```
 1  program problem
 2
 3      use iso_fortran_env, only: dp=>real64
 4      use linspace_mod
 5      use tdma
 6
 7      implicit none
 8      !output utility
 9      integer, parameter :: outfile = 16
10      integer, parameter :: outfile2 = 17
11
12      !enter problem you want to solve here
13      !problem 2
14      !we have a square of given side
15      !the phi function we are supposed to be getting is x^2-y^2
16      !boundary conditions are set accordingly
17
18      !declarations
19      integer :: i,j,n,m,count
20      real(dp), dimension(:,:),allocatable ::
          phi,phi_dash,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp
21      real(dp), dimension(:),allocatable :: y
22      real(dp), dimension(:),allocatable :: x
23      real(dp) :: error,De
24      character(2) :: sweep
25
26      !side of the squeare
27      De = 1.0_dp
28
29      !input for grid size and sweep direction
30      write(*,*) "Enter number of rows"
31      read(*,*) n
32      write(*,*) "Enter number of columns"
33      read (*,*) m
34      write(*,*) "Choose the sweep direction"
```

```
35      read(*,*) sweep
36
37      !allocate arrays
38      allocate(x(m),y(n),phi(n,m),phi_dash(n,m),gmae(n,m),gmaw(n,m),gman(n,m),gmas(n,m),Del_x(n,m),Del_y
          (n,m),delxe(n,m),delxw(n,m),delyn(n,m),delys(n,m),sc(n,m),sp(n,m))
39
40      !initialize count
41      count = 0
42
43      !problem defination
44      !initialize cell properties here
45      gmae = 1.0_dp
46      gmaw = 1.0_dp
47      gman = 1.0_dp
48      gmas = 1.0_dp
49      !>>cell dimensions
50      Del_x = De/(1.0_dp*(m-2))
51      Del_y = De/(1.0_dp*(n-2))
52      delxe = De/(1.0_dp*(m-2))
53      delxw = De/(1.0_dp*(m-2))
54      delyn = De/(1.0_dp*(n-2))
55      delys = De/(1.0_dp*(n-2))
56
57      !setting up cell dimensions for the boundary
58      do i = 1,m
59          !south boundary
60          delys(2,i) = 0.5_dp*delys(2,i)
61          !north boundary
62          delyn(n-1,i) = 0.5_dp*delyn(n-1,i)
63      end do
64      do i = 1,n
65         !west boundary
66         delxw(i,2) = 0.5_dp*delxw(i,2)
67         !east boundary
68         delxe(i,m-1) = 0.5_dp*delxe(i,m-1)
```

```
 69      end do
 70
 71      !initializing boundary conditions
 72      !initialize the 2D array phi here.
 73
 74      x(1) = 0.0_dp
 75      x(2:m-1) = linspace(0.0_dp+(Del_x(1,1)/2),1.0_dp-(Del_x(1,1)/2),m-2)
 76      x(m) = 1.0_dp
 77      y(1) = 0.0_dp
 78      y(2:n-1) = linspace(0.0_dp+(Del_y(1,1)/2),1.0_dp-(Del_y(1,1)/2),n-2)
 79      y(n) = 1.0_dp
 80
 81      phi = 0.0_dp
 82      do i = 1,m
 83          phi(1,i) = x(i)**2
 84          phi(n,i) = x(i)**2-1.0_dp
 85      end do
 86      do i = 1,n
 87          phi(i,1) = -1.0_dp*(y(i)**2)
 88          phi(i,m) = 1.0_dp - (y(i)**2)
 89      end do
 90
 91      !print input
 92      do i = 1,n
 93          print*, phi(i,:)
 94      end do
 95
 96      !main loop start
 97      !source term incorporation
 98      do while (.true.)
 99          !initialize error for source term loop
100          error = 0.0_dp
101          !count to keep track of iterations
102          count = count +1
103          print*,count, 'outer'
```

```
104          !backup array
105          phi_dash = phi
106          !declare source term here
107          sc = 0.0_dp
108          sp = 0.0_dp
109         !call twodtdma solver to solve your problem
110          call twodtdma(n,m,phi,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp,sweep)
111          !error using eucleadian distance
112          error = sqrt(sum((phi-phi_dash)**2))
113          !exit condition
114          if (error<1e-10) exit
115      end do
116
117      !print output
118      do i = 1,n
119         print*, phi(i,:)
120      end do
121
122      !deallocate all the arrays
123      deallocate(x,y,phi,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp,phi_dash)
124
125
126
127
128  end program problem
129
```