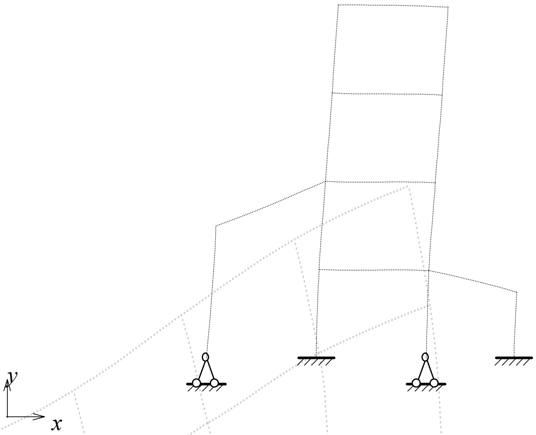
程序运行结果

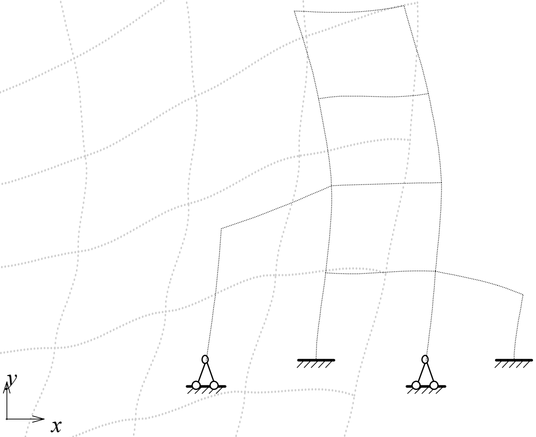
第一阶频率 128.620208740234

第一阶振型



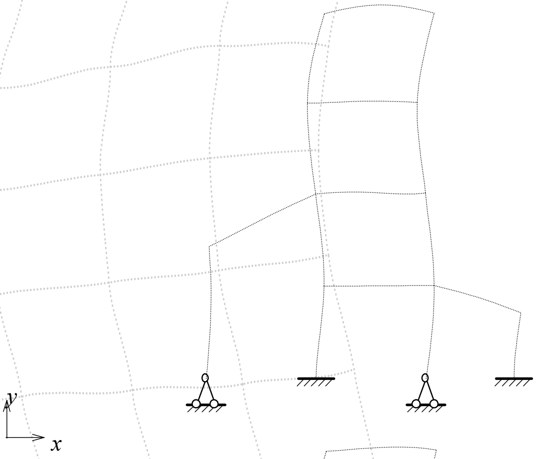
第二阶频率 358.675633493252

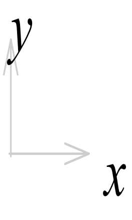
第二阶振型



第三阶频率 686.476133948344

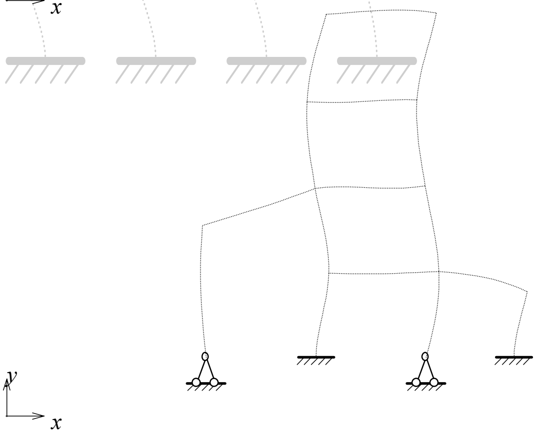
第三阶振型





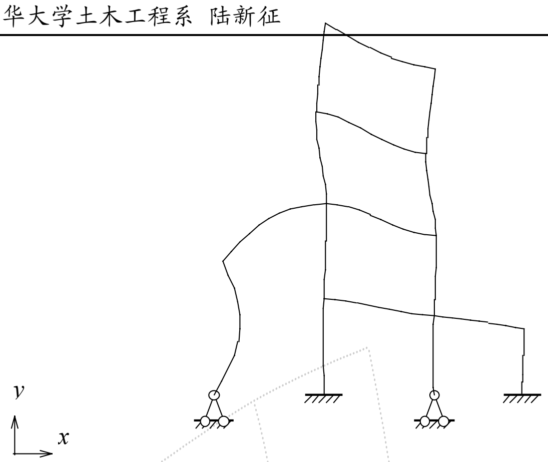
第四阶频率 738.422734953542

第四阶振型



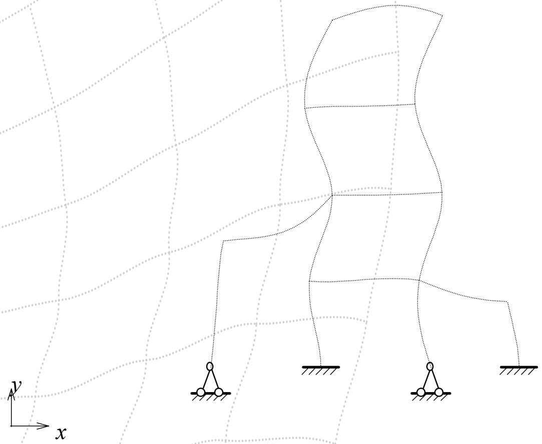
第五阶频率 928.292565142280

第五阶振型



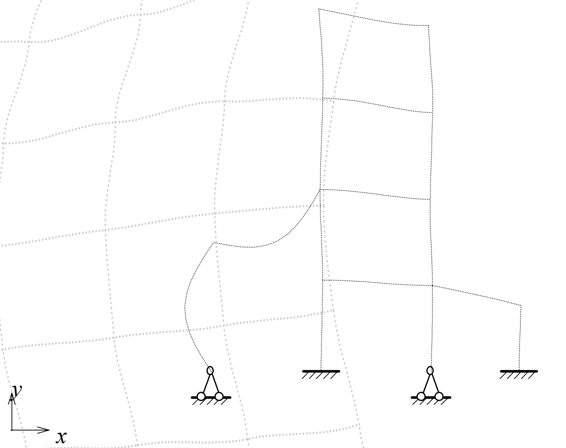
第六阶频率 1037.97205413169

第六阶振型



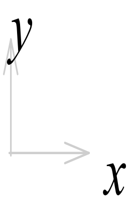
第七阶频率 1101.92035683191

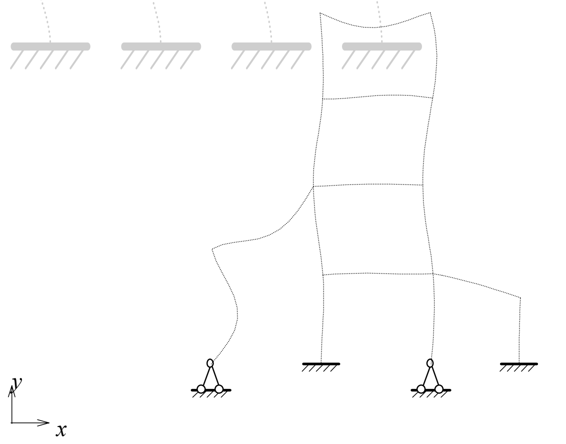
第七阶振型



第八阶频率 1486.01869410407

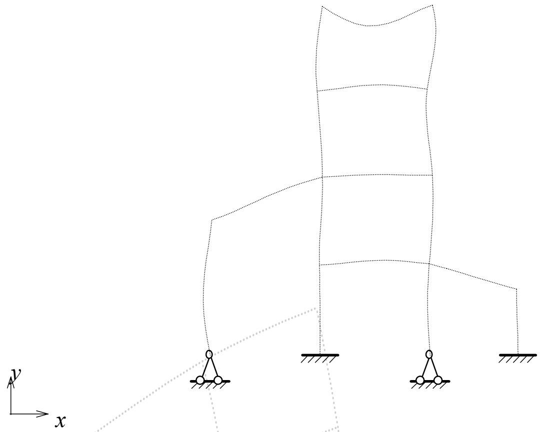
第八阶振型





第九阶频率 1547.47659615027

第九阶振型

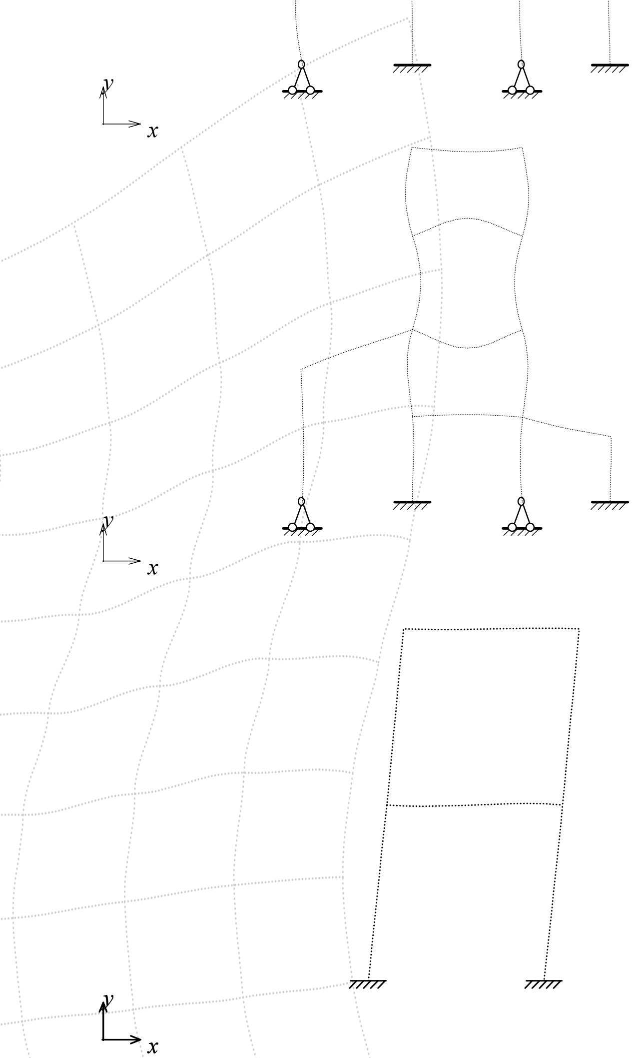


第十阶频率 1811.91903840500

第十阶振型

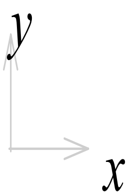
第一阶频率 0.239743232727051

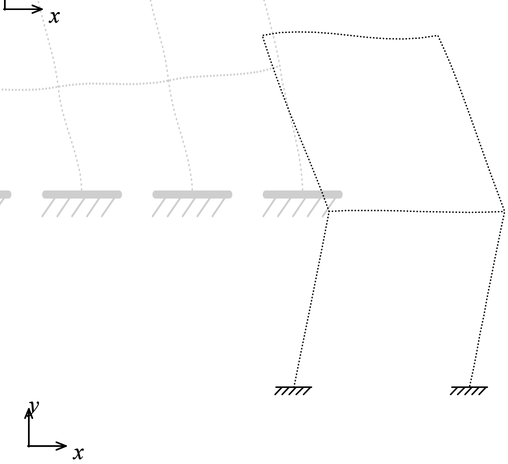
第一阶振型



第二阶频率 0.786964770464693

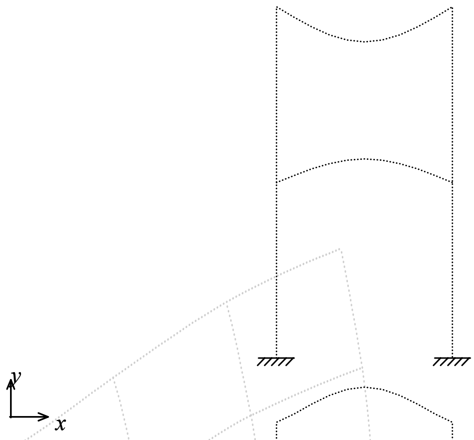
第二阶振型





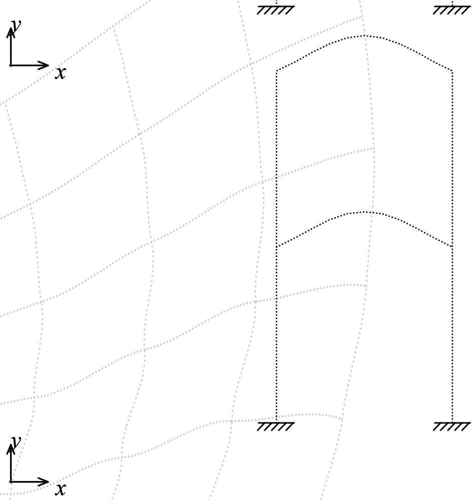
第三阶频率 1.73036664791953

第三阶振型



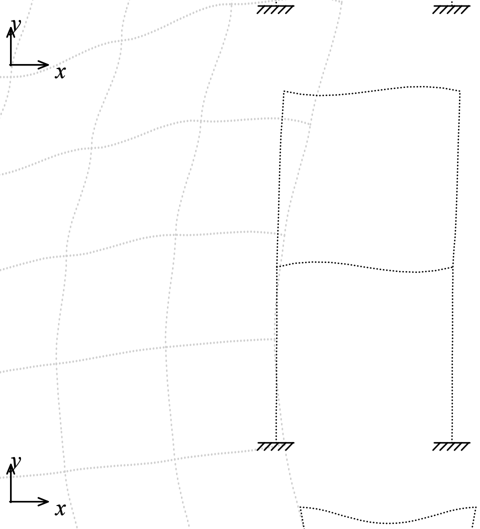
第四阶频率 2.46691309583156

第四阶振型



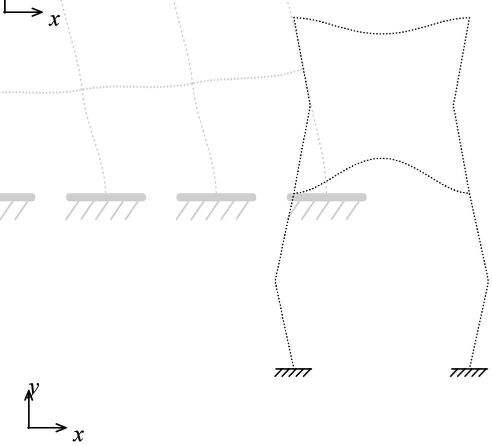
第五阶频率 2.70770212310535

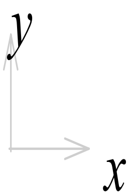
第五阶振型



第六阶频率 3.57983869805967

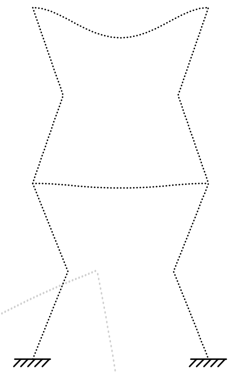
第六阶振型



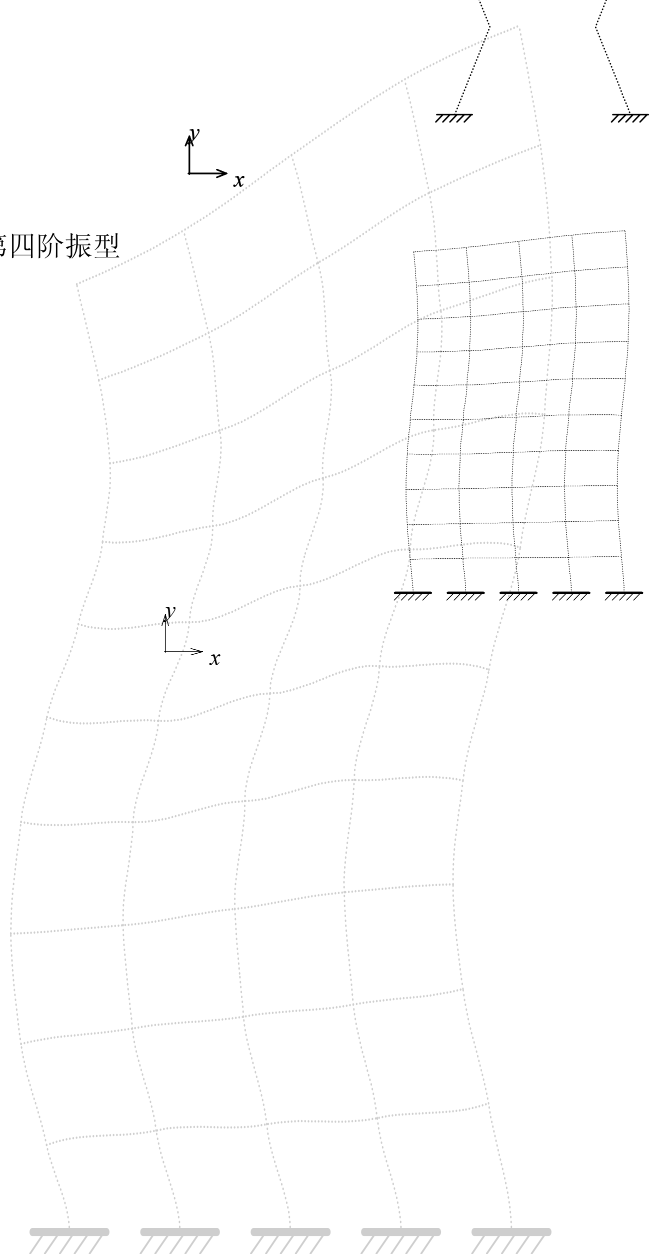


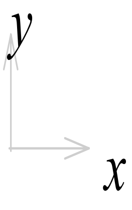
第七阶频率 3.57983869805967

第七阶振型



自选题目：十层框架第四阶振型





程序说明：

1：编程语言为 Fortran 90

2：程序可以计算各种结构的任意阶频率和振型（等截面杆件）

3：采用与结构力学求解器教学版前后处理功能，接口运行良好

4：具有处理重频，添加内部结点的功能

5：频率，振型均由误差限控制，程序运行效率高，结果稳定

程序清单

```
!       Last change:    123    20 Dec 99       0:18 am
!***************
            module NumKind
!***************
    ! This module defines the kind of integer and real numbers.
    ! Every module, subroutine or func must use this module.
    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=3.141592653589793238462643D0
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)              :: EI,EA,Length,CosA,SinA
            integer(ikind)            :: GlbDOF(NDOF*NNode)
               real (rkind)              ::EK(NDOF*NNode,NDOF*NNode)
               real (rkind)              ::ET(NDOF*NNode,NDOF*NNode)
               real (rkind)              ::m
    end type typ_Element

    type :: typ_Kcol
```

```fortran
                    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) :: i,N
            real(rkind)    ::    x1,x2,y1,y2
            real(rkind)    ::    T(NDOF,NDOF)
            N=size(Elem,dim=1)
            !write(*,*) N
            do i=1,N
                x1=Joint(Elem(i)%JointNo(1))%X
                y1=Joint(Elem(i)%JointNo(1))%Y
                x2=Joint(Elem(i)%JointNo(2))%X
                y2=Joint(Elem(i)%JointNo(2))%Y
                Elem(i)%Length=sqrt((x2-x1)**2+(y2-y1)**2)
                Elem(i)%CosA=(x2-x1)/Elem(i)%Length
                Elem(i)%SinA=(y2-y1)/Elem(i)%Length
                Elem(i)%GlbDOF(1:3)=Joint(Elem(i)%JointNo(1))%GDOF
                Elem(i)%GlbDOF(4:6)=Joint(Elem(i)%JointNo(2))%GDOF
                Elem(i)%ET=zero
                T=zero
                T(1,1)=Elem(i)%CosA
                T(1,2)=Elem(i)%SinA
                T(2,1)=-Elem(i)%SinA
                T(2,2)=Elem(i)%CosA
                T(3,3)=one
                Elem(i)%ET(1:3,1:3)=T
                Elem(i)%ET(4:6,4:6)=T
             end do
        return
            !...
    end subroutine SetElemProp
end module TypeDef

module Solve
        use TypeDef

        contains
```

```
        subroutine FindFix(FixNum,freq1,freq2,Elem)  ！判断是否有固端频率
                integer(ikind),intent(in out) :: FixNum(:)
                 real(rkind) ,intent (in out) ::freq1,freq2
                 type(typ_Element),intent (in out) :: Elem(:)
                 integer(ikind)::    NElem,i,Ja,Jb1,Jb2,j
                 real(rkind)    :: mu1,lambda,sg,mu2
                NElem=size(Elem,dim=1)
                 FixNum=0
                 do i=1,NElem
                        mu1=freq1*Elem(i)%Length*sqrt(Elem(i)%m/Elem(i)%EA)
                        mu2=freq2*Elem(i)%Length*sqrt(Elem(i)%m/Elem(i)%EA)
                        Ja=int(mu2/PI)-int(mu1/PI)
                        if (Ja.ne.0) then
                                FixNum(i)=1
                        end if

lambda=Elem(i)%Length*((freq1**2*Elem(i)%m/Elem(i)%EI)**0.25D0)
                        sg=sign(one,one-cosh(lambda)*cos(lambda))
                        j=int(lambda/PI)
                        Jb1=j-(1-(-1)**j*sg)/2

lambda=Elem(i)%Length*((freq2**2*Elem(i)%m/Elem(i)%EI)**0.25D0)
                        sg=sign(one,one-cosh(lambda)*cos(lambda))
                        j=int(lambda/PI)
                        Jb2=j-(1-(-1)**j*sg)/2
                        if(Jb2.ne.Jb1) then
                               FixNum(i)=2
                        end if
                end do
                return
        end subroutine FindFix

        subroutine Calculate_J0(J02,freq2,Elem)  ！计算低于 freq2 的单元固端频率数
                integer(ikind) ,intent(out):: J02
                real(rkind) ,intent(in)::freq2
                type (typ_Element),intent(in)    :: Elem(:)
                integer(ikind)::    NElem,i,Ja,Jb,j
                real(rkind)    :: mu,lambda,sg
                NElem = size(Elem,dim=1)
                J02=0
                do i=1,NElem
                        mu=freq2*Elem(i)%Length*sqrt(Elem(i)%m/Elem(i)%EA)
                        Ja=int(mu/PI)
```

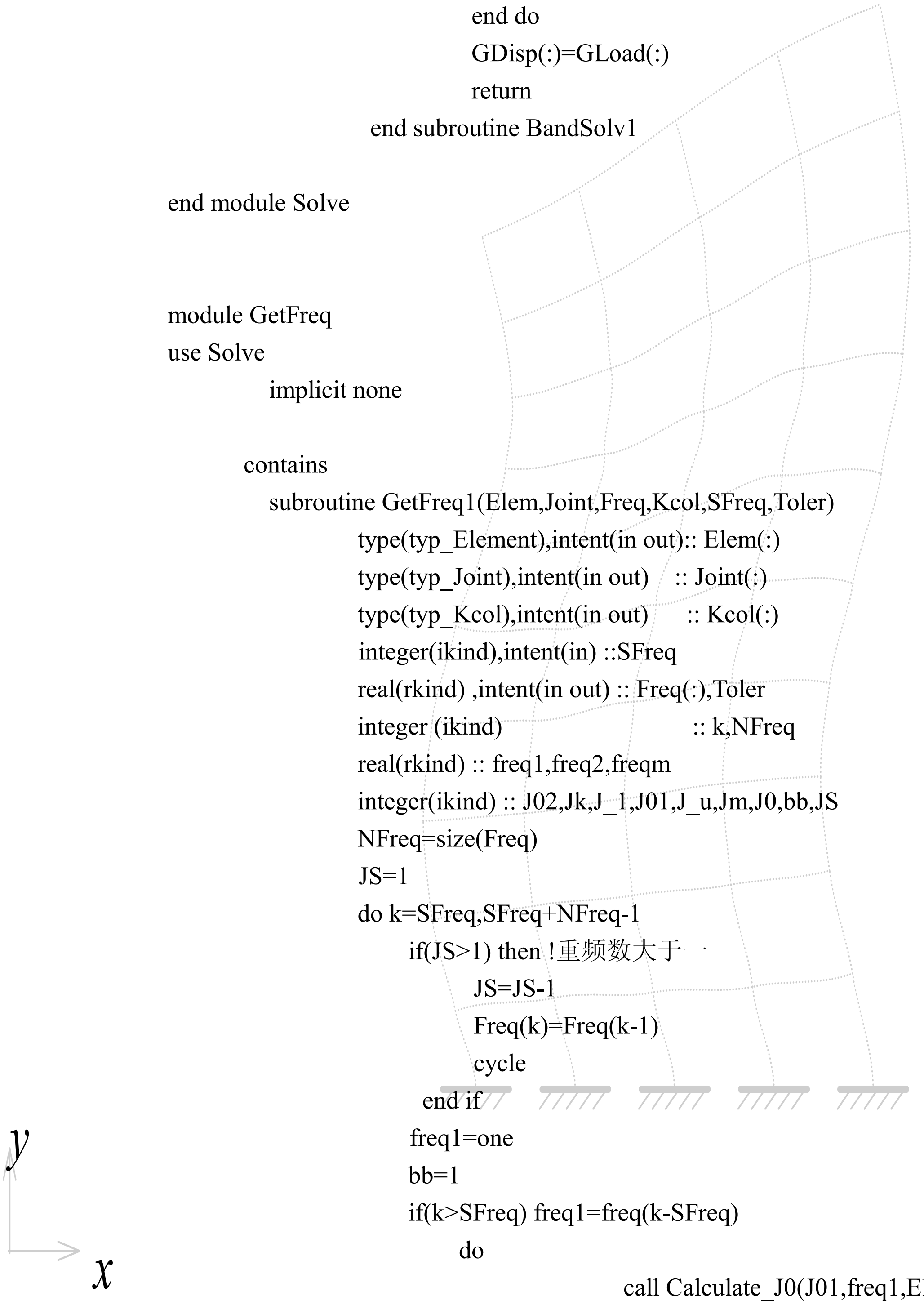
```
lambda=Elem(i)%Length*((freq2**2*Elem(i)%m/Elem(i)%EI)**0.25D0)
                        sg=sign(1D0,one-cosh(lambda)*cos(lambda))
                        j=int(lambda/PI)
                        Jb=j-(1-(-1)**j*sg)/2
                        J02=J02+Ja+Jb
                end do
                return
        end subroutine Calculate_J0

        subroutine Calculate_Jk(Jk,freq2,Kcol,Elem) ! 计算低于 freq2 的整体频率数
                integer(ikind) ,intent(out):: Jk
                real(rkind),intent (in)::freq2
                type(typ_Kcol),intent (in out):: Kcol(:)
                type(typ_Element),intent(in out)::Elem(:)
                !real(rkind) :: EK(NDOF*NNode,NDOF*NNode)
                real(rkind),allocatable :: diag(:)
                integer(ikind) :: NGLbDOF
                NGlbDOF=size(Kcol,dim=1)
                allocate(diag(NGlbDOF))
                call SetMatBand(Elem,Kcol)
                call SetElemEK(Elem,freq2)
                call GStifMat(Elem,Kcol)
                call BandSolv(Kcol,diag)
                Jk=count(mask=diag<zero,dim=1)
                return
        end subroutine Calculate_Jk

        subroutine GStifMat(Elem,Kcol)  ! 形成整体刚度矩阵
                type(typ_Element),intent(in):: Elem(:)
                type(typ_Kcol),intent(out):: Kcol(:)
                real(rkind)::EK(NNode*NDOF,NNode*NDOF)
                integer(ikind)::ie,j,JGDOF
                integer (ikind)::ELocVec(NNode*NDOF)
                do j=1,size(Kcol,dim=1)
                        Kcol(j)%row(:)=zero
                end do
                do ie=1,size(Elem,dim=1)
                        EK=Elem(ie)%EK
                        EK=matmul(transpose(Elem(ie)%ET),matmul(EK,Elem(ie)%ET))
                        ELocVec(:)=Elem(IE)%GlbDOF(:)
                        do j=1,6
                                JGDOF=ELocVec(j)
                                if (JGDOF==0) cycle
                                where (ELocVec>0.and.ELocVec<=JGDOF)
```

```
Kcol(JGDOF)%row(ELocVec)=Kcol(JGDOF)%row(ELocVec)+EK(:,j)
                        end where
                    end do
                end do
                return
        end subroutine GStifMat

        subroutine SetMatBand(Elem,Kcol) ! 设置整体刚度矩阵带宽
                    type(typ_Element),intent(in) :: Elem(:)
                    type(typ_Kcol),intent(in out)::Kcol(:)
                    integer (ikind) :: minDOF
                    integer (ikind),allocatable    :: Row1(:)
                    integer (ikind) :: ie,j
                    integer (ikind),allocatable::ELocVec(:)
                    integer (ikind) :: NElem,NGlbDOF
                    NElem=size(Elem,dim=1)
                    NGlbDOF=size(Kcol,dim=1)
                    allocate (Row1(NGlbDOF))
                    allocate(ELocVec(size(Elem(1)%GlbDOF)))
                    Row1=NGlbDOF
                    do ie=1,NElem
                      ELocVec(:)=Elem(ie)%GlbDOF(:)
                      minDOF=minval(ELocVec,mask=ELocVec>0)
                      where(ELocVec>0)
                          Row1(ELocVec)=min(Row1(ELocVec),minDOF)
                      end where
                  end do
                  do j=1,NGlbDOF
                      allocate (Kcol(j)%row(Row1(j):j))
                      Kcol(j)%row=Zero
                  end do
                return
        end subroutine SetMatBand

        subroutine SetElemEK(Elem,freq2) ! 设置单元刚度矩阵
                type(typ_Element),intent(in out)::Elem(:)
                real(rkind),intent(in)::freq2
                real(rkind):: nu,lambda,T,R,Q,H,S,C
                real(rkind) ::EI,EA,l,m
                REAL(rkind) :: lxzsin,lxzcos,lxzsh,lxzch,lxzt
                integer(ikind)::i
                do i=1,size(Elem,dim=1)
                    EI=Elem(i)%EI
```

```
            EA=Elem(i)%EA
            l=Elem(i)%Length
            m=Elem(i)%m
            nu=freq2*l*sqrt(m/EA)
            lambda=l*((freq2**2*m/EI)**0.25D0)
            lxzsin=sin(lambda)
            lxzsh=sinh(lambda)
            lxzcos=cos(lambda)
            lxzch=cosh(lambda)
            lxzt=one-lxzch*lxzcos
            T=lambda**3*(lxzsin*lxzch+lxzcos*lxzsh)/lxzt
            R=lambda**3*(lxzsh+lxzsin)/lxzt
            Q=lambda**2*(lxzsh*lxzsin)/lxzt
            H=lambda**2*(lxzch-lxzcos)/lxzt
            S=lambda*(lxzsin*lxzch-lxzcos*lxzsh)/lxzt
            C=lambda*(lxzsh-lxzsin)/lxzt
            Elem(i)%EK=zero
            Elem(i)%EK(1,1)=nu*EA/(tan(nu)*l)
            Elem(i)%EK(2,2)=T*EI/l**3
            Elem(i)%EK(3,3)=S*EI/l
            Elem(i)%EK(4,4)=Elem(i)%EK(1,1)
            Elem(i)%EK(5,5)=Elem(i)%EK(2,2)
            Elem(i)%EK(6,6)=Elem(i)%EK(3,3)
            Elem(i)%EK(2,3)=Q*EI/l**2
            Elem(i)%EK(3,2)=Elem(i)%EK(2,3)
            Elem(i)%EK(5,6)=-Elem(i)%EK(2,3)
            Elem(i)%EK(6,5)=Elem(i)%EK(5,6)
            Elem(i)%EK(1,4)=-nu*EA/(sin(nu)*l)
            Elem(i)%EK(4,1)=Elem(i)%EK(1,4)
            Elem(i)%EK(2,5)=-R*EI/l**3
            Elem(i)%EK(5,2)=Elem(i)%EK(2,5)
            Elem(i)%EK(3,5)=-H*(EI/l**2)
            Elem(i)%EK(5,3)=Elem(i)%EK(3,5)
            Elem(i)%EK(6,2)=-Elem(i)%EK(3,5)
            Elem(i)%EK(2,6)=Elem(i)%EK(6,2)
            Elem(i)%EK(3,6)=C*EI/l
            Elem(i)%EK(6,3)=Elem(i)%EK(3,6)
        end do
        return
    end subroutine SetElemEK

    subroutine BandSolv(Kcol,diag)  ！ 求解上三角阵
        type(typ_Kcol),intent(in out)::Kcol(:)
        real(rkind) ,intent(in out):: diag(:)
```

```
                integer(ikind)::row1,ncol,row,j,ie,NGlbDOF
                real(rkind)::s
                NGlbDOF=size(Kcol,dim=1)
                ncol=NGlbDOF
                diag(1:ncol)=(/(Kcol(j)%row(j),j=1,ncol)/)
                do j=2,ncol
                    row1=lbound(Kcol(j)%row,1)
                    do ie=row1,j-1
                        row=max(row1,lbound(Kcol(ie)%row,1))
s=sum(diag(row:ie-1)*Kcol(ie)%row(row:ie-1)*Kcol(j)%row(row:ie-1))
                        Kcol(j)%row(ie)=(Kcol(j)%row(ie)-s)/diag(ie)
                    end do
                    s=sum(diag(row1:j-1)*Kcol(j)%row(row1:j-1)**2)
                    diag(j)=diag(j)-s
                end do
                return
        end subroutine BandSolv

    !-------------------------------------
    subroutine BandSolv1(Kcol,GLoad,GDisp)  ! 求解矩阵
    !-------------------------------------
                    type(typ_Kcol),intent(in out)::Kcol(:)
                  real(rkind) ,intent(in out):: GLoad(:),GDisp(:)
                    real(rkind) ::diag(size(GLoad))
                  integer(ikind)::row1,ncol,row,j,ie,NGlbDOF
                  real(rkind)::s
                  NGlbDOF=size(Kcol,dim=1)
                  ncol=NGlbDOF
                    diag(1:ncol)=(/(Kcol(j)%row(j),j=1,ncol)/)
                    do j=2,ncol
                        row1=lbound(Kcol(j)%row,1)
                        do ie=row1,j-1
                            row=max(row1,lbound(Kcol(ie)%row,1))
s=sum(diag(row:ie-1)*Kcol(ie)%row(row:ie-1)*Kcol(j)%row(row:ie-1))
                            Kcol(j)%row(ie)=(Kcol(j)%row(ie)-s)/diag(ie)
                        end do
                        s=sum(diag(row1:j-1)*Kcol(j)%row(row1:j-1)**2)
                        diag(j)=diag(j)-s
                    end do
                    do ie=2,ncol
                        row1=lbound(Kcol(ie)%row,dim=1)
```

```
GLoad(ie)=GLoad(ie)-sum(Kcol(ie)%row(row1:ie-1)*GLoad(row1:ie-1))
                    end do
                    GLoad(:)=GLoad(:)/diag(:)
                    do j=ncol,2,-1
                          row1=lbound(Kcol(j)%row,dim=1)

GLoad(row1:j-1)=GLoad(row1:j-1)-GLoad(j)*Kcol(j)%row(row1:j-1)
                    end do
                    GDisp(:)=GLoad(:)
                    return
              end subroutine BandSolv1

end module Solve


module GetFreq
use Solve
      implicit none

    contains
      subroutine GetFreq1(Elem,Joint,Freq,Kcol,SFreq,Toler)
            type(typ_Element),intent(in out):: Elem(:)
            type(typ_Joint),intent(in out)   :: Joint(:)
            type(typ_Kcol),intent(in out)     :: Kcol(:)
            integer(ikind),intent(in) ::SFreq
            real(rkind) ,intent(in out) :: Freq(:),Toler
            integer (ikind)                         :: k,NFreq
            real(rkind) :: freq1,freq2,freqm
            integer(ikind) :: J02,Jk,J_1,J01,J_u,Jm,J0,bb,JS
            NFreq=size(Freq)
            JS=1
            do k=SFreq,SFreq+NFreq-1
                if(JS>1) then !重频数大于一
                    JS=JS-1
                    Freq(k)=Freq(k-1)
                    cycle
                 end if
                freq1=one
                bb=1
                if(k>SFreq) freq1=freq(k-SFreq)
                    do
                              call Calculate_J0(J01,freq1,Elem)
                              call Calculate_Jk(Jk,freq1,Kcol,Elem)
                              J_1=J01+JK
```



```
                        if(J_1<k) exit
                        freq1=freq1/two
                end do
                freq2=freq1+ten**bb
                do
                        call Calculate_J0(J02,freq2,Elem)
                        call Calculate_Jk(Jk,freq2,Kcol,Elem)
                        J_u=J02+JK
                        if(J_u>=k) exit
                        bb=bb+1
                        freq1=freq2
                        freq2=freq2+ten**bb
                end do
                do
                    freqm=(freq1+freq2)*0.5D0
                    call Calculate_J0(J0,freqm,Elem)
                    call Calculate_Jk(Jk,freqm,Kcol,Elem)
                    Jm=J0+JK
                    if((freq2-freq1)<=Toler*(one+freq2)) then
                        call Calculate_J0(J0,freq2,Elem)
                        call Calculate_Jk(Jk,freq2,Kcol,Elem)
                        Jm=J0+JK
                        JS=Jm-k+1！ 得到重频数
                        write(*,*) JS
                    call GetModule(JS,freqm,freq1,freq2,Elem,Joint,Kcol,Toler)
                        exit
                    end if
                        if(Jm>=k) then
                            freq2=freqm
                            J_u=Jm
                            J02=J0
                        else
                            freq1=freqm
                            J_1=Jm
                            J01=J0
                        end if
                end do
                Freq(k+1-SFreq)=freqm
            end do
            return
end subroutine GetFreq1

subroutine GetMDisp(GDisp,FElem,FJoint,freq1,freq2)
```

```
                type(typ_Element),intent(in out):: FElem(:)
                type(typ_Joint),intent(in out) :: FJoint(:)
                real(rkind),intent(in out)::GDisp(:)
                real(rkind),intent(in out):: freq1,freq2

                real(rkind)::D(6)
                real(rkind)::Disp1(size(GDisp)),Disp2(size(GDisp))
                integer(ikind):: i
                Disp1=zero
                Disp2=zero
                call SetElemEK(FElem,freq1)
                do i=1,size(FElem)
                    FElem(i)%EK=matmul(transpose(FElem(i)%ET),&
                            matmul(FElem(i)%EK,FElem(i)%ET))
                   D=zero
                    where(FElem(i)%GlbDOF>0)
                        D=GDisp(FElem(i)%GlbDOF)
                    end where
                    D=matmul(FElem(i)%EK,D)
                    where(FElem(i)%GlbDOF>0)
                        Disp1(FElem(i)%GlbDOF)=Disp1(FElem(i)%GlbDOF)+D
                    end where
                end do
                call SetElemEK(FElem,freq2)
                do i=1,size(FElem)
                    FElem(i)%EK=matmul(transpose(FElem(i)%ET),&
                            matmul(FElem(i)%EK,FElem(i)%ET))
                   D=zero
                    where(FElem(i)%GlbDOF>0)
                        D=GDisp(FElem(i)%GlbDOF)
                    end where
                    D=matmul(FElem(i)%EK,D)
                    where(FElem(i)%GlbDOF>0)
                        Disp2(FElem(i)%GlbDOF)=Disp2(FElem(i)%GlbDOF)+D
                    end where
                end do
                GDisp=Disp2-Disp1
                 return
end subroutine GetMDisp


function Getlambda(GDisp,NGlbDOF) result(lambda)
                real(rkind),intent(in)::GDisp(:)
                integer(ikind),intent(in)::NGlbDOF
```

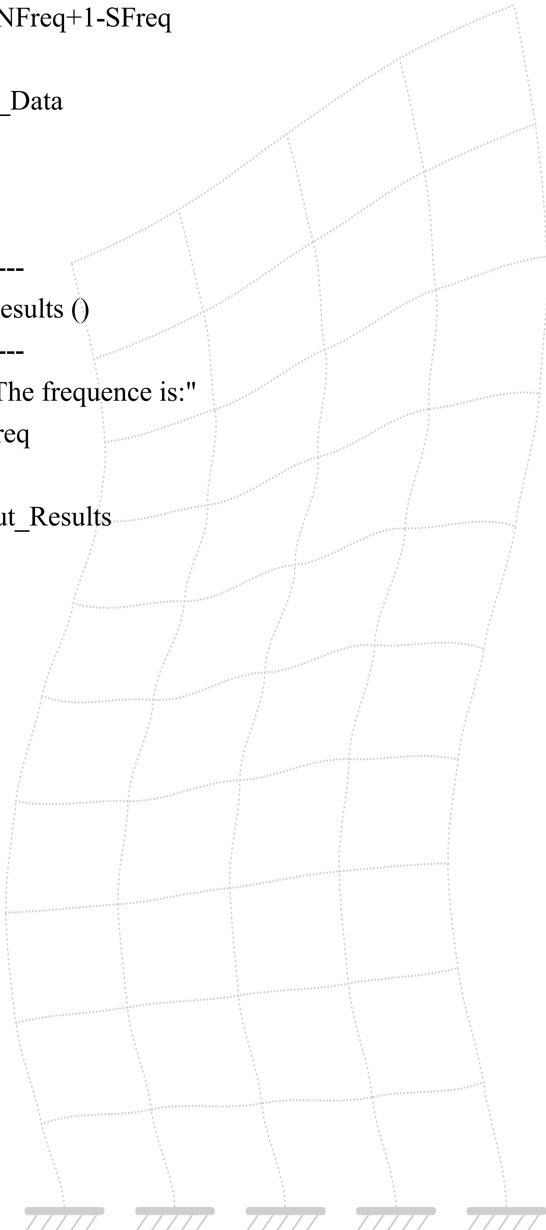
```
                real(rkind) :: lambda
                integer(ikind)::i
                lambda=maxval(abs(GDisp))
                return
        end function Getlambda

    subroutine GetModule(JS,freqm,freq1,freq2,Elem,Joint,Kcol,Toler)
        integer(ikind),intent(in)::JS
        type(typ_Element),intent(in out):: Elem(:)
        type(typ_Joint),intent(in out)    :: Joint(:)
        type(typ_Kcol),intent(in out) ::Kcol(:)
        real(rkind),intent (in) :: Toler
        type (typ_Element),allocatable      :: FElem(:)
        type (typ_Joint),allocatable                    :: FJoint(:)
        type (typ_Kcol),allocatable    ::FKcol(:)
        real(rkind),intent(in out) :: freq1,freq2,freqm
        integer(ikind) :: FixNum(Size(Elem))
        integer(ikind) ::NElem,NJoint,NGlbDOF
        integer(ikind) ::FNElem,FNJoint,FNGlbDOF
        integer(ikind)    :: i,j,k,n1,n2
        real(rkind),allocatable    :: GDisp(:,:),GLoad(:)
        real(rkind) :: lambda,lambda1
        call FindFix(FixNum,freq1,freq2,Elem)

        write(*,*) FixNum

          NElem=size(Elem)
          NJoint=size(Joint)
          NGlbDOF=size(Kcol)
          FNElem=NElem+count(mask=FixNum.ne.0,dim=1)
          FNJoint=NJoint+count(mask=FixNum.ne.0,dim=1)
          FNGlbDOF=NGlbDOF+3*count(mask=FixNum.ne.0,dim=1)
          allocate(FJoint(FNJoint))
          allocate(FElem(FNElem))
          allocate(FKcol(FNGlbDOF))
          allocate(GDisp(FNGlbDOF,JS))
          allocate(GLoad(FNGlbDOF))
          FJoint(1:NJoint)=Joint(:)
          FElem(1:NElem)=Elem(:)
          j=1
          do i=1,NElem
                  if(FixNum(i).ne.0) then
                          n1=FElem(i)%JointNo(1)
                          n2=FElem(i)%JointNo(2)
```

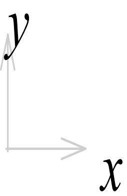
```
                    FJoint(NJoint+j)%X=(FJoint(n2)%X+FJoint(n1)%X)/(two)
                    FJoint(NJoint+j)%Y=(FJoint(n2)%Y+FJoint(n1)%Y)/(two)
                    FJoint(NJoint+j)%GDOF(1)=NGlbDOF+(j-1)*3+1
                    FJoint(NJoint+j)%GDOF(2)=NGlbDOF+(j-1)*3+2
                    FJoint(NJoint+j)%GDOF(3)=NGlbDOF+(j-1)*3+3
                    FElem(i)%JointNo(2)=NJoint+j
                    FElem(NElem+j)%JointNo(1)=NJoint+j
                    FElem(NElem+j)%JointNo(2)=n2
                     FElem(NElem+j)%EA=FElem(i)%EA
                     FElem(NElem+j)%EI=FElem(i)%EI
                     FElem(NElem+j)%m=FElem(i)%m
                     j=j+1
            end if
        end do

        call SetElemProp(FElem,FJoint)

    do k=1,JS！ 重频处理

        call random_number(GDisp(:,k))
        call    Zhengjiao(GDisp,k,FElem,FJoint,freq1,freq2)！ 振型正交化
        lambda=Getlambda(GDisp(:,k),NGlbDOF)
        GDisp(:,k)=GDisp(:,k)/lambda
        call GetMDisp(GDisp(:,k),FElem,FJoint,freq1,freq2)
        call SetElemEK(FElem,freqm)
        call SetMatBand(FElem,FKcol)
        call GStifMat(FElem,FKcol)
        GLoad=GDisp(:,k)
        call BandSolv1(FKcol,GLoad,GDisp(:,k))
        lambda=Getlambda(GDisp(:,k),NGlbDOF)
        lambda1=lambda
        do
                GDisp(:,k)=GDisp(:,k)/lambda
                call GetMDisp(GDisp(:,k),FElem,FJoint,freq1,freq2)
                call SetElemEK(FElem,freq2)
                call GStifMat(FElem,FKcol)
                GLoad=GDisp(:,k)
                call BandSolv1(FKcol,GLoad,GDisp(:,k))
                lambda=Getlambda(GDisp(:,k),NGlbDOF)
                if(abs(lambda1-lambda)<abs((one+lambda)*Toler)) exit
                lambda1=lambda
        end do
        GDisp(:,k)=GDisp(:,k)/lambda
        write(*,*) GDisp(:,k)
```

```
                write(55,*) freqm,count(mask=FixNum.ne.0,dim=1)
                j=1
                do i=1,NElem
                    if (FixNum(i)>0) then
                        write(55,*) i,FixNum(i),1/(two)
                        write(*,*)
GDisp(NGlbDOF+(j-1)*3+1,k),GDisp(NGlbDOF+(j-1)*3+2,k),GDisp(NGlbDOF+(j-1)*3+3,k)
                        write(55,*)
GDisp(NGlbDOF+(j-1)*3+1,k),GDisp(NGlbDOF+(j-1)*3+2,k),GDisp(NGlbDOF+(j-1)*3+3,k)
                        j=j+1
                    end if
                end do
                do i=1,NElem
                    do j=1,6
                        if(Elem(i)%GlbDOF(j)>0) then
                        write(55,*) GDisp(Elem(i)%GlbDOF(j),k)
                    else
                        write(55,*) zero
                    end if
                end do
            end do
        end do
            deallocate(FElem)
            deallocate(FJoint)
            deallocate(FKcol)
            deallocate(GLoad)
            deallocate(GDisp)
            return
    end subroutine GetModule

    subroutine Zhengjiao(GDisp,k,FElem,FJoint,freq1,freq2)！振型正交化
            real(rkind),intent(in out)::GDisp(:,:)
            integer(ikind),intent(in) :: k
            type(typ_Element),intent(in out):: FElem(:)
            type(typ_Joint),intent(in out) :: FJoint(:)
            real(rkind),intent(in out):: freq1,freq2
            integer(ikind)::i
            real(rkind):: alpha(k-1)
            real(rkind):: x,y
            real(rkind):: delta1(size(GDisp,dim=1)),delta2(size(GDisp,dim=1))
            if(k>1) then
                do i=1,k-1
                    delta1=GDisp(:,k)
                    delta2=GDisp(:,i)
```

```
                call GetMDisp(delta1,FElem,FJoint,freq1,freq2)
                x=dot_product(GDisp(:,i),delta1)
                call GetMDisp(delta2,FElem,FJoint,freq1,freq2)
                y=dot_product(GDisp(:,i),delta2)
                alpha(i)=x/y
            end do
            do i=1,k-1
                GDisp(:,k)=GDisp(:,k)-GDisp(:,i)*alpha(i)
            end do
        end if
   end subroutine Zhengjiao

end module GetFreq
!==========================================================================
program SM_90                              ! main prog
!==========================================================================
   use GetFreq                    ! displacement method module
   implicit none
   integer (ikind)                        :: NElem,NJoint,NGlbDOF,NFreq,SFreq
   real(rkind):: Toler
   character (20) :: filename
   type (typ_Element),allocatable      :: Elem(:)
   type (typ_Joint),allocatable         :: Joint(:)
   type (typ_Kcol),allocatable    ::Kcol(:)
   real(rkind),allocatable :: Freq(:)
   call Input_Data ()             ! internal sub, see below
   call SetElemProp (Elem, Joint)
   call GetFreq1(Elem,Joint,Freq,Kcol,SFreq,Toler)
   call Output_Results () ! internal sub, see below

   stop

   contains

   !------------------------
   subroutine Input_Data ()
   !------------------------
          integer (ikind) :: i,ie
          character (20) :: filename
          open (5,file='sm90.ipt',status='OLD',position='REWIND')
          open (55,file='smcai90.out',position='REWIND')
          read(5,*) i
          read(5,*) NFreq,SFreq,Toler
          read(5,*) NElem,NJoint,NGlbDOF
```

```
        allocate (Joint(NJoint))
        allocate (Elem(NElem))
        allocate (Freq(NFreq+1-SFreq))
        allocate (Kcol(NGlbDOF))
        read(5,*) (Joint(i),i=1,NJoint)
        read(5,*) (Elem(ie)%JointNo,Elem(ie)%EA,Elem(ie)%EI,Elem(ie)%m,ie=1,NElem)
        write(55,*) 10,0
        write(55,*) NFreq+1-SFreq
        return
    end subroutine Input_Data



    !----------------------------
    subroutine Output_Results ()
    !----------------------------
        write(*,*) "The frequence is:"
        write(*,*) Freq
        return
    end subroutine Output_Results

end program SM_90
```





程序说明：

1：本程序可以进行各种平面构件的几何稳定性分析

2：本程序输入接口与结构力学求解器教学版文件格式相同可直接使用其前处理成果

程序清单：

```
!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
```

考题一输入文件：

```
4 8 12 0 0
 0.000000000000000E+00 0.000000000000000E+00 1 0 2
 0.000000000000000E+00 0.000000000000000E+00 1 0 11
 1.000000000000000 0.000000000000000E+00 0 3 4
 1.000000000000000 0.000000000000000E+00 0 3 5
 0.000000000000000E+00 1.000000000000000 0 8 9
 0.000000000000000E+00 1.000000000000000 0 8 12
 1.000000000000000 1.000000000000000 6 0 7
 1.000000000000000 1.000000000000000 6 0 10
 1 3 1.000000000000000 1.000000000000000
 4 7 1.000000000000000 1.000000000000000
 5 8 1.000000000000000 1.000000000000000
 2 6 1.000000000000000 1.000000000000000
```

运行结果：

静定结构,无多余约束

考题二输入文件：

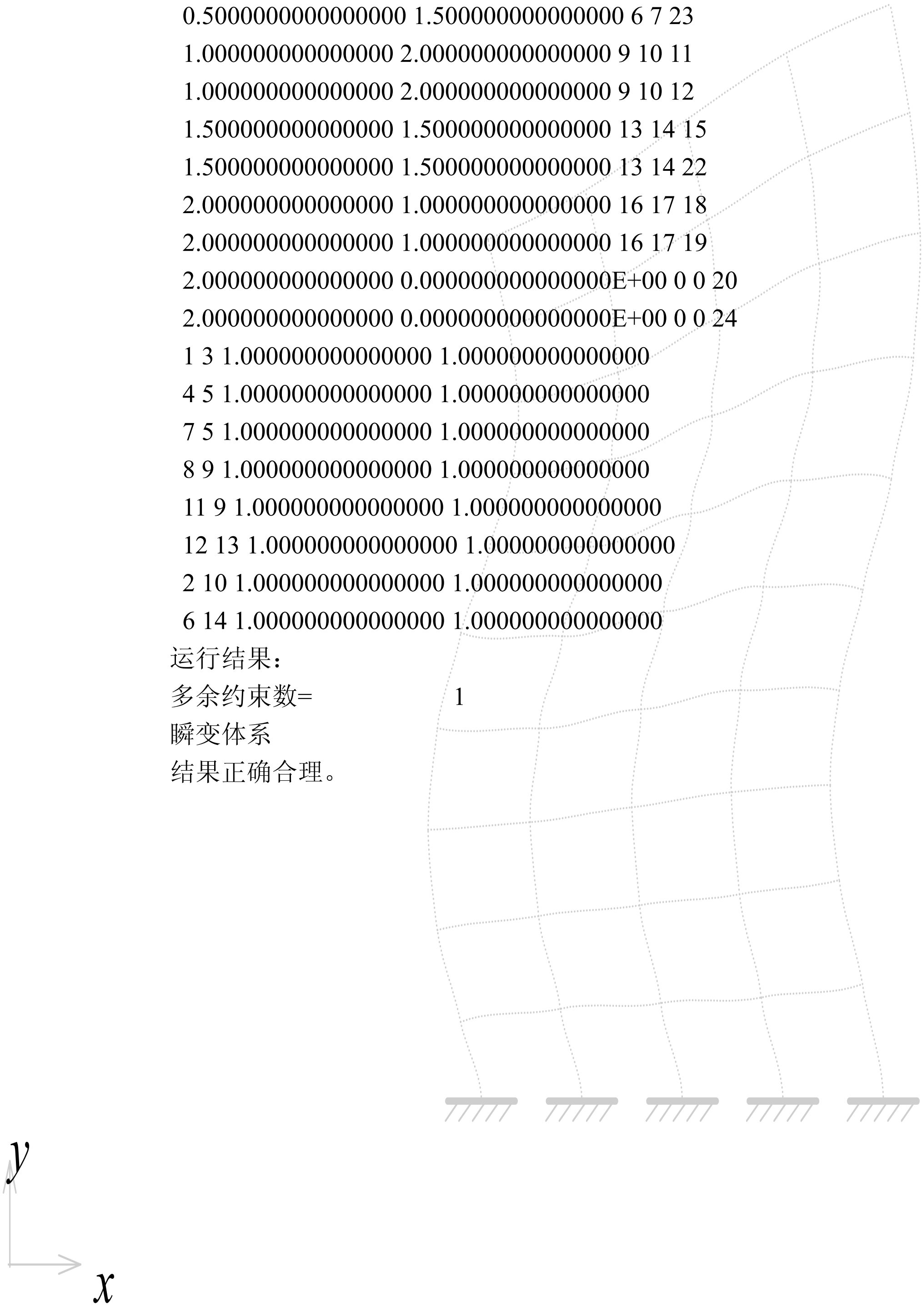
```
8 14 24 0 0
 0.000000000000000E+00 0.000000000000000E+00 0 0 1
 0.000000000000000E+00 0.000000000000000E+00 0 0 21
 0.000000000000000E+00 1.000000000000000 2 3 4
 0.000000000000000E+00 1.000000000000000 2 3 5
 0.5000000000000000 1.500000000000000 6 7 8
 0.5000000000000000 1.500000000000000 6 7 23
 1.000000000000000 2.000000000000000 9 10 11
 1.000000000000000 2.000000000000000 9 10 12
 1.500000000000000 1.500000000000000 13 14 15
 1.500000000000000 1.500000000000000 13 14 22
 2.000000000000000 1.000000000000000 16 17 18
 2.000000000000000 1.000000000000000 16 17 19
 2.000000000000000 0.000000000000000E+00 0 0 20
 2.000000000000000 0.000000000000000E+00 0 0 24
 1 3 1.000000000000000 1.000000000000000
 4 5 1.000000000000000 1.000000000000000
 7 5 1.000000000000000 1.000000000000000
 8 9 1.000000000000000 1.000000000000000
 11 9 1.000000000000000 1.000000000000000
 12 13 1.000000000000000 1.000000000000000
 2 10 1.000000000000000 1.000000000000000
 6 14 1.000000000000000 1.000000000000000
```

运行结果：

多余约束数=　　　　1

瞬变体系

结果正确合理。



程序说明：

1：编程语言为 Fortran 90
2：本程序可以求解任意空间杆系结构
3：节点位移编码为（X, Y, Z, θx, θy, θz），荷载编码为（Fx, Fy, Fz, Mx, My, Mz）
4：单元输入信息为：起点号，终点号，EA, EIy, EIz, GIx, α(α为参考坐标系与整体坐标系夹角
5：程序为使编程简化，采用了多文件技术

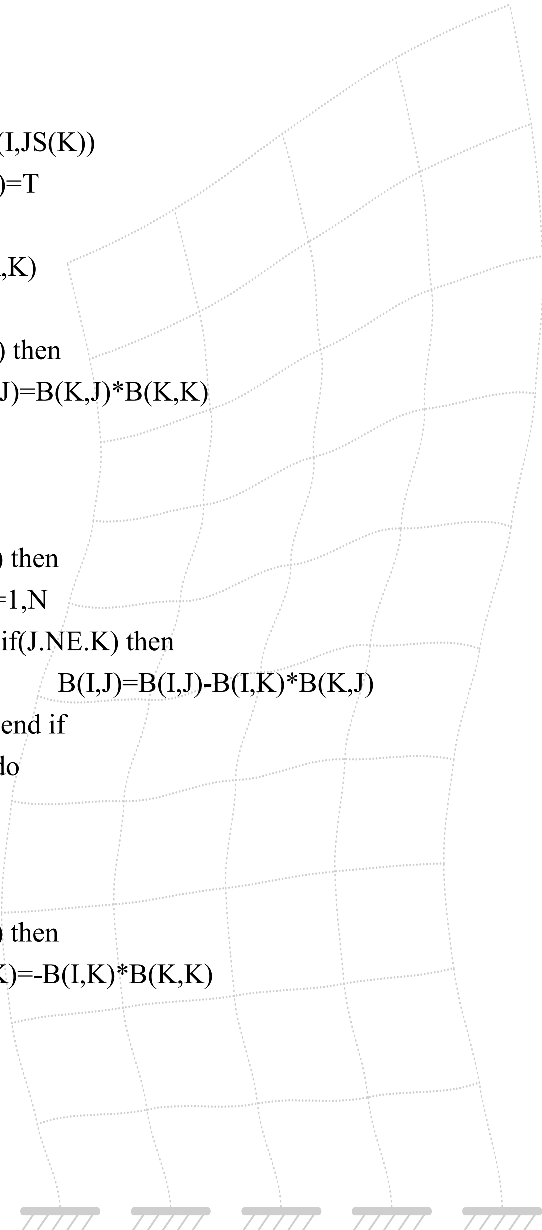
程序清单：

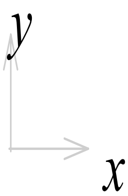
程序由以下四个文件组成
1：Lxz_Tools.f90 ----- 主要为一些工具函数
2：TypeDef.f90 ----- 变量定义，单元属性分析
3：SolveDisp.f90 ----- 矩阵求解
4：3dframe.f90 ----- 整体控制模块

1：Lxz_Tools.f90

```
module Lxz_Tools
      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

      contains
      function matinv(A) result (B)
       real(rkind) ,intent (in)::A(:,:)
       !real(rkind) , allocatable::B(:,:)
       real(rkind) , pointer::B(:,:)
       integer(ikind):: N,I,J,K
       real(rkind)::D,T
       real(rkind), allocatable::IS(:),JS(:)
       N=size(A,dim=2)
       allocate(B(N,N))
       allocate(IS(N));allocate(JS(N))
       B=A
       do   K=1,N
            D=0.0D0
            do I=K,N
                  do J=K,N
                        if(abs(B(I,J))>D) then
                              D=abs(B(I,J))
                              IS(K)=I
                              JS(K)=J
```

```
            end if
        end do
    end do
    do J=1,N
        T=B(K,J)
        B(K,J)=B(IS(K),J)
        B(IS(K),J)=T
    end do
    do I=1,N
        T=B(I,K)
        B(I,K)=B(I,JS(K))
        B(I,JS(K))=T
    end do
    B(K,K)=1/B(K,K)
    do J=1,N
        if(J.NE.K) then
            B(K,J)=B(K,J)*B(K,K)
        end if
    end do
    do I=1,N
        if(I.NE.K) then
            do J=1,N
                if(J.NE.K) then
                    B(I,J)=B(I,J)-B(I,K)*B(K,J)
                end if
            end do
        end if
    end do
    do I=1,N
        if(I.NE.K) then
            B(I,K)=-B(I,K)*B(K,K)
        end if
    end do
end do
do K=N,1,-1
    do J=1,N
        T=B(K,J)
        B(K,J)=B(JS(K),J)
        B(JS(K),J)=T
    end do
    do I=1,N
        T=B(I,K)
        B(I,K)=B(I,IS(K))
        B(I,IS(K))=T
```





```
        end do
  end do
  return
 end function matinv

 subroutine IntSwap(a,b)
  integer(ikind),intent(in out)::a,b
  integer(ikind)::t
  t=a;a=b;b=t
 end subroutine IntSwap

 subroutine RealSwap(a,b)
  real(rkind),intent(in out)::a,b
  real(rkind)::t
  t=a;a=b;b=t
 end subroutine RealSwap

 subroutine matprint(A,n)
  real(rkind),intent(in)::A(:,:)
  integer(ikind)::n
  integer(ikind)::n1,n2
  integer(ikind)::i,j
  character(10)::C
  n1=size(A,dim=1)
  n2=size(A,dim=2)
  C='('//trim(itoc(n2))//'E'//trim(itoc(n))//&
  '.'//trim(itoc(n-7))//')'
  do I=1,n1
        write(*,C)(A(I,J),J=1,n2)
  end do
 end subroutine matprint

 function matdet(B) result(det)
  real(rkind),intent(in)::B(:,:)
  real(rkind)::det
  integer(ikind)::n,i,j,k,is,js
  real(rkind),pointer::A(:,:)
  real(rkind)::f,d,q
  n=size(B,dim=1)
  allocate (A(n,n))
  A=B
  f=1.0D0;      det=1.0D0
  do k=1,n-1
        q=0.0D0
```

```
            do i=k,n
                do j=k,n
                    if(abs(a(i,j)).gt.q) then
                        q=abs(a(i,j))
                        is=i
                        js=j
                    end if
                end do
            end do
            if(q+1.0D0.eq.1.0D0) then
                det=0.0d0
                return
            end if
            if(is.ne.k) then
                f=-f
                do j=k,n
                    d=a(k,j)
                    a(k,j)=a(is,j)
                    a(is,j)=d
                end do
            end if
            if(js.ne.k) then
                f=-f
                do i=k,n
                    d=a(i,js)
                    a(i,js)=a(i,k)
                    a(i,k)=d
                end do
            end if
            det=det*a(k,k)
            do i=k+1,n
                d=a(i,k)/a(k,k)
                do j=k+1,n
                    a(i,j)=a(i,j)-d*a(k,j)
                end do
            end do
        end do
        det=f*det*a(n,n)
        deallocate (a)
        return
    end function matdet

    function itoc(i1) result (c)
        integer(ikind),intent(in)::i1
```

```
      character(len=2)::c
      real(rkind)::x
      integer(ikind) ::n,b,i,j
      i=i1
      x=i
      c(1:2)=' '
      x=log10(x)
      n=int(x)+2
      do j=n-2,0,-1
            b=mod(i,10**j)
            b=(i-b)/(10**j)
            i=i-b*(10**j)
            c(n-j-1:n-j-1)=achar(iachar('0')+b)
      end do
     end function itoc

     subroutine Gauss(GStif,GLoad,GDisp)
      real (rkind),intent (in) :: GStif(:,:),GLoad(:)
         real (rkind),intent (out) :: GDisp(:)
         integer (ikind) :: i,j,k
         integer (ikind) :: N
         real (rkind) :: P,I1,X,Y
         real (rkind),allocatable :: A(:,:)
         N=size(GDisp,dim=1)
         allocate (A(N,N+1))
         A(1:N,1:N)=GStif(1:N,1:N)
         A(1:N,N+1)=GLoad(1:N)
         DO    j=1,N
            P=0.0D0
         DO k=j,N
              IF(ABS(A(k,j)).LE.P) cycle
              P=ABS(A(k,j))
              I1=k
          end do
          IF(P.GE.1E-15)GO TO 230
          WRITE(22,'(A)') 'NO UNIQUE SOLUTION'
          RETURN
230       IF(I1.EQ.j)GO TO 280
          DO 270 K=J,N+1
              X=A(J,K)
              A(J,K)=A(I1,K)
270           A(I1,K)=X
280       Y=1.D0/A(J,J)
          DO 310 K=J,N+1
```

```
310          A(J,K)=Y*A(J,K)
        DO 380 I=1,N
            IF(I.EQ.J)GO TO 380
            Y=-A(I,J)
            DO 370 K=J,N+1
370            A(I,K)=A(I,K)+Y*A(J,K)
380     CONTINUE
390 end do

     GDisp=A(1:N,N+1)
     end subroutine Gauss

end module Lxz_Tools
```

```
2：TypeDef.f90
include 'Lxz_Tools.f90'
module TypeDef
    use Lxz_Tools
    implicit none
    integer(ikind),parameter :: NDOF=6,NNode=2

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

    type::typ_Element
     integer(ikind):: JointNo(NNode)
     real(rkind):: EIy,EIz,EA,GIp,Length,CosA,CosB,CosC,A
     real(rkind)::EK(NDOF*NNode,NDOF*NNode),ET(NDOF*NNode,NDOF*NNode)
     integer(ikind)::GlbDOF(NDOF*NNode)
     real(rkind)::EForce(NDOF*NNode),ELForce(NDOF*NNode)
    end type typ_Element

    type::typ_JointLoad
     integer(ikind)::JointNo,LodDOF
     real(rkind)::LodVal
    end type typ_JointLoad

    type::typ_ElemLoad
     integer (ikind):: ElemNo,Indx
     real(rkind)::Pos,LodVal
    end type typ_ElemLoad
```

```
contains

subroutine SetElemProp(Elem,Joint)
 type(typ_Element),intent(inout)::Elem(:)
 type(typ_Joint),intent(in)::Joint(:)
 integer(ikind):: i,EJ1,EJ2
 real(rkind)::T(3,3),x,y,z
 real(rkind)::cx,cy,cz,cs,ca,sa
 do i=1,size(Elem)
     Elem(i)%EK=zero;Elem(i)%ET=zero;T=zero
     EJ1=Elem(i)%JointNo(1);EJ2=Elem(i)%JointNo(2)
     Elem(i)%GlbDOF(1:6)=Joint(EJ1)%GDOF(:)
     Elem(i)%GlbDOF(7:12)=Joint(EJ2)%GDOF(:)
     x=Joint(EJ2)%X-Joint(EJ1)%X
     y=Joint(EJ2)%Y-Joint(EJ1)%Y
     z=Joint(EJ2)%Z-Joint(EJ1)%Z
     Elem(i)%Length=sqrt(x**2+y**2+z**2)
     Elem(i)%CosA=x/Elem(i)%Length
     Elem(i)%CosB=y/Elem(i)%Length
     Elem(i)%CosC=z/Elem(i)%Length
     cx=Elem(i)%CosA;cy=Elem(i)%CosB;cz=Elem(i)%CosC
     ca=cos(Elem(i)%A);sa=sin(Elem(i)%A)
     cs=sqrt(cx**2+cy**2)
     if(cs.NE.zero) then
         T(1,1)=cx;T(1,2)=cy;T(1,3)=cz;
         T(2,1)=-(ca*cy+sa*cx*cz)/cs;
         T(2,2)=(ca*cx-sa*cy*cz)/cs;
         T(2,3)=cs*sa;
         T(3,1)=(sa*cy-ca*cx*cz)/cs;
         T(3,2)=-(sa*cx+ca*cy*cz)/cs;
         T(3,3)=cs*ca;
     else
         T(1,3)=one;
         T(2,1)=-sa;T(2,2)=ca;
         T(3,1)=-ca;T(3,2)=-sa;
     end if
     Elem(i)%ET(1:3,1:3)=T;Elem(i)%ET(4:6,4:6)=T;
     Elem(i)%ET(7:9,7:9)=T;Elem(i)%ET(10:12,10:12)=T;
     T=zero
     T(1,1)=Elem(i)%EA/Elem(i)%Length
     T(2,2)=12D0*Elem(i)%EIz/(Elem(i)%Length**3)
     T(3,3)=12D0*Elem(i)%EIy/(Elem(i)%Length**3)
     Elem(i)%EK(1:3,1:3)=T
     Elem(i)%EK(7:9,7:9)=T
```

```
        Elem(i)%EK(1:3,7:9)=-T
        Elem(i)%EK(7:9,1:3)=transpose(-T)
        T=zero
        T(1,1)=Elem(i)%GIp/Elem(i)%Length
        T(2,2)=4D0*Elem(i)%EIy/Elem(i)%Length
        T(3,3)=4D0*Elem(i)%EIz/Elem(i)%Length
        Elem(i)%EK(4:6,4:6)=T;Elem(i)%EK(10:12,10:12)=T
        T=zero
        T(2,3)=6D0*Elem(i)%EIz/(Elem(i)%Length**2)
        T(3,2)=-6D0*Elem(i)%EIy/(Elem(i)%Length**2)
        Elem(i)%EK(1:3,4:6)=T;Elem(i)%EK(1:3,10:12)=T
        Elem(i)%EK(4:6,1:3)=transpose(T)
        Elem(i)%EK(10:12,1:3)=transpose(T)
        Elem(i)%EK(7:9,10:12)=-T
        Elem(i)%EK(10:12,7:9)=-transpose(T)
        T=zero
        T(2,3)=6D0*Elem(i)%EIy/(Elem(i)%Length**2)
        T(3,2)=-6D0*Elem(i)%EIz/(Elem(i)%Length**2)
        Elem(i)%EK(4:6,7:9)=T
        Elem(i)%EK(7:9,4:6)=transpose(T)
        T=zero
        T(1,1)=-Elem(i)%GIp/Elem(i)%Length
        T(2,2)=2D0*Elem(i)%EIy/Elem(i)%Length
        T(3,3)=2D0*Elem(i)%EIz/Elem(i)%Length
        Elem(i)%EK(4:6,10:12)=T
        Elem(i)%EK(10:12,4:6)=transpose(T)
        !call matprint(Elem(i)%EK,10)
        !write(16,*) Elem(i)%ET
        !write(*,*)
    end do
   end subroutine SetElemProp
end module TypeDef

3：SolveDisp.f90
include 'TypeDef.f90'
module Solve
   use TypeDef
   type :: typ_Kcol
        real(rkind),pointer :: row(:)
    end type typ_Kcol
   contains
   !==================================================
   subroutine SolveDisp (GDisp, Elem,Joint,GLoad)
    !==================================================
```

```
real(rkind),intent(out)        :: GDisp(:)
type (typ_Element),intent(in)  :: Elem(:)
type (typ_Joint),intent(in)    :: Joint(:)
real(rkind)                  :: GLoad(:)  !?
integer(ikind) NElem,NGlbDOF
type (typ_Kcol),allocatable    ::Kcol(:)
NElem = size(Elem,dim=1)
NGlbDOF = size(GDisp,dim=1)
allocate (Kcol(NGlbDOF))


call SetMatBand()
call GStifMat()
call BandSolv()
contains

  !--------------------------
subroutine SetMatBand()
  !--------------------------
    integer (ikind) :: minDOF
    integer (ikind),allocatable  :: Row1(:)
    integer (ikind) :: ie,j
    integer (ikind),allocatable::ELocVec(:)
    allocate (Row1(NGlbDOF))
    allocate(ELocVec(size(Elem(1)%GlbDOF)))
    Row1=NGlbDOF
    do ie=1,NElem
             ELocVec(:)=Elem(ie)%GlbDOF(:)
             minDOF=minval(ELocVec,mask=ELocVec>0)
             where(ELocVec>0)
                 Row1(ELocVec)=min(Row1(ELocVec),minDOF)
             end where
         end do
         do j=1,NGlbDOF
             allocate (Kcol(j)%row(Row1(j):j))
             Kcol(j)%row=Zero
         end do
     return
end subroutine SetMatBand

  !--------------------------------------
subroutine BandSolv()
  !--------------------------------------
    integer(ikind)::row1,ncol,row,j,ie
```

```
    real(rkind)::diag(1:NGlbDOF),s
    !integer (ikind)::ELocVec(NNode*NDOF)
    ncol=NGlbDOF
    diag(1:ncol)=(/(Kcol(j)%row(j),j=1,ncol)/)
    do j=2,ncol
        row1=lbound(Kcol(j)%row,1)
        do ie=row1,j-1
            row=max(row1,lbound(Kcol(ie)%row,1))

s=sum(diag(row:ie-1)*Kcol(ie)%row(row:ie-1)*Kcol(j)%row(row:ie-1))
            Kcol(j)%row(ie)=(Kcol(j)%row(ie)-s)/diag(ie)
        end do
        s=sum(diag(row1:j-1)*Kcol(j)%row(row1:j-1)**2)
        diag(j)=diag(j)-s
    end do
    do ie=2,ncol
        row1=lbound(Kcol(ie)%row,dim=1)
        GLoad(ie)=GLoad(ie)-sum(Kcol(ie)%row(row1:ie-1)*GLoad(row1:ie-1))
    end do
    GLoad(:)=GLoad(:)/diag(:)
    do j=ncol,2,-1
        row1=lbound(Kcol(j)%row,dim=1)
        GLoad(row1:j-1)=GLoad(row1:j-1)-GLoad(j)*Kcol(j)%row(row1:j-1)
    end do
    GDisp(:)=GLoad(:)
    return
end subroutine BandSolv

  !---------------------
subroutine GStifMat()
  !---------------------
    integer(ikind)::ie,j,JGDOF
    integer (ikind),allocatable::ELocVec(:)
    real(rkind),allocatable::EK(:,:),ET(:,:)
    integer(ikind)::n
    n=size(Elem(1)%GlbDOF)
    allocate(ELocVec(n))
    allocate(EK(n,n))
    allocate(ET(n,n))
    do IE=1,NElem
        EK=Elem(IE)%EK
        ET=Elem(IE)%ET
        EK = matmul(transpose(ET),matmul(EK,ET))
        !write(16,*) EK
```

```
            !write(16,*)
            ELocVec(:)=Elem(IE)%GlbDOF(:)
            do j=1,12
                JGDOF=ELocVec(j)
                if (JGDOF==0) cycle
                where (ELocVec>0.and.ELocVec<=JGDOF)
                    Kcol(JGDOF)%row(ELocVec)=Kcol(JGDOF)%row(ELocVec)+EK(:,j)
                end where
            end do
        end do
        return
      end subroutine GStifMat

   end subroutine SolveDisp
end module Solve
```

4：3dframe.f90

```
include 'SolveDisp.f90'

module DispMethod
    use Solve
    implicit none

    contains
    subroutine GLoadVec(Elem,Joint,JLoad,ELoad,GLoad)
     type (typ_Element),intent(inout)::Elem(:)
     type (typ_Joint),intent(in)::Joint(:)
     type (typ_JointLoad),intent(in)::JLoad(:)
     type (typ_ElemLoad),intent(in)::ELoad(:)
     real(rkind)::GLoad(:),EF(NNode*NDOF)
     integer (ikind)::i,m,n
     real(rkind)::a,b,l,q
     do i=1,size(JLoad)
         n=JLoad(i)%JointNo
         m=JLoad(i)%LodDOF
         m=Joint(n)%GDOF(m)
         GLoad(m)=GLoad(m)+JLoad(i)%LodVal
     end do
     do i=1,size(ELoad)
         n=ELoad(i)%ElemNo
         l=Elem(n)%Length
         a=ELoad(i)%Pos
         q=ELoad(i)%LodVal
         b=l-a
```

```
        EF=zero
        if(ELoad(i)%Indx.eq.1) then
            EF(5)=q*a*a*(6D0-8D0*a/l+3D0*a*a/(l*l))/12D0
            EF(11)=-q*(a**3D0)*(4D0-3D0*a/l)/(12D0*l)
            EF(3)=-0.5D0*q*a*(2D0-2D0*a*a/(l*l)+a**3D0/l**3D0)
            EF(9)=-0.5D0*q*a**3D0*(2D0-a/l)/l**2D0
        end if
        if(ELoad(i)%Indx.eq.2) then
            EF(5)=q*a*b**2D0/l**2D0
            EF(11)=-q*a*a*b/l**2D0
            EF(3)=-q*b**2D0*(1D0+2D0*a/l)/l**2D0
            EF(9)=-q*a**2D0*(1D0+2D0*b/l)/l**2D0
        end if
        if(ELoad(i)%Indx.eq.-1) then
            EF(6)=-q*a*a*(6D0-8D0*a/l+3D0*a*a/(l*l))/12D0
            EF(12)=q*(a**3D0)*(4D0-3D0*a/l)/(12D0*l)
            EF(2)=-0.5D0*q*a*(2D0-2D0*a*a/(l*l)+a**3D0/l**3D0)
            EF(8)=-0.5D0*q*a**3D0*(2D0-a/l)/l**2D0
        end if
        if(ELoad(i)%Indx.eq.-2) then
            EF(6)=-q*a*b**2D0/l**2D0
            EF(12)=q*a*a*b/l**2D0
            EF(2)=-q*b**2D0*(1D0+2D0*a/l)/l**2D0
            EF(8)=-q*a**2D0*(1D0+2D0*b/l)/l**2D0
        end if
        if(ELoad(i)%Indx.eq.3) then
            EF(5)=q*b*(2D0-3D0*b/l)/l
            EF(11)=q*a*(2D0-3D0*a/l)/l
            EF(3)=-6D0*q*a*b/l**3D0
            EF(9)=6D0*q*a*b/l**3D0
        end if
        if(ELoad(i)%Indx.eq.-3) then
            EF(6)=q*b*(2D0-3D0*b/l)/l
            EF(12)=q*a*(2D0-3D0*a/l)/l
            EF(2)=6D0*q*a*b/l**3D0
            EF(8)=-6D0*q*a*b/l**3D0
        end if

        Elem(n)%ELForce=Elem(n)%ELForce+EF
        EF=matmul(transpose(Elem(n)%ET),EF)
        where(Elem(n)%GlbDOF>0)
            GLoad(Elem(n)%GlbDOF)=GLoad(Elem(n)%GlbDOF)-EF(:)
        end where
    end do
```

```
    i=1
   end subroutine GLoadVec

end module DispMethod

program DFrame
   use DispMethod
   implicit none
   integer(ikind):: NElem,NJoint,NGlbDOF,NJLoad,NELoad
   type(typ_Element),allocatable::Elem(:)
   type(typ_Joint),allocatable::Joint(:)
   type(typ_JointLoad),allocatable::JLoad(:)
   type(typ_ElemLoad),allocatable::ELoad(:)
   real(rkind),allocatable::Disp(:),GLoad(:)
   call Input_Data()
   call SetElemProp(Elem,Joint)
   call GLoadVec(Elem,Joint,JLoad,ELoad,GLoad)
   write(16,*) "整体荷载向量"
   write(16,*) GLoad
   write(16,*) "整体节点位移"

   call SolveDisp(Disp,Elem,Joint,GLoad)
   write(16,*) Disp
   write(16,*)
   call Output_Results()

   stop

   contains
   subroutine Input_Data()
    integer(ikind)::i
    open(5,file='data.ipt',status='old',position='rewind')
    open(16,file='data.opt',position='rewind')
    read(5,*) NElem,NJoint,NGlbDOF,NJLoad,NELoad
    allocate (Elem(NElem))
    allocate (Joint(NJoint))
    allocate (JLoad(NJLoad))
    allocate (ELoad(NELoad))
    allocate (Disp(NGlbDOF))
    allocate (GLoad(NGlbDOF))
    read(5,*) (Joint(i),i=1,NJoint)
    read(5,*) (Elem(i)%JointNo,Elem(i)%EA,Elem(i)%EIy,&
                   Elem(i)%EIz,Elem(i)%GIp,Elem(i)%A,i=1,NElem)
    if(NJLoad>0) read(5,*) (JLoad(i),i=1,NJLoad)
```

```
 if(NELoad>0) read(5,*) (ELoad(i),i=1,NELoad)
end subroutine Input_Data

subroutine Output_Results()
 integer(ikind)::i
 real(rkind) :: EDisp(12)
 do i=1,size(Elem)
       EDisp=zero
       where(Elem(i)%GlbDOF>0)
             EDisp(:)=Disp(Elem(i)%GlbDOF)
       end where
       write(16,*) "单元    ",itoc(i),"位移"
       write(16,*) EDisp
       EDisp=matmul(Elem(i)%ET,EDisp)
       Elem(i)%EForce=matmul(Elem(i)%EK,EDisp)+Elem(i)%ELForce
       write(16,*) "单元    ",itoc(i),"内力"
       write(16,*) Elem(i)%EForce

 end do

end subroutine Output_Results
end program DFrame
```

程序考题：

1：自选考题

如图所示空间刚架，各杆材料，几何参数相同已知 EA=5.4E6kN，GIp=6.08E4kNm^2，EIy=1.62E5kNm^2，EIz=3.75E4 kNm^2 。（题目选自《结构矩阵分析和程序设计》，匡文起）

输入文件：

```
3,4,6,2,6
0,0,5,1,2,3,4,5,6
0,5,5,0,0,0,0,0,0
5,0,5,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0
1, 2, 5.4e6, 1.62e5, 3.75e4, 6.08e4,0
1, 3, 5.4e6, 1.62e5, 3.75e4, 6.08e4,0
4, 1, 5.4e6, 1.62e5, 3.75e4, 6.08e4,0
1,4,15
1,5,20
1, 1,     5, -14
1,-2, 2.5, -16
2, 2, 2.5, -17
2,-3, 2.5, 9
3, 2, 2.5, -12
3,-2, 2.5, 12
```

输出文件：

整体荷载向量

14.0000000000000	3.30000000000000	-43.5000000000000
-6.66666666666667	23.1250000000000	-12.2500000000000

整体节点位移

1.426524186386400E-005	4.688151946271983E-006	-3.518505521174383E-005
-3.109492948038260E-005	8.222154207338749E-005	-1.685671589626063E-004

单元 1 位移

1.426524186386400E-005	4.688151946271983E-006	-3.518505521174383E-005
-3.109492948038260E-005	8.222154207338749E-005	-1.685671589626063E-004
0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000
0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000

单元 1 内力

5.06320410197374	6.43154069862663	33.2438311631497
0.999813951612392	-23.7687688593765	4.81459805434703
-5.06320410197374	9.56845930137337	36.7561688368503
-0.999813951612392	32.5496130436281	-12.6568945612139

单元 2 位移

1.426524186386400E-005	4.688151946271983E-006	-3.518505521174383E-005
-3.109492948038260E-005	8.222154207338749E-005	-1.685671589626063E-004
0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000
0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000

单元 2 内力

15.4064612129731	1.19977291634312	4.75602846553365
-0.378114342481452	1.39890679934362	-2.76482140136174
-15.4064612129731	-1.19977291634312	12.2439715344663
0.378114342481452	17.3209508729881	-0.236314016922647

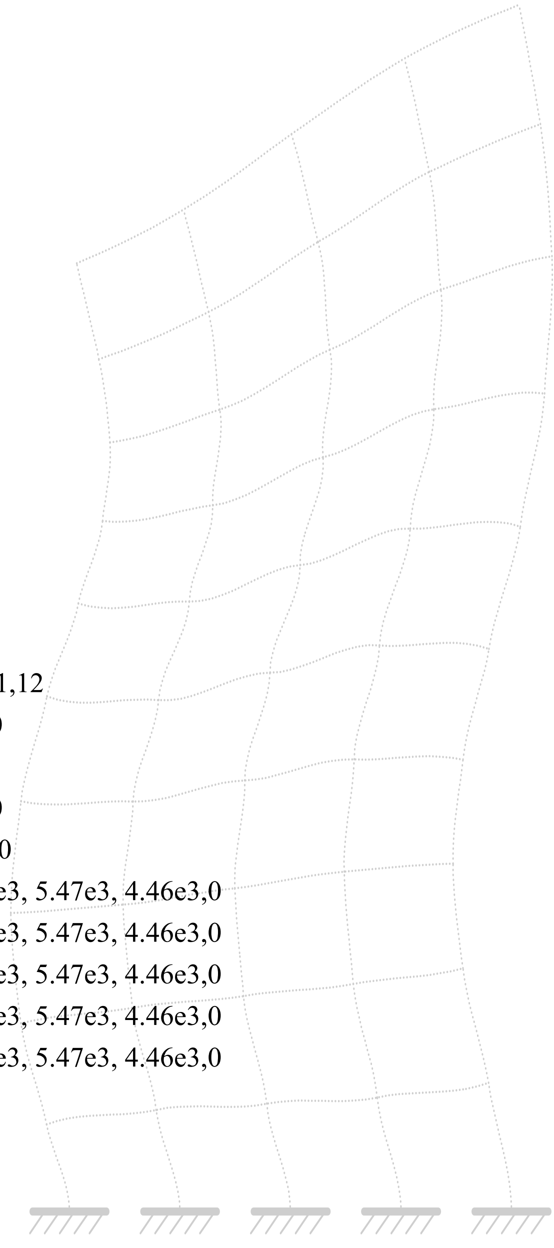
单元 3 位移

0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000
0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000
1.426524186386400E-005	4.688151946271983E-006	-3.518505521174383E-005
-3.109492948038260E-005	8.222154207338749E-005	-1.685671589626063E-004

单元 3 内力

37.9998596286833	-5.73702298168314	3.02507948565351
2.04977665298529	-2.72667667731152	-7.07576942531071
-37.9998596286833	-6.26297701831686	8.97492051434649
-2.04977665298529	17.6012792490440	8.39065451689503

运行结果正确。

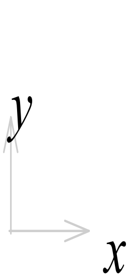


2：规定考题：

题目见附后。

输入文件：

5,6,12,6,0
0,0,2,1,2,3,4,5,6
2.5,0,2,7,8,9,10,11,12
0,1.5,2,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0
4,1.5,0,0,0,0,0,0,0
4,-1.5,0,0,0,0,0,0,0
1, 2, 3.39e2, 5.47e3, 5.47e3, 4.46e3,0
1, 3, 3.39e2, 5.47e3, 5.47e3, 4.46e3,0
1, 4, 3.39e2, 5.47e3, 5.47e3, 4.46e3,0
2, 5, 3.39e2, 5.47e3, 5.47e3, 4.46e3,0
2, 6, 3.39e2, 5.47e3, 5.47e3, 4.46e3,0
1,4,7.9e3
1,5,8.1e3
1,6,8.3e3
2,1,-7.1e3
2,2,-7.3e3
2,3,-7.5e3



输出文件：

整体荷载向量

0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000
7900.00000000000	8100.00000000000	8300.00000000000
-7100.00000000000	-7300.00000000000	-7500.00000000000

0.000000000000000E+000 0.000000000000000E+000 0.000000000000000E+000

整体节点位移

-0.309630428262080 0.263278832839125 -0.760790620509566

0.822951752899139 0.771280204272966 -0.134994574555115

-2.26398531672088 -1.37476947766886 -1.34992089009757

0.738654146991511 -0.856298991704140 -0.312286073228411

单元 1 位移

-0.309630428262080 0.263278832839125 -0.760790620509566

0.822951752899139 0.771280204272966 -0.134994574555115

-2.26398531672088 -1.37476947766886 -1.34992089009757

0.738654146991511 -0.856298991704140 -0.312286073228411

单元 1 内力

265.010522875014 4532.61529287078 2921.36335388699

150.386928939208 -90.5609115608313 6053.68291518564

-265.010522875014 -4532.61529287078 -2921.36335388699

-150.386928939208 -7212.84747315664 5277.85531699130

单元 2 位移

-0.309630428262080 0.263278832839125 -0.760790620509566

0.822951752899139 0.771280204272966 -0.134994574555115

0.000000000000000E+000 0.000000000000000E+000 0.000000000000000E+000

0.000000000000000E+000 0.000000000000000E+000 0.000000000000000E+000

单元 2 内力

59.5010162216422 4052.84693504433 -2792.40934371062

2293.27314070495 -906.723717789231 2547.35498607226

-59.5010162216422 -4052.84693504433 2792.40934371062

-2293.27314070495 5095.33773335516 3531.91541649423

单元 3 位移

-0.309630428262080 0.263278832839125 -0.760790620509566

0.822951752899139 0.771280204272966 -0.134994574555115

0.000000000000000E+000 0.000000000000000E+000 0.000000000000000E+000

0.000000000000000E+000 0.000000000000000E+000 0.000000000000000E+000

单元 3 内力

-128.954010176371 -4592.11630909242 -3787.83641216932

-301.037901257905 5897.28777085588 -6842.88935327156

128.954010176371 4592.11630909242 3787.83641216932

301.037901257905 1678.38505348276 -2341.34326491327

单元 4 位移

-2.26398531672088 -1.37476947766886 -1.34992089009757

0.738654146991511 -0.856298991704140 -0.312286073228411

0.000000000000000E+000 0.000000000000000E+000 0.000000000000000E+000

0.000000000000000E+000 0.000000000000000E+000 0.000000000000000E+000

单元 4 内力

-110.007111701298 567.768288251598 -2922.16720868377

235.123266125875	2143.77783413962	294.278508063015
110.007111701298	-567.768288251598	2922.16720868377
-235.123266125875	6375.73037712511	1361.03628004385

单元 5 位移

-2.26398531672088	-1.37476947766886	-1.34992089009757
0.738654146991511	-0.856298991704140	-0.312286073228411
0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000
0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000

单元 5 内力

54.4800128621408	-4705.27180303564	-3422.91841323718
1583.03862012604	4833.64228117416	-5833.81735887245
-54.4800128621408	4705.27180303564	3422.91841323718
-1583.03862012604	5145.79402260895	-7884.28940896397

内力图：