do
  300 continue
c
      do 400 i = 1, n
        do j = 1, n
          if (i.eq.j) then
            amat(i,n+j) = 1.0d0
          else
            amat(i,n+j) = 0.0d0
          end if
        end do
  400 continue
c
      call inverse (n, amat)
c
       do 410 i = 2, n-1
         thetaitvl0(1,i) = ( c(i+1) - c(i-1) )/2.0/dx
  410 continue
c
      do 420 i = 1, n
        citvl0(1,i) = c(i)
  420 continue
c
      do 500 it = 1, itend
c
        do i = 1, n
          vec0(i) = 0.0d0
          do n1 = 1, n
            vec0(i) = vec0(i) -
c(n1)*Gchldx( x(n1),x(i),gam )*dx
          end do
          vec0(i) = vec0(i) + c1*Gchld( L,x(i),gam )
     1                      - c0*Gchld(-L,x(i),gam )
        end do
c
        do i = 1, n
          vec1(i) = 0.0d0
          do j = 1, n
            vec1(i) = vec1(i) + amat(i,n+j)*vec0(j)
          end do
        end do
c
        do i = 1, n
          theta(i) = vec1(i)
        end do
c
        do i = 1, n
          qq(i) = -kpp*theta(i)
        end do
c
        do i = 1, n
          vec0(i) = 0.0d0
          do j = 1, n
            vec0(i) = vec0(i) +
qq(j)*Gchldx( x(j),x(i),gam )*dx
          end do
          do j = 1, n
            vec0(i) = vec0(i) +
u*c(j)*Gchldx( x(j),x(i),gam )*dx
          end do
          vec0(i) = vec0(i) + 2.0d0*kpp*(c1 - c(n))/dx
     1                      *Gchld( L,x(i),gam )
     2                      - 2.0d0*kpp*(c(1) - c0)/dx
     3                      *Gchld(-L,x(i),gam )
c
          vec0(i) = vec0(i) - u*c(n)*Gchld( L,x(i),gam )
     1                      + u*c(1)*Gchld(-L,x(i),gam )
        end do
c
        do i = 1, n
          vec1(i) = 0.0d0
          do j = 1, n
            vec1(i) = vec1(i) + amat(i,n+j)*vec0(j)
          end do
        end do
c
        do i = 1, n
          dtc(i) = vec1(i)
        end do
c
        do i = 1, n
          c(i) = c(i) + dtc(i)*dt
        end do
c
        do i = 2, n-1
          tmp(i) = c(i) + alp*(( c(i+1)
     *                -2.0d0*c(i)+c(i-1) )/dx/dx)
        end do
c
        do i = 2, n-1
          c(i) = tmp(i)
        end do
c
        if ( it .eq. 1) then
          ita1 = it
          do i = 2, n-1
            dtcitvl0( ita1,i ) = dtc(i)
          end do
        else if ( mod( it+0,itvl0 ) .eq. 0   ) then
          ita1 = it
          itb1 = ita1/itvl0 + 1
          do i = 2, n-1
            dtcitvl0( itb1,i ) = dtc(i)
          end do
        end if
c
        if (it .eq. 1) then
          ita2 = it
          do i = 2, n-1
            thetaitvl0( ita2,i ) = theta(i)
          end do
        else if ( mod( it+1,itvl0 ) .eq. 0 ) then
          ita2 = it+1
```

```
            itb2 = ita2/itvl0 + 1
            do i = 2, n-1
              thetaitvl0( itb2,i ) = theta(i)
            end do
          end if
c
          if (it .eq. 1) then
            ita3 = it
            do i = 1, n
              citvl0( ita3,i ) = c(i)
            end do
          else if ( mod( it+1,itvl0 ) .eq. 0 ) then
            ita3 = it+1
            itb3 = ita3/itvl0 + 1
            do i = 1, n
              citvl0( itb3,i ) = c(i)
            end do
          end if
c
  500 continue
c=========================
c       Output
c=========================
          m = itend/itvl0
          a(1) = 1
          tt(1) = 0.0
c
        do k = 2, m+1
          a(k) = itvl0*(k - 1)
          tt(k) = dt*a(k)
        end do
c--
        write(12,96) itend-1, (itend-1)*dt
   96 format(1x,'Dtc at it =',i5,', ( t =',f7.4,' )')
        write(12,95)
   95 format('i,x,Dtc')
        do i = 1, n
          write(12,1111) i, x(i), dtc(i)
        end do
c--
        write(12,*)
        write(12,94) itend-1, (itend-1)*dt
   94 format(1x,'theta at it =',i5,', ( t =',f7.4,' )')
        write(12,93)
   93 format('i,x,theta')
        do i = 1, n
          write(12,1111) i, x(i), theta(i)
        end do
c--
        write(12,*)
        write(12,92) itend, itend*dt
   92 format(1x,'C at it =',i5,', ( t =',f7.4,' )')
        write(12,91)
   91 format('i,x,C')
        do i = 1, n
          write(12,1111) i, x(i), c(i)
        end do
c--
        write(12,99)
   99 format(1x,'Dt_c')
        write(12,102) (tt(k),k=1,m)
  102 format(' ,','x',',','t =',f4.1,',','t =',f4.1,',','t =',f4.1,
     1                ',','t =',f4.1,',','t =',f4.1)
        do i = 1, n
          write(12,1111) i, x(i), dtcitvl0(a(1),i),
     1                          ( dtcitvl0(k,i),k=2,m )
        end do
c--
        write(12,98)
   98 format(1x,'theta')
        write(12,101) (tt(k),k=1,m+1)
  101 format(' ,','x',',','t =',f4.1,',','t =',f4.1,',','t =',f4.1,
     1                ',','t =',f4.1,',','t =',f4.1,',','t =',f4.1)
        do i = 1, n
          write(12,1111) i, x(i), thetaitvl0(a(1),i),
     1                          ( thetaitvl0(k,i),k=2,m+1 )
        end do
c--
        write(12,97)
   97 format(1x,'C')
        write(12,101) (tt(k),k=1,m+1)
        do i = 1, n
          write(12,1111) i, x(i), citvl0(a(1),i),
     1                          ( citvl0(k,i),k=2,m+1 )
        end do
c
 1111 format(i5,',',f9.2,',',6(f15.6,','))
c
        stop
        end
c-------------------------------------------------------------
        subroutine inverse (n, mat)
c-------------------------------------------------------------
        integer n, i, j, k, n1
        real*8 aik, akk
        real*8 mat(4001,8001)
c
        nn = n + n
c
        do 100 k = 1, n
          akk = mat(k,k)
          do 200 j = k, nn
            mat(k,j) = mat(k,j)/akk
  200     continue
c
          do 300 i = 1, n
            if (i.eq.k) go to 300
              aik = mat(i,k)
              do 400 j = k, nn
                mat(i,j) = mat(i,j) - aik*mat(k,j)
  400         continue
  300     continue
  100 continue
c
```

```
        return
        end
  c-------------------------------------------------------------
        function Gchld(x1, xsi1, gam1)
  c-------------------------------------------------------------
        real*8 x1, xsi1, gam1
        pi = 3.14159265
        Gchld = 1.0d0/sqrt(2.0d0*pi)/gam1
       1
*exp(-(x1-xsi1)*(x1-xsi1)/2.0d0/gam1/gam1)
        return
        end
  c-------------------------------------------------------------
        function Gchldx(x2, xsi2, gam2)
  c-------------------------------------------------------------
        real*8 x2, xsi2, gam2
        pi = 3.14159265
        Gchldx = -(x2-xsi2)/sqrt(2.0d0*pi)/gam2**3.0
       1
*exp(-(x2-xsi2)*(x2-xsi2)/2.0d0/gam2/gam2)
        return
        end
  c-------------------------------------------------------------
        function fini(xx, L)
  c-------------------------------------------------------------
        real*8 xx, L
        real*8 aa1
           aa1   = ( xx/(L/8.0d0) )**2.0/2.0
           fini = exp( -aa1 )
        return
        end
```

References

[1] H. Isshiki, "From Integral Representation Method (IRM) to Generalized Integral Representation Method (GIRM)," Applied and Computational Mathematics, Special Issue: Integral Representation Method and Its Generalization, (2015), under publication. http://www.sciencepublishinggroup.com/journal/archive.aspx?journalid=147&issueid=-1

[2] H. Isshiki, T. Takiya, and H. Niizato, "Application of Generalized Integral representation (GIRM) Method to Fluid Dynamic Motion of Gas or Particles in Cosmic Space Driven by Gravitational Force," Applied and Computational Mathematics, Special Issue: Integral Representation Method and Its Generalization, (2015), under publication. http://www.sciencepublishinggroup.com/journal/archive.aspx?journalid=147&issueid=-1

[3] H. Isshiki, "Effects of Generalized Fundamental Solution (GFS) on Generalized Integral Representation Method (GIRM)," Applied and Computational Mathematics, Special Issue: Integral Representation Method and Its Generalization, (2015), under publication. http://www.sciencepublishinggroup .com/journal/archive.aspx?journalid=147&issueid=-1

[4] H. Isshiki, "Application of the Generalized Integral Representation Method (GIRM) to Tidal Wave Propagation," Applied and Computational Mathematics, Special Issue: Integral Representation Method and Its Generalization, (2015), under publication. http://www.sciencepublishinggroup.com/journal/archive.aspx?journalid=147&issueid=-1