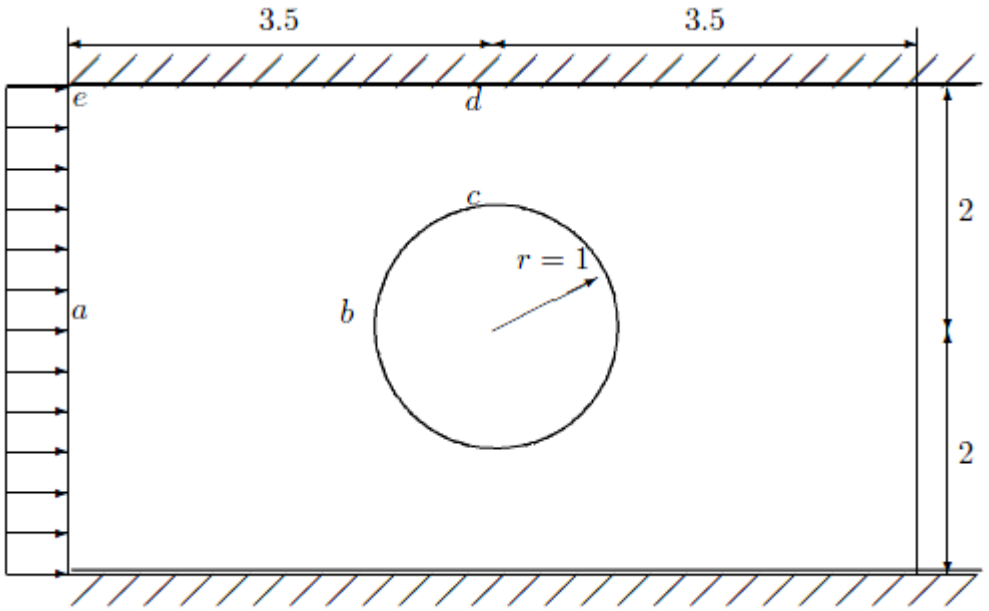
《计算流体力学基础》第五次作业

姓名：杨阳
学号：150*******
专业：流体力学

1. 问题描述

考虑位于两块无限长平板间的圆柱体的平面绕流问题，几何尺寸如下图所示，来流为$v_x = 1, v_y = 0$。由于流场具有上下左右的对称性，只考虑左上角四分之一的计算区域 abcde，

把它作为有限元的求解区域Ω。要求求解出整个区域中的流函数、v_x、v_y以及压强值。



物理问题的边界条件和满足的微分方程

1. 边界 ab 为流线，取 ψ=0，$\frac{\partial \varphi}{\partial n} = 0$;

2. 边界 bc 也为流线，同样取 ψ=0，$\frac{\partial \varphi}{\partial n} = 0$;

3. 边界 cd，切向速度v_τ=0，$\frac{\partial \psi}{\partial n} = 0$，取$\varphi = 0$;

4. 边界 de 为流线,满足 $\psi_e = \psi_a + \int_a^e v_x dy = \int_0^2 dy = 2$

于是在 ed 上， ψ=2，$\frac{\partial \varphi}{\partial n} = 0$;

5. 进口边界 ae 上， ψ=$\psi_a + \int_a^y v_x dy = y$（本文中采取此条件）

也可以提自然边界条件$\frac{\partial \psi}{\partial n} = 0$，$\frac{\partial \varphi}{\partial n} = v_x = 1$

我们以流函数 ψ 作为未知函数来解此问题，流函数所满足的微分方程如下：

$$\begin{cases}\nabla^2\psi = 0 \\ \psi|_{\Gamma_1} = \overline{\psi}\text{（本质边界条件）} \\ \frac{\partial\psi}{\partial n}|_{\Gamma_2} = -v_s\text{(自然边界条件)}\end{cases} \quad (1)$$

此处Γ_1指 ab, bc, de 和 ae 四段边界，而 Γ_2就是就是 cd 段边界，且切向速度v_s= 0， Γ1 和Γ2 合起来是整个边界，并且此二者不重合。

2. 计算步骤

有限元的计算过程可以分为生成单元刚度阵，单元刚度阵装配总刚度阵，处理本质边界条件，求解总刚度阵这几个步骤进行。

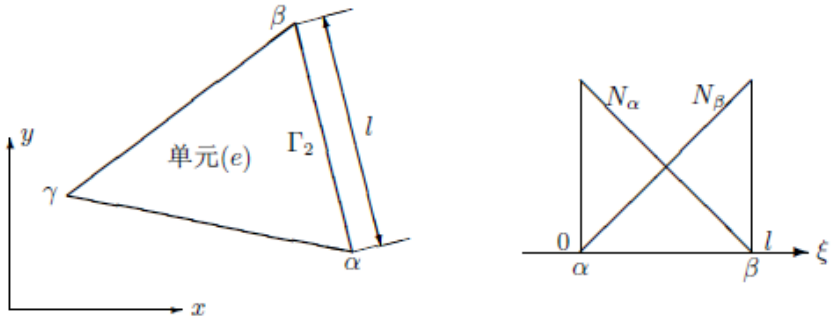
（1）生成单元刚度矩阵

$$A_{ij}^{(e)} = A_{ij}^{(e)}\begin{bmatrix} b_1^2 + c_1^2 & b_1b_2 + c_1c_2 & b_1b_3 + c_1c_3 \\ b_1b_2 + c_1c_2 & b_2^2 + c_2^2 & b_2b_3 + c_2c_3 \\ b_1b_3 + c_1c_3 & b_2b_3 + c_2c_3 & b_3^2+c_3^2 \end{bmatrix}$$

$$b_1 = \frac{1}{2A^{(e)}}(y_2 - y_3) \quad b_2 = \frac{1}{2A^{(e)}}(y_3 - y_1) \quad b_3 = \frac{1}{2A^{(e)}}(y_1 - y_2)$$

$$c_1 = \frac{1}{2A^{(e)}}(x_3 - x_2) \quad c_2 = \frac{1}{2A^{(e)}}(x_1 - x_3) \quad c_3 = \frac{1}{2A^{(e)}}(x_2 - x_1)$$

$$A_{ij}^{(e)} = \frac{1}{2}[(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)]$$



自然边界条件

$$f_\alpha^{(e)} = f_\beta^{(e)} = f_\gamma^{(e)} = 0$$

（2）装配总刚度阵

$$A = \sum_{e=1}^{N_e}\overline{A^{(e)}}, \qquad f = \sum_{e=1}^{N_e}\overline{f^{(e)}}。$$

这里的总刚度阵将单元节点的变量按照顺序装配在总体矩阵中就可以了，不需要其它额外的操作。

（3）处理自然边界条件

自然边界条件的处理采用惩罚因子的办法，将对应本质边界条件的节点的矩阵对角元和右向量乘以一个很大的因子并加到原来的总体刚度阵上，得到最终求解的总体刚度阵。在设计的程序中惩罚因子取10^8

（4）总刚度阵的求解

在经过了上面的步骤后问题就转化为一个求解大型稀疏对角阵的线性代数或者矩阵计算问题。因为这里问题的特点得到的总刚度阵是半正定的，因此可以考虑使用计算量更小的共轭梯度法或 Krylov 方法，在计算程序中求解部分使用共轭梯度法。

共轭梯度法计算过程如下：

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$

$\mathbf{p}_0 := \mathbf{r}_0$

$k := 0$

repeat

$\alpha_k := \dfrac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$

$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$

$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$

if r_{k+1} is sufficiently small then exit loop

$\beta_k := \dfrac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$

$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$

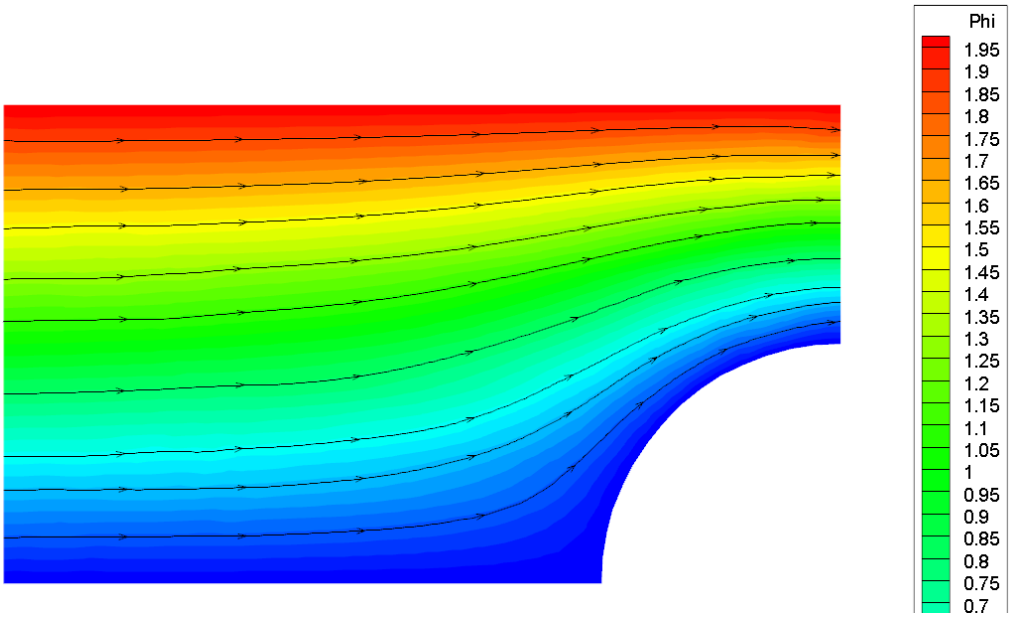
$k := k + 1$

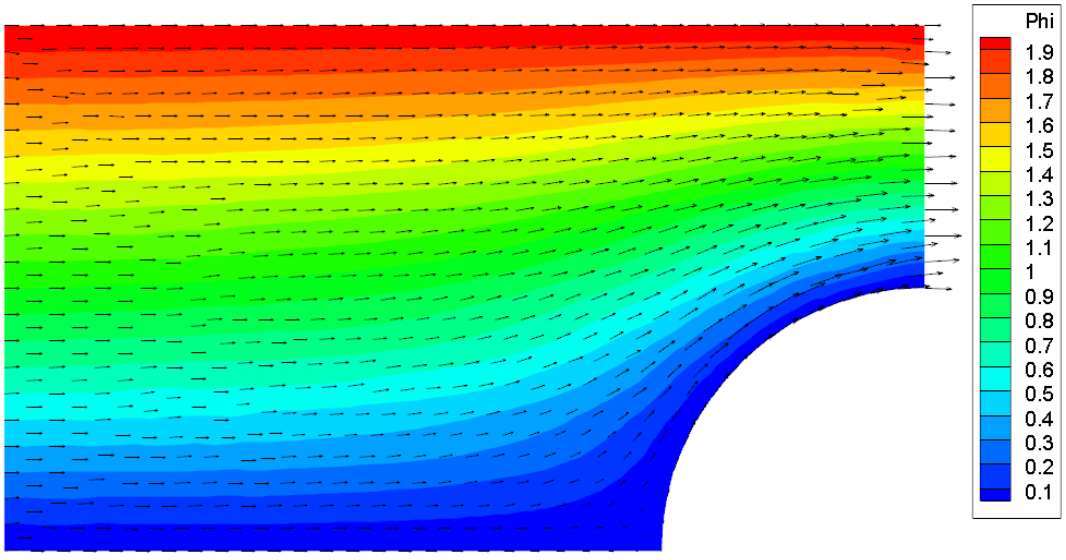
end repeat

The result is $\mathbf{x}_{k+1}$

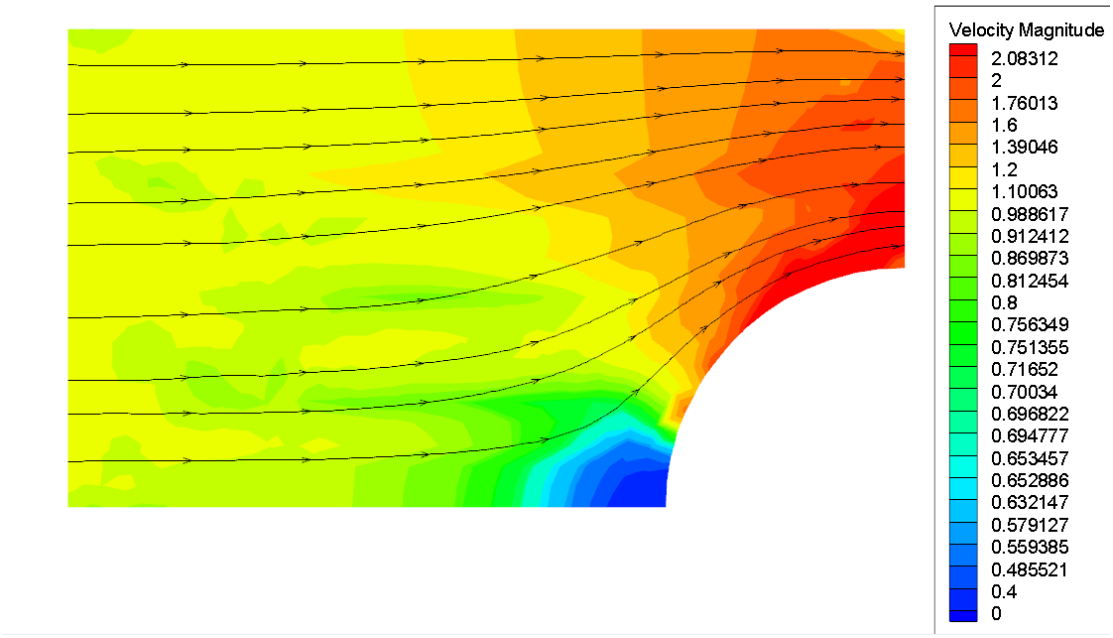
3. 计算结果

流函数：

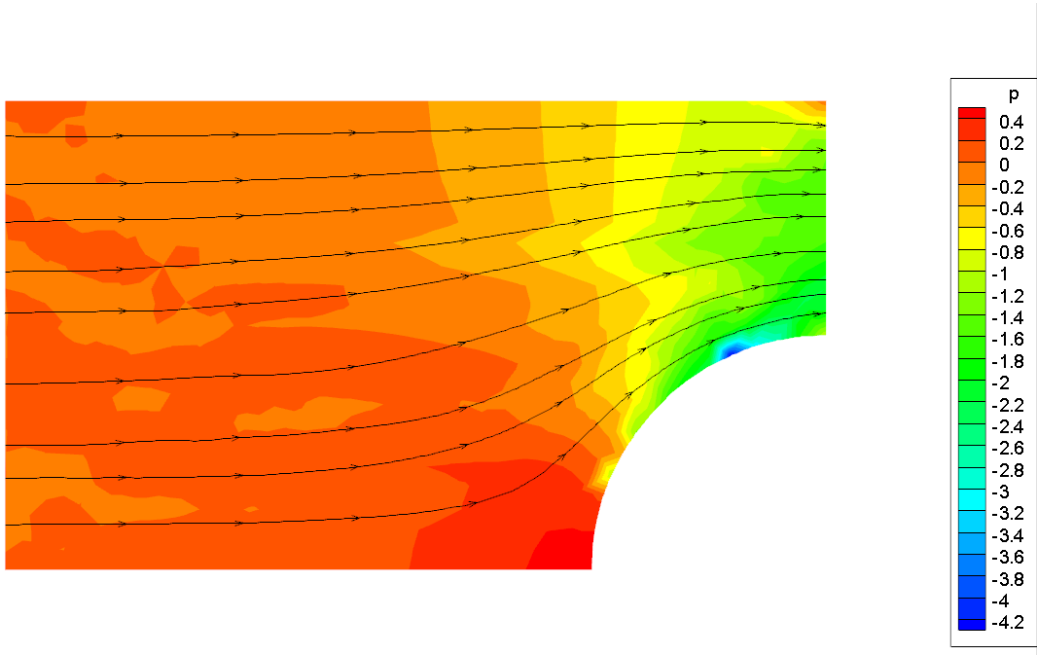

Phi
1.95
1.9
1.85
1.8
1.75
1.7
1.65
1.6
1.55
1.5
1.45
1.4
1.35
1.3
1.25
1.2
1.15
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7


Phi
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

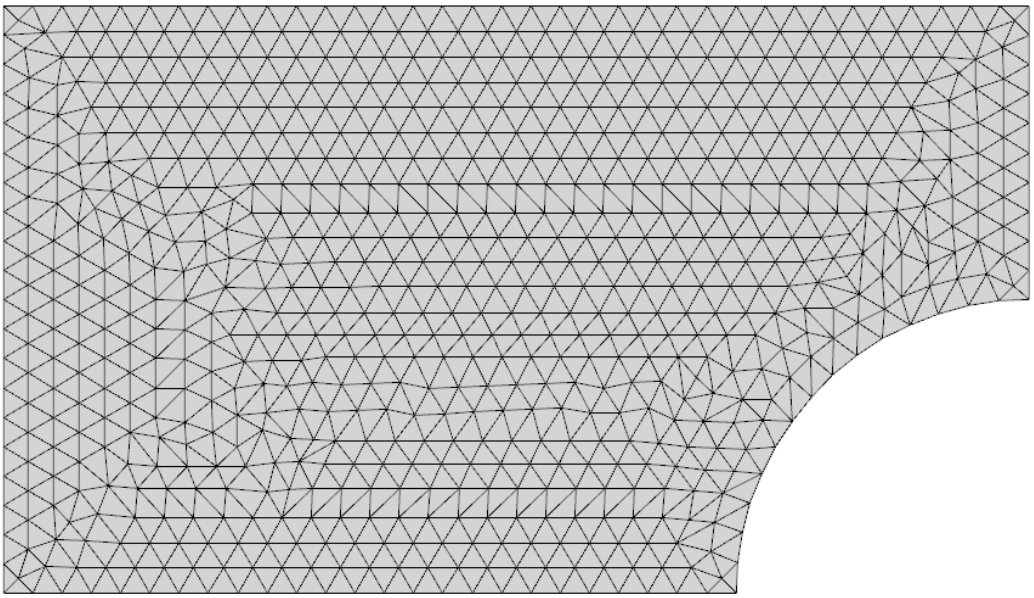
速度：


Velocity Magnitude
2.08312
2
1.76013
1.6
1.39046
1.2
1.10063
0.988617
0.912412
0.869873
0.812454
0.8
0.756349
0.751355
0.71652
0.70034
0.696822
0.694777
0.653457
0.652886
0.632147
0.579127
0.559385
0.485521
0.4
0

压力：


p
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
-2.2
-2.4
-2.6
-2.8
-3
-3.2
-3.4
-3.6
-3.8
-4
-4.2

网格读取：



存在的问题：在速度的计算过程中计算出的速度场不够光滑，因为速度值是在单元中心的而计算后要将其转换到单元顶点上，这个过程实际上是一个插值的过程，因此可以通过考虑更光滑的插值方法或者加密网格使得速度场更光滑。

程序附录

1. 有限元方法计算圆柱绕流问题

Main.f90

```
PROGRAM Main
! This is main program to compute potential flow by using Finite Element
Method
USE READ_GRID                ! Module read grid
USE FIND_BOUND               ! Module find boundary
USE CONJ_GRAD                ! Module Conjugate Gradient Solver for
symmetry positive define system
implicit none
!---------------------------------------------------------------------

! Mesh variables
integer NP,NE                                    ! Number of Nodes and
Elements
real*8, allocatable:: X(:),Y(:)                  ! Position of Nodes
integer, allocatable::   NOD(:,:)                ! Topology of Elements

! Record boundary nodes
integer:: N_ab = 0
integer, allocatable::   ab(:)
integer:: N_bc = 0
integer, allocatable::   bc(:)
integer:: N_de = 0
integer, allocatable::   de(:)
integer:: N_ea = 0
integer, allocatable::   ea(:)


! local index variables
integer i,j,ii
integer status_msg

! Stiffness Matrix and right column vector
real*8, allocatable:: A(:,:), f(:)

real*8 A_e                          ! Area of element
real*8 b1,b2,b3,c1,c2,c3            ! interpolation coefficient
integer i_e1,i_e2,i_e3              ! index of elements' nodes
```

```
! Punishment factor for Dirichelet boundary condition
real*8,parameter:: punishment = 1D8


! Result variables
real*8, allocatable:: phi(:),u(:),p(:)


!---------------------------------------------------------------------

! Message
print*, 'Program Main exit...'
!---------------------------------------------------------------------

! Read Grid data
call ReadGrid(NP,NE,X,Y,NOD)                    ! Call sbroutine to read grid
!---------------------------------------------------------------------

! Find Boundary Nodes
call FindBoundary(NP,X,Y,N_ab,N_bc,N_de,N_ea,ab,bc,de,ea)
!---------------------------------------------------------------------

! Generate Element Stifness Matrix and Assemble Total Stiffness Matrix
Allocate(A(1:NP,1:NP), STAT = status_msg)
print*, 'memory allocate state for A(:,:) is:("0" means success)    ',status_msg

do i = 1,NP
    do j = 1,NP
        A(i,j) = 0.d0              ! initializing stiffness matrix
    end do
end do

do i = 1,NE
    ! load nodes index
    i_e1 = NOD(1,i)
    i_e2 = NOD(2,i)
    i_e3 = NOD(3,i)

    ! computing element's area
    A_e = 5.d-1*( (X(i_e2)-X(i_e1))*(Y(i_e3)-Y(i_e1)) - (Y(i_e2)-
Y(i_e1))*(X(i_e3)-X(i_e1)) )

    ! computing interpolation coefficient b_{i}
```

```
    b1 = 5.d-1 * 1.d0/A_e*( Y(i_e2)-Y(i_e3) )
    b2 = 5.d-1 * 1.d0/A_e*( Y(i_e3)-Y(i_e1) )
    b3 = 5.d-1 * 1.d0/A_e*( Y(i_e1)-Y(i_e2) )

    ! computing interpolation coefficient c_{i}
    c1 = 5.d-1 * 1.d0/A_e*( X(i_e3)-X(i_e2) )
    c2 = 5.d-1 * 1.d0/A_e*( X(i_e1)-X(i_e3) )
    c3 = 5.d-1 * 1.d0/A_e*( X(i_e2)-X(i_e1) )

    ! assemble total stiffness matrix
    A(i_e1,i_e1) = A(i_e1,i_e1) + b1**2 + c1**2
    A(i_e1,i_e2) = A(i_e1,i_e2) + b2*b1 + c2*c1
    A(i_e1,i_e3) = A(i_e1,i_e3) + b3*b1 + c3*c1

    A(i_e2,i_e1) = A(i_e2,i_e1) + b1*b2 + c1*c2
    A(i_e2,i_e2) = A(i_e2,i_e2) + b2**2 + c2**2
    A(i_e2,i_e3) = A(i_e2,i_e3) + b3*b2 + c3*c2

    A(i_e3,i_e1) = A(i_e3,i_e1) + b1*b3 + c1*c3
    A(i_e3,i_e2) = A(i_e3,i_e2) + b2*b3 + c2*c3
    A(i_e3,i_e3) = A(i_e3,i_e3) + b3**2 + c3**2

end do

!---------------------------------------------------------------------

! Adding Dirichelet boundary condition in Total Stiffness Matrix and creat
right column vector
Allocate(f(1:NP), STAT = status_msg)
print*, 'memory allocate state for f(:) is:("0" means success)    ',status_msg

do i = 1,NP            ! Initializing f(:)
      f(i) = 0.d0
end do


do i = 1,N_ab         ! boundary ab
       ii = ab(i)     ! get node's ID in boundary
       A(ii,ii) = A(ii,ii) + punishment
       f(ii) = punishment * 0.d0
end do

do i = 1,N_bc         ! boundary bc
       ii = bc(i)     ! get node's ID in boundary
```

```
        A(ii,ii) = A(ii,ii) + punishment
        f(ii) = punishment * 0.d0
end do

do i = 1,N_de         ! boundary de
        ii = de(i)      ! get node's ID in boundary
        A(ii,ii) = A(ii,ii) + punishment
        f(ii) = punishment * 2.d0
end do

do i = 1,N_ea         ! boundary ea
        ii = ea(i)      ! get node's ID in boundary
        A(ii,ii) = A(ii,ii) + punishment
        f(ii) = punishment * Y(ii)
end do

!---------------------------------------------------------------------

! Solve Total Stiffness Matrix
Allocate(phi(1:NP), STAT = status_msg)
print*, 'memory allocate state for phi(:) is:("0" means success)    ',status_msg
Allocate(u(1:NP), STAT = status_msg)
print*, 'memory allocate state for u(:) is:("0" means success)    ',status_msg
Allocate(p(1:NP), STAT = status_msg)
print*, 'memory allocate state for p(:) is:("0" means success)    ',status_msg


call solve(A,f,phi,NP)
!---------------------------------------------------------------------

! Regroup data


!---------------------------------------------------------------------

! Output Result
call Output(NP,NE,X,Y,Phi,NOD)
!---------------------------------------------------------------------

! Free Memory
DEALLOCATE(X,Y,NOD)
DEALLOCATE(ab,bc,de,ea)
!---------------------------------------------------------------
```

```
! Message
print*, 'Program Main exit...'
print*, ''
!---------------------------------------------------------------------

! Program end
END PROGRAM Main
```

ReadGrid.f90

```
MODULE READ_GRID
! Read Grid data
CONTAINS

SUBROUTINE ReadGrid(NP,NE,X,Y,NOD)
! This Subroutine is to read mesh in Finite Element Method
implicit none
! variables
integer NP,NE
real*8, allocatable,INTENT(INOUT):: X(:),Y(:)
integer, allocatable,INTENT(INOUT)::    NOD(:,:)

! local Variables
integer i
integer status_msg
!-----------------------------------------------------------------------------------------------
---


! Message
print*, 'Enter Subroutine: ReadGrid...'
!-----------------------------------------------------------------------------------------------
---

! Open grid file
open(10,FILE='GRID.DAT')
!-----------------------------------------------------------------------------------------------
---

! read nodes number
read(10,1001) NP,NE
print*, 'NUmber of Points =',NP,', Number of Element = ',NE
!-----------------------------------------------------------------------------------------------
```

```
---

! Alloc memory
ALLOCATE(X(1:NP), STAT = status_msg )
if (status_msg == 0) then
   print*, 'Allocate X(NP) success!'
else
   print*, 'Allocate X(NP) fail!'
end if
ALLOCATE(Y(1:NP), STAT = status_msg)
if (status_msg == 0) then
   print*, 'Allocate Y(NP) success!'
else
   print*, 'Allocate Y(NP) fail!'
end if
ALLOCATE(NOD(1:3,1:NE), STAT = status_msg)
if (status_msg == 0) then
   print*, 'Allocate NOD(1:3,NE) success!'
else
   print*, 'Allocate NOD(1:3,NE) fail!'
end if
!-----------------------------------------------------------------------------------------------
---

! Read nodes' position
do i=1,NP
     read(10,1002) X(i),Y(i)
end do

!print*, X,Y
!-----------------------------------------------------------------------------------------------
---

! Read elements' topology
do i=1,NE
   read(10,1003) NOD(1,i),NOD(2,i),NOD(3,i)
end do
!-----------------------------------------------------------------------------------------------
---

close(10)
1001 format(1X,2I10)
1002 format(1X,E13.6,2X,E13.6)
1003 format(1X,3I10)
```

```
!-------------------------------------------------------------------------------------------------
---


! Message
print*, 'Subroutine: ReadGrid exit...'
print*, ''
!-------------------------------------------------------------------------------------------------
---


! Program end
END SUBROUTINE ReadGrid
END MODULE READ_GRID
```

FindBoundary.f90

```
MODULE FIND_BOUND
! Find Mesh Boundarys
CONTAINS

SUBROUTINE FindBoundary(NP,X,Y,N_ab,N_bc,N_de,N_ea,ab,bc,de,ea)
! This subroutine is to find mesh boundary
implicit none

! Mesh Information
integer NP
real*8 X(1:NP),Y(1:NP)

! Record boundary nodes
integer N_ab
integer, allocatable::    ab(:)
integer N_bc
integer, allocatable::    bc(:)
integer N_de
integer, allocatable::    de(:)
integer N_ea
integer, allocatable::    ea(:)

! Local variables
integer:: i=0, i_ab=1, i_bc=1, i_de=1, i_ea=1
integer status_msg
!-------------------------------------------------------------------------------------------------
```

```
---


! Message
print*, 'Enter Subroutine FindBoundary...'
!-------------------------------------------------------------------------------------------------
---

! Find Boundary Nodes
do i = 1,NP
   ! ab
   if( ABS(Y(i) - 0.D0) < 1D-10 ) then
     N_ab = N_ab + 1
   end if

   ! bc
   if( ABS(X(i)**2 + Y(i)**2 - 1) < 1D-4 .and. ABS(Y(i) - 0.D0) > 1D-10) then
     N_bc = N_bc + 1
   end if

   ! de
   if( ABS(Y(i) - 2.D0) < 1D-10 ) then
     N_de = N_de + 1
   end if

   ! ea
   if( ABS(X(i) + 3.5D0) < 1D-10 .and. ABS(Y(i) - 0.D0) > 1D-10 .and. ABS(Y(i)
- 2.D0) > 1D-10 ) then
     N_ea = N_ea + 1
   end if

end do

print*, 'N_ab=', N_ab
print*, 'N_bc=', N_bc
print*, 'N_de=', N_de
print*, 'N_ea=', N_ea


ALLOCATE(ab(1:N_ab), STAT = status_msg )
print*, 'memory allocate state for ab is:("0" means success)    ',status_msg
ALLOCATE(bc(1:N_bc), STAT = status_msg )
print*, 'memory allocate state for bc is:("0" means success)    ',status_msg
ALLOCATE(de(1:N_de), STAT = status_msg )
```

```
print*, 'memory allocate state for de is:("0" means success)    ',status_msg
ALLOCATE(ea(1:N_ea), STAT = status_msg )
print*, 'memory allocate state for ea is:("0" means success)    ',status_msg

do i = 1,NP
   ! ab
   if( ABS(Y(i) - 0.D0) < 1D-10 ) then
     ab(i_ab) = i
     i_ab = i_ab + 1
   end if

   ! bc
   if( ABS(X(i)**2 + Y(i)**2 - 1) < 1D-4 .and. ABS(Y(i) - 0.D0) > 1D-10) then
     bc(i_bc) = i
     i_bc = i_bc + 1
   end if

   ! de
   if( ABS(Y(i) - 2.D0) < 1D-10 ) then
     de(i_de) = i
     i_de = i_de + 1
   end if

   ! ea
   if( ABS(X(i) + 3.5D0) < 1D-10 .and. ABS(Y(i) - 0.D0) > 1D-10 .and. ABS(Y(i)
- 2.D0) > 1D-10 ) then
      ea(i_ea) = i
      i_ea = i_ea + 1
   end if

end do
!-----------------------------------------------------------------------------------------------
--------------------

! Message
print*, 'Subroutine FindBoundary exit...'
print*, ''
!-----------------------------------------------------------------------------------------------
--------------------

! Program end
END SUBROUTINE FindBoundary
END MODULE FIND_BOUND
```

ConjugateGradientSolver.f90

```
module CONJ_GRAD
!---------------------------------------module coment
!   Version        :   V1.0
!   Coded by       :   syz
!   Date           :   2010-4-5
!-------------------------------------------------
!   Description :  共轭梯度法
!
!-------------------------------------------------
!   Parameters   :
!         1.      IMAX--最大允许迭代次数
!         2.      tol--误差容限
!
!   Contains      :
!         1.      solve  迭代法方法函数
!         2.
!         3.      dr(r,N)  计算向量长度平方函数
!         4.       Ar(A,r,N)   计算矩阵乘以向量函数，返回向量
!         5.      rAr(A,r,N)    计算（Ar,r）函数
!-------------------------------------------------
!   Post Script :
!         1.      里面的三个函数，可以简化程序，同时可以用在其他地方
!         2.
!-------------------------------------------------

implicit real*8(a-z)
integer::IMAX = 5000
real*8::tol = 1d-16


contains

subroutine solve(A,b,x,N)
!--------------------------------subroutine   comment
!   Version     :   V1.0
!   Coded by    :   syz
!   Date        :
!-------------------------------------------------
!   Purpose     :   共轭梯度法
!                   用于计算方程  AX=b
!-------------------------------------------------
```

```
!   Input    parameters    :
!           1.   A,b  意义即    AX=b
!           2.   x0迭代初值
!           3.   N  方程的维数
!   Output parameters    :
!           1. x  方程的解
!           2.
!   Common parameters    :
!
!-------------------------------------------------

implicit real*8(a-z)
integer::N
integer::i,j,k

real*8::A(N,N),b(N),x(N),x0(N)
real*8::r0(N),r1(N),p0(N),p1(N)

real*8::x1(N),x2(N)

! Message
print*, 'Enter Subroutine solve...'
!-----------------------------------------------------------------------------------------------------
---

! Open grid file
open(102,FILE='Solver.dat')
!-----------------------------------------------------------------------------------------------------
---
!写入标题
   write(102,501)
   501 format(1x,//,18x,'共轭梯度法中间结果',//)
!-----------------------------------------------------------------------------------------------------
---

! Initializing value for iterative
do i = 1,N
     x0(i) = 0.d0
end do

!-----------------------------------------------------------------------------------------------------
----

x1=x0
```

```
r0=b-Ar(A,x1,N)

p0=r0

do k=1,IMAX


    tmp1=dr(r0,N)
    tmp2=rAr(A,p0,N)

    afa=tmp1/tmp2

    x2=x1+afa*p0




    !如果r0接近于0，则停止迭代
    !该部分算法在《数值分析原理》（李庆扬、关治、
    !白峰彬编）中叙述较为详细
    dr_s=dsqrt(dr(r0,N))

    if (dr_s<tol) exit


    r1=r0-afa*Ar(A,p0,N)

    tmp3=dr(r1,N)

    beta=tmp3/tmp1

    p1=r1+beta*p0

    ! Residual in interative
    write(102,502) k, tmp3
    502 format(1x,'k = ',1I8,',',5x,'residual = ',ES15.8)

    !全部更新
    r0=r1
    p0=p1
    x1=x2
```

```
end do

      x=x2

! Message
print*, 'Subroutine solve exit...'
print*, ''
!-------------------------------------------------------------------------------------------------
---

! Subroutine end
end subroutine solve




function dr(r,N)
!--------------------------------subroutine  comment
!  Version    :  V1.0
!  Coded by   :  syz
!  Date       :
!-----------------------------------------------------
!  Purpose    :  计算向量长度平方  (r,r)
!
!-----------------------------------------------------
!  Input  parameters  :
!          1.      r向量
!          2.      N维数
!  Output parameters  :
!          1.      dr  长度平方
!          2.
!  Common parameters  :
!
!-----------------------------------------------------

implicit real*8(a-z)
integer::N,i
real*8::r(N),dr

s=0
do i=1,N
  s=s+r(i)**2
end do
dr=s
```

```
end function dr

function Ar(A,r,N)
!--------------------------------subroutine    comment
!   Version     :   V1.0
!   Coded by    :   syz
!   Date        :
!-----------------------------------------------------
!   Purpose     :   !计算   A*r,返回  N维向量
!
!-----------------------------------------------------
!   Input   parameters   :
!           1.      r向量
!           2.      N维数
!           3.      A矩阵
!   Output parameters   :
!           1.      Ar返回向量
!           2.
!   Common parameters   :
!
!-----------------------------------------------------

implicit real*8(a-z)
integer::i,N
real*8::A(N,N),r(N),temp(N),Ar(N)

temp=0

do i=1,N
   do j=1,n
      temp(i)=temp(i)+A(i,j)*r(j)
   end do
end do
Ar=temp
end function ar

function v1v2(v1,v2,N)
!--------------------------------subroutine    comment
!   Version     :   V1.0
!   Coded by    :   syz
!   Date        :   2010-7-29
!-----------------------------------------------------
!   Purpose       :  向量点乘     v1v2=v1(1)*v2(1)+v1(2)*v(2)+...
!
```

```
!   Post Script :
!           1.
!           2.
!           3.
!
!-------------------------------------------------
!   Input    parameters    :
!           1.      v1,v2  向量
!           2.      N  向量维数
!   Output parameters    :
!           1.      v1,v2向量点乘值
!           2.
!   Common parameters    :
!           1.
!           2.
!-------------------------------------------------

implicit real*8(a-z)
integer::n

real*8::v1(n),v2(n)

integer::i

v1v2=0
do i=1,n

 v1v2=v1v2+v1(i)*v2(i)

end do
end function




function rAr(A,r,N)
!-------------------------------subroutine    comment
!   Version     :    V1.0
!   Coded by    :    syz
!   Date        :
!-------------------------------------------------
!   Purpose     :    !计算（Ar,r）,返回标量
!
!-------------------------------------------------
```

```
!   Input    parameters    :
!           1.          r向量
!           2.          N维数
!           3.          A矩阵
!   Output parameters    :
!           1.          Ar返回标量
!           2.
!   Common parameters    :
!-------------------------------------------------

implicit real*8(a-z)
integer::i,N
real*8::A(N,N),r(N),temp(N)

temp=Ar(A,r,N)
rAr=v1v2(r,temp,N)
end function rAr


end module CONJ_GRAD
```

Output.f90

```
SUBROUTINE Output(NP,NE,X,Y,Phi,NOD)
! This program is to output data
implicit none

integer NP,NE
real*8    X(1:NP),Y(1:NP),Phi(1:NP)
integer NOD(1:3,1:NE)

! local variables
integer i
!-------------------------------------------------------------------------------------------------
---

!Message
print*, 'Enter Subroutine Output...'
!-------------------------------------------------------------------------------------------------
---

!Output data in tecplot format
```

```
open(10,file='Result.dat')

Write(10,1001) 'TITLE = "FEM:Potential Flow"'
Write(10,1002) 'VARIABLES = "X", "Y", "Phi"'
Write(10,1003) 'ZONE T = "TRIANGLES", NODES =',NP,', ELEMENTS
=',NE,', DATAPACKING =POINT, ZONETYPE = FETRIANGLE'
!-------------------------------------------------------------------------------------------------
---

! Write nodes' position
do i=1,NP
     write(10,1004) X(i),Y(i),Phi(i)
end do

write(10,*) ''
!-------------------------------------------------------------------------------------------------
---

! Write elements' topology
do i=1,NE
   write(10,1005) NOD(1,i),NOD(2,i),NOD(3,i)
end do
!-------------------------------------------------------------------------------------------------
---

1001 format(1x,A)
1002 format(1x,A)
1003 format(1x,A,1I6,A,1I6,A)
1004 format(1X,3ES15.6)
1005 format(1X,3I10)

close(10)
!-------------------------------------------------------------------------------------------------
---

! Message
print*, 'Subroutine Output exit...'
print*, ''
!-------------------------------------------------------------------------------------------------
---

! Subroutine end
END SUBROUTINE OutPut
```