```fortran
!  ============================================================
!      Stabilized Finite Element Analysis for
!        1-Dimensional Advection Equation
!
!      * Stabilized F.E.M. ( SUPG )
!      * Crank-Nicolson Method ( Selectable )
!      * E-by-E Bi-CGSTABSolver
!                            2001.08.21. coded by Seizo Tanaka
!     ++++++ Transport ++++++
!      * E-by-E GP-BiCG Solver
!      * Fortran90 Ver.
!                            2005.11.03. coded by Seizo Tanaka
!  ============================================================
!-- MODULES --
module datacm
      integer :: node, nelm
      double precision :: dt, alpha
      integer, allocatable :: nc(:,:)
      double precision, allocatable :: xx(:), elength(:)
      double precision, allocatable :: uu(:), phi(:), dphi(:), bphi(:)
      double precision, allocatable :: ea(:,:,:), diag(:)
      integer :: iphbc
      integer, allocatable :: nphbc(:)
      double precision, allocatable :: fphbc(:)
      integer :: isfem
end module datacm
!-- MODULES --
!
!
!-- MAIN PROGRAM -
program advection_1D
      use datacm, only: dt
      implicit none
      integer :: istep, iend, iout
      double precision :: time
!
      call datain( iend, iout )
!
      do istep = 1, iend        ! ========= TIME STEP LOOP
       call solphi ( istep )
       if ( istep / iout * iout == istep ) then
        time = dt * dble(istep)
        call output(time)
       endif
      enddo                     ! ========= TIME STEP LOOP
end program advection_1D
!-- MAIN PROGRAM --
!
!
!-- SUBROUTINES --
! ------------------------------------------------------------------------------
      subroutine datain ( iend, iout )
! ------------------------------------------------------------------------------
      use datacm
      implicit none
      integer,         intent(out)   :: iend, iout
      integer :: ndiv
      double precision :: dlength, ela, el, uin
      integer :: i, j, n, m
      double precision :: xdum
!
      character(50) :: inpfile, mesfile, bcfile, inifile, outfile
!
      open(9, file = 'file.dat', status = 'unknown' )
       read(9,'(a)')   inpfile
       read(9,'(a)')   mesfile
       read(9,'(a)')   bcfile
       read(9,'(a)')   inifile
       read(9,'(a)')   outfile
      close(9)
      open(10, file = inpfile, status = 'unknown')
      open(11, file = mesfile, status = 'unknown')
      open(12, file = bcfile,  status = 'unknown')
      open(13, file = inifile, status = 'unknown')
      open(50, file = outfile, status = 'unknown')
!
! READ INPUT DATA
      read(10,*) iend, iout
      read(10,*) dt,   alpha
      read(10,*) uin
      read(10,*) isfem
      close(10)
!
! READ MESH
      read(11,*) node, nelm
      allocate( xx(node) )
      allocate( nc(2,nelm), elength(nelm) )

      do n = 1, node
        read(11,*) i, xx(i)
      enddo
      do m = 1, nelm
        read(11,*) i, (nc(j,i), j=1,2)
      enddo
      close(11)
!
      do m = 1, nelm
        elength(m) = xx(nc(2,m)) - xx(nc(1,m))
      enddo

! Dynamic Memory Allocation
      allocate( uu(node), phi(node), dphi(node), bphi(node) )
      allocate( ea(2,2,nelm), diag(node) )

      uu(1:node) = uin
!
! DEFINE BOUNDARY CONDITION
      read(12,*) iphbc
      allocate( nphbc(iphbc), fphbc(iphbc) )
      do i = 1, iphbc
        read(12,*) j, nphbc(i), fphbc(i)
      enddo
      close(12)
!
! DEFINE INITIAL CONDITION
      do n = 1, node
        read(13,*) i, xdum, phi(n)
      enddo
      close(13)

! Output Initial Condition
      write(50,'(a)') '# Time=0.0'
      write(50,500) (n, xx(n), phi(n), n = 1, node)
 500  format(i7,2e15.6)
!
      end subroutine datain
!
! ------------------------------------------------------------------------------
      subroutine solphi ( istep )
! ------------------------------------------------------------------------------
      use datacm
      implicit none
      integer, intent(in) :: istep
      integer :: k
```

```fortran
!
      call mkmtvc( node, nelm, nc, elength, uu, dt, alpha, &
                   phi,  dphi, ea, diag, bphi, iphbc, nphbc, fphbc, isfem )
      call gpbicg &
      ( k, node, nelm, nc, dphi, ea, diag, bphi, iphbc, nphbc )
!
      phi = phi + dphi
!
      write(6,*) ' ==== Now Computing ==== '
      write(6,*) '    STEP =', istep
      write(6,*) '    TIME =', dt * dble(istep)
      write(6,*) 'GPBICG(k):',k
!
      end subroutine solphi
!
! ----------------------------------------------------------------------
      subroutine mkmtvc &
                ( node, nelm, nc, el, uu, dt, alpha, &
                  phi,  dphi, ea, d, b, ibc, nbc, fbc, isfem )
! ----------------------------------------------------------------------
      implicit none
      integer, intent(in) :: node, nelm, nc(2,nelm), ibc, nbc(ibc), isfem
      double precision, intent(in) :: el(nelm), fbc(ibc), uu(node), dt,alpha
      double precision, intent(inout) :: phi(node)
      double precision, intent(inout) :: ea(2,2,nelm), d(node)
      double precision, intent(out)   :: b(node), dphi(node)
!
      integer :: m, i, n1, n2
      double precision :: tau, u1, u2, dl, dl06, dl03
      double precision :: emu1, emu2
      double precision :: etg11, etg12, etg21, etg22
      double precision :: ets11, ets12, ets21, ets22
      double precision :: eag11, eag12, eag21, eag22, eag1, eag2
      double precision :: eas11, eas12, eas21, eas22, eas1, eas2, aau
!
      do i = 1, ibc
       phi(nbc(i)) = fbc(i)
      enddo
!
      b = 0.0d0
      d = 0.0d0
!
      do m = 1, nelm
       n1 = nc(1,m)
       n2 = nc(2,m)
       u1 = uu(n1)
       u2 = uu(n2)
       dl = el(m)
       tau = dl / ( u1 + u2 ) * dble(isfem)
       dl06 = dl / 6.0d0
       dl03 = dl06 * 2.0d0
!
       emu1 = ( 2.0d0 * u1 + u2 ) * dl06
       emu2 = ( u1 + 2.0d0 * u2 ) * dl06
!
       etg11 = dl03 / dt
       etg12 = dl06 / dt
       etg21 = dl06 / dt
       etg22 = dl03 / dt
!
       ets11 = - emu1 / ( dt*dl )
       ets12 = - emu2 / ( dt*dl )
       ets21 =   emu1 / ( dt*dl )
       ets22 =   emu2 / ( dt*dl )
!
       eag11 = - emu1 / dl
       eag12 =   emu1 / dl
       eag21 = - emu2 / dl
       eag22 =   emu2 / dl
!
       aau = u1 *  emu1 + u2 * emu2
       eas11 = aau * ( - 1.0d0 / dl ) * ( - 1.0d0 / dl )
       eas12 = aau * ( - 1.0d0 / dl ) * (   1.0d0 / dl )
       eas21 = aau * (   1.0d0 / dl ) * ( - 1.0d0 / dl )
       eas22 = aau * (   1.0d0 / dl ) * (   1.0d0 / dl )
!
       ea(1,1,m) = ( etg11 + ets11*tau ) + ( eag11 + eas11*tau ) * alpha
       ea(1,2,m) = ( etg12 + ets12*tau ) + ( eag12 + eas12*tau ) * alpha
       ea(2,1,m) = ( etg21 + ets21*tau ) + ( eag21 + eas21*tau ) * alpha
       ea(2,2,m) = ( etg22 + ets22*tau ) + ( eag22 + eas22*tau ) * alpha
!
       d(n1) = d(n1) + ea(1,1,m)
       d(n2) = d(n2) + ea(2,2,m)
!
       eag1 = eag11 * phi(n1) + eag12 * phi(n2)
       eag2 = eag21 * phi(n1) + eag22 * phi(n2)
       eas1 = eas11 * phi(n1) + eas12 * phi(n2)
       eas2 = eas21 * phi(n1) + eas22 * phi(n2)
!
       b(n1) = b(n1) - ( eag1 + eas1*tau )
       b(n2) = b(n2) - ( eag2 + eas2*tau )
      enddo
!
      d = 1.0d0 / dsqrt(dabs(d))
!
      do m = 1, nelm
       ea(1,1,m) = ea(1,1,m) * ( d(nc(1,m)) * d(nc(1,m)))
       ea(1,2,m) = ea(1,2,m) * ( d(nc(1,m)) * d(nc(2,m)))
       ea(2,1,m) = ea(2,1,m) * ( d(nc(2,m)) * d(nc(1,m)))
       ea(2,2,m) = ea(2,2,m) * ( d(nc(2,m)) * d(nc(2,m)))
      enddo
!
      dphi = 0.0d0
!
      end subroutine mkmtvc
!
! ----------------------------------------------------------------------
      subroutine gpbicg &
      ( k, node, nelm, nc, xx, ea, d, rr, ibc, nbc )
! ----------------------------------------------------------------------
      implicit none
      integer, intent(in)  :: node, nelm, nc(2,nelm), ibc, nbc(ibc)
      integer, intent(out) :: k
      double precision, intent(in) :: ea(2,2,nelm), d(node)
      double precision, intent(inout) :: xx(node), rr(node)
      double precision, allocatable :: pp(:), tt(:), rs(:), Ap(:), At(:)
      double precision, allocatable :: yy(:), uu(:), zz(:), ww(:)
      double precision :: rtr, rsr, rsr0, Att, Aty, At2, ydy, ydt
      double precision :: alph,beta, zeta, eta, pk, epsbtb
      integer :: kmax
      double precision :: eps
      data kmax / 1000 /
      data eps / 1.0d-06 /
!
      allocate( pp(node), tt(node), rs(node), Ap(node), At(node) )
      allocate( yy(node), uu(node), zz(node), ww(node) )
!
      rr= rr * d
!
! Define Initial Guess
      call matvec ( node, nelm, nc, ibc, nbc, ea, xx, Ap )
      rr = rr - Ap
      rs = rr
      pp = rr
```

```fortran
      tt = 0.0d0
      ww = 0.0d0
      zz = 0.0d0
      uu = 0.0d0
      rtr = dot_product( rr, rr )
      rsr = rtr
      epsbtb = eps * dmax1(1.0d0,dsqrt(rtr))
      beta = 0.0d0
      pk = 0.0d0
!
!=================================================================
      do  k = 1, kmax
       if( dsqrt(rtr) <= epsbtb ) exit      ! Convergence Check!
! -----------------------------------------------------------
       call matvec ( node, nelm, nc, ibc, nbc, ea, pp, Ap )
       rsr0 = rsr
       alph = rsr / dot_product(rs,Ap)
       yy = tt - rr - alph * ww + alph * Ap
       uu = tt - tt + beta * uu
       tt = rr - alph * Ap
! -----------------------------------------------------------
       call matvec ( node, nelm, nc, ibc, nbc, ea, tt, At )
       Att = dot_product(At,tt)
       Aty = dot_product(At,yy)
       At2 = dot_product(At,At)
       ydy = dot_product(yy,yy)
       ydt = dot_product(yy,tt)
       zeta =(ydy*Att - ydt*Aty*pk ) / ( At2*ydy - Aty*Aty*pk )
       eta  =(ydt*At2 - Att*Aty    ) / ( At2*ydy - Aty*Aty ) * pk
       pk = 1.0d0
       uu = zeta * Ap + eta * uu
       zz = zeta * rr + eta * zz - alph * uu
       xx = xx + alph * pp + zz
       rr = tt - zeta * At - eta * yy
       rsr = dot_product(rs,rr)
       rtr = dot_product(rr,rr)
! -----------------------------------------------------------
       beta = alph / zeta * rsr / rsr0
       ww = At + beta * Ap
       pp = rr + beta * ( pp - uu )
      enddo
!=================================================================
      xx = xx * d
!
      deallocate( pp, tt, rs, Ap, At )
      deallocate( yy, uu, zz, ww )
!
      end subroutine gpbicg
!
! ---------------------------------------------------------------------------
      subroutine output(time)
! ---------------------------------------------------------------------------
      use datacm
      implicit none
      double precision, intent(in) :: time
      integer :: n
      character(50) :: cjunk
!
      dphi = phi
      do n = 1, node
       if( dabs( dphi(n) ) <= 1.0d-16 ) then
        dphi(n) = 0.0d0
       endif
      enddo
!
      write(50,*) ' '
      write(cjunk,'(e20.12)') time
      write(50,'(a,a)') '#Time=',trim(adjustl(cjunk))
      write(50,500) (n, xx(n), dphi(n), n = 1, node)
 500  format(i7,2e15.6)
!
      end subroutine output
! ---------------------------------------------------------------------------
      subroutine matvec ( node, nelm, nc, ibc, nbc, ea, xx, Ap )
! ---------------------------------------------------------------------------
      implicit none
      integer, intent(in) :: node, nelm, nc(2,nelm), ibc, nbc(ibc)
      double precision, intent(in) :: ea(2,2,nelm), xx(node)
      double precision, intent(inout) :: Ap(node)
      integer :: m, i, j
!
      Ap = 0.0d0
      do m = 1, nelm
       do i = 1, 2
        do j = 1, 2
         Ap(nc(i,m)) = Ap(nc(i,m)) + ea(i,j,m) * xx(nc(j,m))
        enddo
       enddo
      enddo
      do i = 1, ibc
       Ap(nbc(i)) = 0.0d0
      enddo
!
      end subroutine matvec
!
!----+------+------+------+------+------+------+------+------+------+----->
! ---------------------------------------------------------------------------
```