```
! 平面结构几何构造分析
! 判断平面杆系结构为
! 1: 静定结构,无多余约束
! 2: 超静定结构
! 3: 瞬变体系
! 4: 部分瞬变，部分常变体系
! 5: 常变体系
! 清华大学土木工程系结62
! 陆新征，学号：960133
! Last change:  123  18 Jan 98    2:08 am

!***************
 module NumKind
!***************
   implicit none
   integer (kind(1)),parameter :: ikind=kind(1)
   integer (kind(1)),parameter :: rkind=kind(0.D0)
   real (rkind),     parameter :: Zero=0.D0,One=1.D0,Two=2.D0,Three=3.D0, &
   &    Four=4.D0,Five=5.D0,Six=6.D0,Seven=7.D0,Eight=8.D0,Nine=9.D0, &
   &    Ten=10.D0,PI=3.141592653589793d0
end module NumKind

!**************
 module TypeDef
!**************
   use NumKind
   implicit none
   integer (ikind),parameter :: NDOF=3,NNode=2

   type :: typ_Joint
      real (rkind)      :: X,Y
      integer (ikind)   :: GDOF(NDOF)
   end type typ_Joint

   type :: typ_Element
      integer (ikind)   :: JointNo(NNode)
      real (rkind)      :: Length,EI,EA
      integer(ikind)    :: GlbDOF(NDOF*NNode)
   end type typ_Element

   type :: typ_Kcol
         real(rkind),pointer   :: row(:)
   end type typ_Kcol
   contains

   !==================================
    subroutine SetElemProp (Elem, Joint)
   !==================================
      type (typ_Element),intent(in out) :: Elem(:)
      type (typ_Joint),intent(in) :: Joint(:)
      integer(ikind) :: NElem,n1,n2,i
      real(rkind) :: x1,y1,x2,y2

      NElem=size(Elem,dim=1)
      do i=1,NElem
```

```
         n1=Elem(i)%JointNo(1)
         n2=Elem(i)%JointNo(2)
         x1=Joint(n1)%X
         y1=Joint(n1)%Y
         x2=Joint(n2)%X
         y2=Joint(n2)%Y
         Elem(i)%Length=sqrt((x2-x1)**2+(y2-y1)**2)
         Elem(i)%GlbDoF(1:3)=Joint(n1)%GDoF
         Elem(i)%GlbDoF(4:6)=Joint(n2)%GDoF
       end do

       return

   end subroutine SetElemProp

end module TypeDef

!***************
module stability
!***************
   use TypeDef
   implicit none
   contains

    !==================================
    subroutine ElemG(EG, Elem,ie,Joint)
    !==================================
     real (rkind),intent (in out) :: EG(:,:)
     type (typ_Element),intent (in) :: Elem(:)
     type (typ_Joint),intent (in) :: Joint(:)
     integer (ikind),intent (in) :: ie
     real (rkind) :: x1,y1,x2,y2
     integer (ikind) :: n1,n2

     n1=Elem(ie)%JointNo(1)
     n2=Elem(ie)%JointNo(2)
     x1=Joint(n1)%X
     y1=Joint(n1)%Y
     x2=Joint(n2)%X
     y2=Joint(n2)%Y

     EG(1,:)=(/one,zero,y1-y2,-one,zero,zero/)
     EG(2,:)=(/zero,one,zero,zero,-one,x2-x1/)
     EG(3,:)=(/zero,zero,one,zero,zero,-one/)

     return
   end subroutine ElemG

  !========================
   subroutine GetRank(r,js, GG)
  !========================
   integer (ikind),intent (in out)        :: r
   real (rkind),intent (in out)           :: GG(:,:)
   integer (ikind)                        :: m,n,nn,i,j,l,is
   real (rkind)                           :: d,T
```

```
integer (ikind),intent (in out)        :: js(:)

m=size (GG,dim=1)
n=size (GG,dim=2)
nn=min (m,n)

r=0
do l=1,nn
   T=Zero
   do i=l,m
      do j=l,n
         if (abs(GG(i,j))>T) then
            T=abs(GG(i,j))
            is=i
            js(l)=j
         end if
      end do
   end do
   if (T<abs(1e-9)) then
      return
   end if

   r=r+1
   if (is.ne.l) then
      do j=l,n
         d=GG(l,j)
         GG(l,j)=GG(is,j)
         GG(is,j)=d
      end do
   end if
   if (js(l).ne.l) then
      do i=l,m
         d=GG(i,js(l))
         GG(i,js(l))=GG(i,l)
         GG(i,l)=d
      end do
   end if
   do i=l+1,m
      d=GG(i,l)/GG(l,l)
      do j=l,n
         GG(i,j)=GG(i,j)-d*GG(l,j)
      end do
   end do
end do
return
end subroutine GetRank

!=========================================
 function SolveInstant(r,GG,Elem,Joint,js) result (e)
!=========================================
    integer (ikind),intent (in out) :: r
    real (rkind),intent (in) :: GG(:,:)
    type (typ_Element),intent (in out) :: Elem(:)
    type (typ_Joint),intent (in out) :: Joint(:)
    real (rkind),allocatable :: G(:,:),GDisp(:)
```

```
real (rkind),allocatable :: d(:),c(:),K(:,:),A(:,:)
integer (ikind) :: m,n,i,j,NElem,n1,n2,rr,nn
integer (ikind) :: e
real (rkind) :: EDisp(6),scalar,h
integer (ikind),intent (in) :: js(:)
integer (ikind),allocatable :: is(:)

m=size (GG,dim=1)
n=size (GG,dim=2)
nn=min (m,n)
NElem=size (Elem,dim=1)

allocate (G(m,n))
allocate (A(r,n-r))
allocate (GDisp(n))
allocate (d(r))
allocate (c(r))
allocate (K(r,r))
allocate (is(nn))

K(:,:)=GG(1:r,1:r)
do i=1,r
   do j=1,n-r
      A(i,j)=GG(i,r+j)
   end do
end do

GDisp=Zero
e=0
scalar=minval (Elem(:)%Length)*10.D-5
do i=r+1,n
   GDisp(i)=one
   c=matmul(A,GDisp(r+1:n))
   c=-c
   call Gauss(d,K,c)
   GDisp(1:r)=d
   do j=nn-1,1,-1
         if (js(j).ne.j) then
            h=GDisp(j)
            GDisp(j)=GDisp(js(j))
            GDisp(js(j))=h
         end if
   end do
   do j=1,NElem
      call GetEDisp(EDisp,Elem,j,GDisp)
      n1=Elem(j)%JointNo(1)
      n2=Elem(j)%JointNo(2)
      Joint(n1)%x=Joint(n1)%x+scalar*EDisp(1)
      Joint(n1)%y=Joint(n1)%y+scalar*EDisp(2)
      Joint(n2)%x=Joint(n2)%x+scalar*EDisp(4)
      Joint(n2)%y=Joint(n2)%y+scalar*EDisp(5)
   end do
   call GstifMat(G,Elem,Joint)
   call GetRank(rr,is,G)
   if (n-rr>0) then
```

```
          e=e+1
        end if
     end do

     return
  end function SolveInstant

  !=======================================
   subroutine GetEDisp(EDisp, Elem,ie,GDisp)
  !=======================================
      real (rkind),intent (in out) :: EDisp(:)
      type (typ_Element),intent (in) :: Elem(:)
      integer (ikind),intent (in) :: ie
      real (rkind),intent (in) :: GDisp(:)

      where (Elem(ie)%GlbDOF>0)
         EDisp(:)=GDisp(Elem(ie)%GlbDOF(:))
      elsewhere
         EDisp(:)=Zero
      end where
      return

   end subroutine GetEDisp

    !=============================
     subroutine Gauss (d, A,b)
    !=============================
        real (rkind),intent (in out) :: d(:)
        real (rkind),intent (in) :: B(:),A(:,:)
        real (rkind) :: s
        integer (ikind) :: i,j,n

        n=size (A,dim=1)
        s=Zero
        d(n)=B(n)/A(n,n)
        do i=n-1,1,-1
           do j=i+1,n
              s=s+A(i,j)*d(j)
           end do
           d(i)=B(i)-s
        end do

        return
    end subroutine Gauss

 !================================
  subroutine GstifMat(G, Elem,Joint)
 !================================
  real (rkind),intent (in out)  :: G(:,:)
  type (typ_Element),intent (in)  :: Elem(:)
  type (typ_Joint),intent (in)  :: Joint(:)
  integer (ikind) :: Nelem,i,ie,Row1
  integer (ikind) :: ElocVec(6)
  real (rkind) :: EG(3,6)
```

```
    NElem=size (Elem,dim =1)
    G=Zero

    do ie=1,Nelem
       call ElemG(EG,Elem,ie,joint)
       ElocVec=Elem(ie)%GlbDOF
       do i=1,3
          Row1=(ie-1)*3+i
          where (ElocVec>0)
             G(Row1,ElocVec)=EG(i,:)
          end where
       end do
    end do
    return
 end subroutine GstifMat


end module stability

!==============
 program stab
!==============
    use stability
    implicit none
    integer (ikind) :: NElem,NJoint,NGlbDOF,NJLoad,NEload
    real (rkind),allocatable :: G(:,:)
    type(typ_Element),allocatable  :: Elem(:)
    type(typ_Joint),allocatable :: Joint(:)
    integer(ikind) :: Over,Typ

    call input_data()
    call SetElemProp(Elem,Joint)
    call Judge()
    call output_data()

    stop
    contains

    !-----------------------
     subroutine input_data()
    !-----------------------
        integer (ikind)            :: i,ie
        character (len=20)   :: inputfile
        read (*,*) inputfile

        open(5,file=inputfile,status='OLD',position='REWIND')
        read(5,*) NElem,NJoint,NGlbDOF,NJLoad,NELoad

        allocate(Elem(NELem))
        allocate(Joint(NJoint))
        allocate(G(3*NElem,NGlbDOF))
        write (*,*) NElem,NJoint,NGlbDOF,NJLoad,NELoad
        do i=1,NJoint
           read (5,*) Joint(i)
           !write (*,*) Joint(i)
```

```
    end do
    !read (*,*)
    do ie=1,NElem
       write (*,*) ie
       read (5,*) Elem(ie)%JointNo(1),Elem(ie)%JointNo(2),ELem(ie)%EA,Elem(ie)%EI
       write (*,*) Elem(ie)
    end do
    Over=0
    return

end subroutine input_data

subroutine Judge()
    integer (ikind)               :: M,N,nn
    integer(ikind)                :: r,e
    integer (ikind),allocatable   :: js(:)


    call GstifMat(G,Elem,Joint)
    M=3*NElem
    N=NGlbDOF
    nn=min (M,N)
    allocate (js(nn))
    call GetRank(r,js,G)
    if (r==N) then
       if (M==N) then
          Typ=1
       else
          if (M>N) then
              Typ=2
              Over=M-r
          end if
       end if
    end if

    if (M<N) then
       Typ=-2
       if (r<M) then
          Over=M-r
       end if
    end if

    if (M==N.and.r<M) then
       Over=M-r
       e=SolveInstant(r,G,Elem,Joint,js)
       if (e>0) then
          if (e<n-r) then
             Typ=-1
          else
             Typ=-2
          end if
       else
          Typ=0
       end if
    end if
```

```
        if (M>N.and.r<N) then
           Over=M-r
           e=SolveInstant(r,G,Elem,Joint,js)
           if (e>0) then
              if (e<n-r) then
                 Typ=-1
              else
                 Typ=-2
              end if
           else
              Typ=0
           end if
        end if

        return

     end subroutine Judge

    !------------------------
     subroutine output_data()
    !------------------------
         open (55,file='output.ipt')
         if(Over>0) then
               write (55,*) '多余约束数=',Over
         end if
         select case(Typ)
              case(1)
                  write (55,*) '静定结构,无多余约束'
              case(2)
                  write (55,*) '超静定结构'
              case(0)
                  write (55,*) '瞬变体系'
              case(-1)
                  write (55,*) '部分瞬变，部分常变体系'
              case(-2)
                  write (55,*) '常变体系'
        end select
         return

     end subroutine output_data

end program stab
```