```
!
! To compile and run
!   pgf95 007.f95 -o 007.exe  && ./007.exe
!
!-------------------------
!
! time development of the temperature field.
!
!-------------------------
!

module constants
  implicit none
  integer,  parameter :: SP = kind(1.0)
  integer,  parameter :: DP = selected_real_kind(2*precision(1.0_SP))
  real(DP), parameter :: PI = 3.14159265358979323846264338327950 2_DP
  real(DP), parameter :: TWOPI = PI*2
  real(DP), parameter :: K_THERM = 1.0_DP  ! thermal diffusivity
end module constants

module ut
  use constants
  implicit none

contains

  subroutine ut__assert(condition, message)
    logical, intent(in) :: condition
    character(len=*), intent(in) :: message

    if ( .not.condition ) then
       stop message ! print out the message and die.
    end if
  end subroutine ut__assert
end module ut


module stopwatch
  !-------------------------------------------------------------------------------
---
  ! usage: see the sample code below.               Akira Kageyama, Kobe Univ. 2
010
  !-------------------------------------------------------------------------------
---
  !                          call stopwatch__strt(0)  ! whole exec time
  !                          call stopwatch__strt(1)  ! time for sub1
  ! call sub1
  !                          call stopwatch__stop(1)  ! time for sub1
  !                          call stopwatch__strt(2)  ! time for sub2
  ! call sub2
  !                          call stopwatch__stop(2)  ! time for sub2
  !                          call stopwatch__strt(3)  ! time for main_loop
  ! do loop = 1 , LOOP_MAX
  !    call main_loop
  ! end do
  !                          call stopwatch__stop(3)
  !                          call stopwatch__stop(0)
  !                          call stopwatch__print    ! print out info
  !-------------------------------------------------------------------------------
---
  use constants
  implicit none
```



```
  private
  public :: stopwatch__print, stopwatch__strt, stopwatch__stop
  integer, parameter :: max_watch_id = 100
  integer            :: used_watch_id_max=-1
  real(SP), dimension(0:max_watch_id) :: time_start_saved, time_total

contains

  subroutine stopwatch__print
    integer :: id
    do id = 0 , used_watch_id_max
       print *,'stopwatch:', id, time_total(id)
    end do
  end subroutine stopwatch__print

  subroutine stopwatch__strt(id)
    integer, intent(in) :: id
    logical :: firstcall = .true.
    if (firstcall) then
       time_total(:) = 0.0_DP
       firstcall = .false.
    end if
    if (id>used_watch_id_max) used_watch_id_max = id
    call cpu_time(time_start_saved(id))
  end subroutine stopwatch__strt

  subroutine stopwatch__stop(id)
    integer, intent(in) :: id
    real(SP) :: time_now, elapsed
    call cpu_time(time_now)
    elapsed = time_now - time_start_saved(id)
    time_total(id) = time_total(id) + elapsed
  end subroutine stopwatch__stop
end module stopwatch


module grid
  use constants
  implicit none
  private
  public :: grid__initialize, grid__d, grid__n

  integer  :: nx, ny
  real(DP) :: dx, dy

contains

  subroutine grid__initialize(xlength, ylength, nx_mesh, ny_mesh)
    real(DP), intent(in) :: xlength, ylength
    integer,  intent(in) :: nx_mesh, ny_mesh
    integer  :: i, j
    nx = nx_mesh
    ny = ny_mesh
    dx = xlength / real(nx,DP)
    dy = ylength / real(ny,DP)
  end subroutine grid__initialize

  function grid__n(which)
    character, intent(in) :: which
    integer :: grid__n

    select case (which)
```

```
    case ('x')
       grid__n = nx
    case ('y')
       grid__n = ny
    case default
       stop 'grid__n arg incorrect.'
    end select
  end function grid__n

  function grid__d(which)
    character, intent(in) :: which
    real(DP) :: grid__d

    select case (which)
    case ('x')
       grid__d = dx
    case ('y')
       grid__d = dy
    case default
       stop 'grid__d arg incorrect.'
    end select
  end function grid__d
end module grid


module field
  use constants
  use ut
  use stopwatch
  use grid
  implicit none
  private
  public :: field__initialize,  &
            field__statistics,  &
            field__time_advance
  public :: tmp, src

  real(DP), dimension(:,:), allocatable :: tmp, src

contains

  subroutine boundary_condition
    integer :: nx, ny
    nx = grid__n('x')
    ny = grid__n('y')
    !
    !   +--------(D)-------+
    !   |                  |
    !   |                  |
    !   |                  |
    !  (A)                (B)
    !   |                  |
    !   |                  |
    !   |                  |
    !   |                  |
    !   +--------(C)-------+
    !
    tmp(     0,0:ny) = 0.0_DP  ! (A)
    tmp(    nx,0:ny) = 0.0_DP  ! (B)
    tmp(1:nx-1,   0) = 0.0_DP  ! (C)
    tmp(1:nx-1,  ny) = 0.0_DP  ! (D)
  end subroutine boundary_condition
```



```
  subroutine field__initialize
    integer  :: nx, ny, i, j
    nx = grid__n('x')
    ny = grid__n('y')
    allocate(tmp(0:nx,0:ny)) ! temperature field
    allocate(src(0:nx,0:ny)) ! heat source field
    tmp(:,:) = 0.0_DP
    src(:,:) = 0.0_DP
    call iSet_src
    call iSet_tmp
    call boundary_condition

  contains

    subroutine iSet_src
      src(:,:) = 4.0_DP  ! constant heat source
    end subroutine iSet_src

    subroutine iSet_tmp
      integer  :: i, j
      real(DP) :: dx, dy, x, y, xlen, ylen
      dx = grid__d('x')
      dy = grid__d('y')
      xlen = dx*nx  ! here we suppose xpos(0) = xmin = 0.0
      ylen = dy*ny  ! here we suppose ypos(0) = ymin = 0.0
      do j = 0 , ny
         y = dy*j
         do i = 0 , nx
            x = dx*i
            tmp(i,j) = sin(PI*x/xlen)*sin(PI*y/ylen)
         end do
      end do
    end subroutine iSet_tmp
  end subroutine field__initialize

  recursive subroutine field__statistics(which)
    character(len=*), intent(in) :: which
    integer :: total_grid_points
    total_grid_points = grid__n('x') * grid__n('y')
    select case (which)
    case ('max')
       print *,' temp max: ', maxval(tmp(:,:))
    case ('min')
       print *,' temp min: ', minval(tmp(:,:))
    case ('mean')
       print *,' temp mean: ', sum(tmp(:,:)) / total_grid_points
    case default
       call field__statistics('max')
       call field__statistics('min')
       call field__statistics('mean')
    end select
  end subroutine field__statistics

  subroutine field__time_advance(dt,debug)
    real(DP), intent(in) :: dt
    character(len=*), intent(in), optional :: debug
    integer  :: i, j
    real(DP) :: dx, dy
    integer, save  :: nx, ny
    real(DP), save :: alpha_x, alpha_y, beta
    real(DP) :: beta2
```

```
    logical :: first_time = .true.  ! automatically 'save'd.
    real(DP), dimension(:,:), allocatable, save :: tmp0

                                        call stopwatch__strt(5)
    if ( first_time ) then
       nx = grid__n('x')
       ny = grid__n('y')
       dx = grid__d('x')
       dy = grid__d('y')
       alpha_x = K_THERM / (dx*dx)
       alpha_y = K_THERM / (dy*dy)
       beta = 2 * (alpha_x + alpha_y)
       allocate(tmp0(0:nx,0:ny))
       first_time = .false.
    end if

!----<origian form>-----------------------------------------------------------
!   tmp_new(i,j) = tmp(i,j)                                                 &
!                + ( alpha_x * ( tmp(i+1,j) - 2*tmp(i,j) + tmp(i-1,j) )  &
!                  + alpha_y * ( tmp(i,j+1) - 2*tmp(i,j) + tmp(i,j-1) )  &
!                  + src(i,j) ) * dt
!----</origian form>----------------------------------------------------------

    beta2 = 1.0_DP - beta*dt

    if ( present(debug) ) call iDebug
                                        call stopwatch__stop(5)
                                        call stopwatch__strt(6)
    tmp0(:,:) = tmp(:,:)     ! copy
                                        call stopwatch__stop(6)
                                        call stopwatch__strt(7)
    do j = 1 , ny-1
       do i = 1 , nx-1
          tmp(i,j) = ( alpha_x * ( tmp0(i+1,j) + tmp0(i-1,j) )     &
                     + alpha_y * ( tmp0(i,j+1) + tmp0(i,j-1) )     &
                     + src(i,j)                                    &
                     ) * dt                                        &
                     + beta2 * tmp0(i,j)
       end do
    end do
                                        call stopwatch__stop(7)

  contains

    subroutine iDebug
      select case (debug)
      case ('beta2')
         print *, '[debug] beta2 = ', beta2
      case default
         print *, '[debug]    dt = ',       dt
         print *, '[debug] alpha_x = ', alpha_x
         print *, '[debug] alpha_y = ', alpha_y
         print *, '[debug]   beta2 = ',   beta2
      end select
    end subroutine iDebug
  end subroutine field__time_advance
end module field


program main
  use constants
  use ut
```



```
  use stopwatch
  use grid
  use field
  implicit none

  integer  :: i, j, nx, ny
  integer  :: loop, loop_max = 10000
  real(DP) :: xlen = 1.0_DP ! meter
  real(DP) :: ylen = 1.0_DP ! meter

  real(DP) :: dt_critical, dt, dx, dy
                                        call stopwatch__strt(0)
                                        call stopwatch__strt(1)
  nx = 50
  ny = 50
  call ut__assert(nx>0 .and. ny>0, 'nx/ny out of range.')
  call grid__initialize(xlen,ylen,nx,ny)
  dx = grid__d('x')
  dy = grid__d('y');                    call stopwatch__stop(1)
                                        call stopwatch__strt(2)
  call field__initialize;           call stopwatch__stop(2)
                                        call stopwatch__strt(3)
  call field__statistics('all')
  dt_critical = 0.5_DP / ( K_THERM * ( 1/(dx*dx) + 1/(dy*dy) ) )
  dt = dt_critical*1.0_DP;              call stopwatch__stop(3)
                                        call stopwatch__strt(4)
  do loop = 1 , loop_max
     call field__time_advance(dt)
     if ( mod(loop,100)==0 ) call field__statistics('max')
  end do;                               call stopwatch__stop(4)
                                        call stopwatch__stop(0)
                                        call stopwatch__print
end program main
```

```
!
! To compile and run
!   pgf95 007.f95 -o 007.exe  && ./007.exe
!
!------------------------
!
! time development of the temperature field.
! print out 1d profile on ../data/
!
!------------------------
!

module constants
  implicit none
  integer,  parameter :: SP = kind(1.0)
  integer,  parameter :: DP = selected_real_kind(2*precision(1.0_SP))
  real(DP), parameter :: PI = 3.14159265358979323846264338327950288_DP
  real(DP), parameter :: TWOPI = PI*2
  real(DP), parameter :: K_THERM = 1.0_DP  ! thermal diffusivity
end module constants

module ut
  use constants
  implicit none

contains

  subroutine ut__assert(condition, message)
    logical, intent(in) :: condition
    character(len=*), intent(in) :: message

    if ( .not.condition ) then
       stop message ! print out the message and die.
    end if
  end subroutine ut__assert

  subroutine ut__print_1d_prof(n,f)
    integer,  intent(in) :: n
    real(DP), intent(in), dimension(0:n) :: f
    character(len=*), parameter :: base = "../data/1d-prof."
    integer :: i, counter = 0
    character(len=4) :: serial_num

    write(serial_num,'(i4.4)') counter
    open(10,file=base//serial_num)
    do i = 0 , n
       write(10,*) i, f(i)
    end do
    close(10)
    counter = counter + 1
  end subroutine ut__print_1d_prof

  subroutine ut__print_2d_prof(nx,ny,f)
    integer,  intent(in) :: nx, ny
    real(DP), intent(in), dimension(0:nx,0:ny) :: f
    integer                              :: counter = 0    ! saved
    integer                              :: ierr           ! use for MPI
    character(len=4)                     :: serial_num     ! put on file name
    character(len=*), parameter          :: base = "../data/2d-prof."
    integer :: i, j
    write(serial_num,'(i4.4)') counter
    open(10,file=base//serial_num)
```



```
    do j = 0 , ny
       do i = 0 , nx
          write(10,*) i, j, f(i,j)
       end do
       write(10,*)' ' ! gnuplot requires a blank line here.
    end do
    close(10)
    counter = counter + 1
  end subroutine ut__print_2d_prof
end module ut


module stopwatch
  !-------------------------------------------------------------------------
---
  ! usage: see the sample code below.               Akira Kageyama, Kobe Un
010
  !-------------------------------------------------------------------------
---
  !                           call stopwatch__strt(0)  ! whole exec time
  !                           call stopwatch__strt(1)  ! time for sub1
  ! call sub1
  !                           call stopwatch__stop(1)  ! time for sub1
  !                           call stopwatch__strt(2)  ! time for sub2
  ! call sub2
  !                           call stopwatch__stop(2)  ! time for sub2
  !                           call stopwatch__strt(3)  ! time for main_loop
  ! do loop = 1 , LOOP_MAX
  !    call main_loop
  ! end do
  !                           call stopwatch__stop(3)
  !                           call stopwatch__stop(0)
  !                           call stopwatch__print    ! print out info
  !-------------------------------------------------------------------------
---
  use constants
  implicit none
  private
  public :: stopwatch__print, stopwatch__strt, stopwatch__stop
  integer, parameter :: max_watch_id = 100
  integer            :: used_watch_id_max=-1
  real(SP), dimension(0:max_watch_id) :: time_start_saved, time_total

contains

  subroutine stopwatch__print
    integer :: id
    do id = 0 , used_watch_id_max
       print *,'stopwatch:', id, time_total(id)
    end do
  end subroutine stopwatch__print

  subroutine stopwatch__strt(id)
    integer, intent(in) :: id
    logical :: firstcall = .true.
    if (firstcall) then
       time_total(:) = 0.0_DP
       firstcall = .false.
    end if
    if (id>used_watch_id_max) used_watch_id_max = id
    call cpu_time(time_start_saved(id))
  end subroutine stopwatch__strt
```

```
  subroutine stopwatch__stop(id)
    integer, intent(in) :: id
    real(SP) :: time_now, elapsed
    call cpu_time(time_now)
    elapsed = time_now - time_start_saved(id)
    time_total(id) = time_total(id) + elapsed
  end subroutine stopwatch__stop
end module stopwatch


module grid
  use constants
  implicit none
  private
  public :: grid__initialize, grid__d, grid__n

  integer  :: nx, ny
  real(DP) :: dx, dy

contains

  subroutine grid__initialize(xlength, ylength, nx_mesh, ny_mesh)
    real(DP), intent(in) :: xlength, ylength
    integer,  intent(in) :: nx_mesh, ny_mesh
    integer  :: i, j
    nx = nx_mesh
    ny = ny_mesh
    dx = xlength / real(nx,DP)
    dy = ylength / real(ny,DP)
  end subroutine grid__initialize

  function grid__n(which)
    character, intent(in) :: which
    integer :: grid__n

    select case (which)
    case ('x')
       grid__n = nx
    case ('y')
       grid__n = ny
    case default
       stop 'grid__n arg incorrect.'
    end select
  end function grid__n

  function grid__d(which)
    character, intent(in) :: which
    real(DP) :: grid__d

    select case (which)
    case ('x')
       grid__d = dx
    case ('y')
       grid__d = dy
    case default
       stop 'grid__d arg incorrect.'
    end select
  end function grid__d
end module grid
```



```
module field
  use constants
  use ut
  use stopwatch
  use grid
  implicit none
  private
  public :: field__initialize,   &
            field__statistics,   &
            field__time_advance
  public :: tmp, src

  real(DP), dimension(:,:), allocatable :: tmp, src

contains

  subroutine boundary_condition
    integer :: nx, ny
    nx = grid__n('x')
    ny = grid__n('y')
    !
    !   +--------(D)-------+
    !   |                  |
    !   |                  |
    !   |                  |
    !  (A)                (B)
    !   |                  |
    !   |                  |
    !   |                  |
    !   |                  |
    !   +--------(C)-------+
    !
    tmp(     0,0:ny) = 0.0_DP  ! (A)
    tmp(    nx,0:ny) = 0.0_DP  ! (B)
    tmp(1:nx-1,   0) = 0.0_DP  ! (C)
    tmp(1:nx-1,  ny) = 0.0_DP  ! (D)
  end subroutine boundary_condition

  subroutine field__initialize
    integer  :: nx, ny, i, j
    nx = grid__n('x')
    ny = grid__n('y')
    allocate(tmp(0:nx,0:ny)) ! temperature field
    allocate(src(0:nx,0:ny)) ! heat source field
    tmp(:,:) = 0.0_DP
    src(:,:) = 0.0_DP
    call iSet_src
    call iSet_tmp
    call boundary_condition

  contains

    subroutine iSet_src
      src(:,:) = 4.0_DP  ! constant heat source
    end subroutine iSet_src

    subroutine iSet_tmp
      integer  :: i, j
      real(DP) :: dx, dy, x, y, xlen, ylen
      dx = grid__d('x')
      dy = grid__d('y')
      xlen = dx*nx  ! here we suppose xpos(0) = xmin = 0.0
```

```
      ylen = dy*ny  ! here we suppose ypos(0) = ymin = 0.0
      do j = 0 , ny
         y = dy*j
         do i = 0 , nx
            x = dx*i
            tmp(i,j) = sin(PI*x/xlen)*sin(PI*y/ylen)
         end do
      end do
    end subroutine iSet_tmp
  end subroutine field__initialize

  recursive subroutine field__statistics(which)
    character(len=*), intent(in) :: which
    integer :: total_grid_points
    total_grid_points = grid__n('x') * grid__n('y')
    select case (which)
    case ('max')
       print *,' temp max: ', maxval(tmp(:,:))
    case ('min')
       print *,' temp min: ', minval(tmp(:,:))
    case ('mean')
       print *,' temp mean: ', sum(tmp(:,:)) / total_grid_points
    case default
       call field__statistics('max')
       call field__statistics('min')
       call field__statistics('mean')
    end select
  end subroutine field__statistics

  subroutine field__time_advance(dt,debug)
    real(DP), intent(in) :: dt
    character(len=*), intent(in), optional :: debug
    integer  :: i, j
    real(DP) :: dx, dy
    integer, save  :: nx, ny
    real(DP), save :: alpha_x, alpha_y, beta
    real(DP) :: beta2
    logical :: first_time = .true.  ! automatically 'save'd.
    real(DP), dimension(:,:), allocatable, save :: tmp0

                                        call stopwatch__strt(5)
    if ( first_time ) then
       nx = grid__n('x')
       ny = grid__n('y')
       dx = grid__d('x')
       dy = grid__d('y')
       alpha_x = K_THERM / (dx*dx)
       alpha_y = K_THERM / (dy*dy)
       beta = 2 * (alpha_x + alpha_y)
       allocate(tmp0(0:nx,0:ny))
       first_time = .false.
    end if

!----<origian form>-----------------------------------------------------------
!   tmp_new(i,j) = tmp(i,j)                                                 &
!                + ( alpha_x * ( tmp(i+1,j) - 2*tmp(i,j) + tmp(i-1,j) )  &
!                  + alpha_y * ( tmp(i,j+1) - 2*tmp(i,j) + tmp(i,j-1) )  &
!                  + src(i,j) ) * dt
!----</origian form>----------------------------------------------------------

    beta2 = 1.0_DP - beta*dt
```



```
    if ( present(debug) ) call iDebug
                                        call stopwatch__stop(5)
                                        call stopwatch__strt(6)
    tmp0(:,:) = tmp(:,:)      ! copy
                                        call stopwatch__stop(6)
                                        call stopwatch__strt(7)
    do j = 1 , ny-1
       do i = 1 , nx-1
          tmp(i,j) = ( alpha_x * ( tmp0(i+1,j) + tmp0(i-1,j) )     &
                     + alpha_y * ( tmp0(i,j+1) + tmp0(i,j-1) )     &
                     + src(i,j)                                    &
                     ) * dt                                        &
                     + beta2 * tmp0(i,j)
       end do
    end do
                                        call stopwatch__stop(7)

  contains

    subroutine iDebug
      select case (debug)
      case ('beta2')
         print *, '[debug] beta2 = ', beta2
      case default
         print *, '[debug]    dt = ',       dt
         print *, '[debug] alpha_x = ', alpha_x
         print *, '[debug] alpha_y = ', alpha_y
         print *, '[debug]  beta2 = ',    beta2
      end select
    end subroutine iDebug
  end subroutine field__time_advance
end module field


program main
  use constants
  use ut
  use stopwatch
  use grid
  use field
  implicit none

  integer  :: i, j, nx, ny
  integer  :: loop, loop_max = 10000
  real(DP) :: xlen = 1.0_DP ! meter
  real(DP) :: ylen = 1.0_DP ! meter

  real(DP) :: dt_critical, dt, dx, dy
                                        call stopwatch__strt(0)
                                        call stopwatch__strt(1)
  nx = 50
  ny = 50
  call ut__assert(nx>0 .and. ny>0, 'nx/ny out of range.')
  call grid__initialize(xlen,ylen,nx,ny)
  dx = grid__d('x')
  dy = grid__d('y');                    call stopwatch__stop(1)
                                        call stopwatch__strt(2)
  call field__initialize;             call stopwatch__stop(2)
                                        call stopwatch__strt(3)
  call field__statistics('all')
  call ut__print_1d_prof(nx,tmp(:,ny/2))
  call ut__print_2d_prof(nx,ny,tmp)
```

```
  dt_critical = 0.5_DP / ( K_THERM * ( 1/(dx*dx) + 1/(dy*dy) ) )
  dt = dt_critical*1.0_DP;                call stopwatch__stop(3)
                                          call stopwatch__strt(4)
  do loop = 1 , loop_max
     call field__time_advance(dt)
                                          call stopwatch__strt(8)
     if ( mod(loop,100)==0 ) call field__statistics('max')
     if ( mod(loop,100)==0 ) call ut__print_1d_prof(nx,tmp(:,ny/2))
     if ( mod(loop,10) ==0 ) call ut__print_2d_prof(nx,ny,tmp)
                                          call stopwatch__stop(8)
  end do;                                 call stopwatch__stop(4)
                                          call stopwatch__stop(0)
                                          call stopwatch__print
end program main
```

```
#
# temperature profile of x at j=ny/2
#

# set terminal png
# set output  '008-plot1d.png'

set xrange [0:50]
set yrange [0:1.1]
set xlabel 'i'
set ylabel 'temp at y=middle'
plot '../data/1d-prof.0000' w lp

pause -1
```

```
!
!  008-plot1d-anime-gp-generator.f95
!
! Usage:
!    (1) check the mesh size nx and the counter_end.
!    (2) pgf95 this_code
!    (3) ./a.out > anyname
!    (4) gnuplot anyname
!

program main
  implicit none
  integer, parameter :: nx = 50
  integer :: counter, counter_end = 50
  character(len=*), parameter :: base='../data/1d-prof.'
  character(len=4)            :: serial_num

  print *, "#"
  print *, "# gnuplot script generated 008-plot1d-anime-gp-generator.f95"
  print *, "#"
  print *, " "
  print *, "set xlabel 'j'          # x-axis"
  print *, "set ylabel 'temperature'    # y-axis"
  print *, "set xrange[0:", nx, "]     # i-grid min & max"
  print *, "set yrange[0.0:1.1]        # temperature min & max"

  do counter = 0 , counter_end
     write(serial_num,'(i4.4)') counter
     print *, "plot '"//base//serial_num//"' w lp"
     if ( counter==0) then
        print *, "pause 5"
     else
        print *, "pause 1"
     end if
  end do
  print *, "pause -1"
end program main
```

```
#
# a sample gnuplot script
#
#   [ line contours ]
#
# set terminal png
# set output 'test.png'

set size square            # same side lengths for x and y
set xlabel 'i'             # x-axis
set ylabel 'j'             # y-axis
set xrange[0:50]           # i-grid min & max
set yrange[0:50]           # j-grid min & max
set nosurface              # do not show surface plot
unset ztics                # do not show z-tics
set contour base           # enables contour lines
set cntrparam levels 10    # draw 10 contours
set view 0,0               # view from the due north
set title 'Temperature at  0000'
splot '../data/2d-prof.0100' using 1:2:3 w l # with lines
pause -1
```

```
#
# a sample gnuplot script
#
#   [ line contours ]
#
set terminal png
set output 'test.png'

set size square            # same side lengths for x and y
set xlabel 'i'             # x-axis
set ylabel 'j'             # y-axis
set xrange[0:50]           # i-grid min & max
set yrange[0:50]           # j-grid min & max
set contour base           # enables contour lines
set cntrparam levels 10    # draw 10 contours
set title 'Temperature at  0000 '
splot '../data/2d-prof.0000' using 1:2:3 w l
pause -1
```