```
!       Last change:  CYCO    28 Dec 104      2:49 pm
!modules are highlighted by ***, Subroutines by ===
!*****************************
module NumKind
!*****************************
!This module defines the kind of integer and real number
!Every module, subroutine and function must use this module.
    implicit none
    INTEGER(KIND(1)),PARAMETER::ikind=KIND(1),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,Night=9.D0,Ten=10.D0,Twelve=12.D0
    !Integer from 0-10 and 12 relating to any calculation should be written as above,
    !for the convenience of changing the precision of the analysis.
end module NumKind

!*****************************
module TypeDef
!*****************************
    USE NumKind
    implicit none
    !Every joint should be stored as the following format:
    TYPE::typ_Joint
    REAL(rkind)       :: x,y         !The two coordicates of a joint
    INTEGER(ikind)   :: GDOF(3)   !The three Global DOF of a joint
    END TYPE typ_Joint

  !Every Element should be stored as the following format:
  TYPE::typ_Element
       !The index number of the starting and the end joint of an element
       INTEGER(ikind) ::JointNo(2)
       !The Element Location Vector
       INTEGER(ikind)::GlbDOF(6)
       !The Element properties
       REAL(rkind)      ::Length,CosA,SinA,EI,EA
  END TYPE typ_Element

    !Every Joint Load should be stored as the following format:
    TYPE::typ_JointLoad
       !Here load can be force, moment, or displacement.
       INTEGER(ikind)   :: JointNo !The no. of the Joint where the Load is applied
       INTEGER(ikind)   :: LodDOF   !The direction of the load
       REAL(rkind)       :: LodVal   !The magnitude of the load
    END TYPE typ_JointLoad

    !Every Element Load should be stored as the following format:
```

```
    TYPE::typ_Elemload
        !Here load can be force, pressure, moment or temperature changes
        INTEGER(ikind)   :: ElemNO       !the no. of the element where the Load is applied
        INTEGER(ikind)   :: Indx         !The type index of the Load
        REAL(rkind)      :: Pos,LodVal !The position and magnitude of the load
        !Here the Pos refers to the relative distance of the length of the element.
    END TYPE typ_Elemload
contains
    !=======================
    subroutine SetElemProp(Elem,Joint)
    !=======================
    !Calculate the basic properties of an element according to the
    !joints' information of the element
        TYPE(typ_Element),INTENT(in out) ::Elem(:)
        TYPE(typ_Joint),INTENT(in)       ::Joint(:)
        INTEGER(ikind)                    ::ie,Nelem
        REAL(rkind)                       ::x1,x2,y1,y2
        Nelem=SIZE(Elem,1)
        do ie=1,Nelem
          x1 = Joint(Elem(ie)%JointNo(1))%x
          x2 = Joint(Elem(ie)%JointNo(2))%x
          y1 = Joint(Elem(ie)%JointNo(1))%y
          y2 = Joint(Elem(ie)%JointNo(2))%y
          Elem(ie)%Length      = sqrt((x2-x1)**Two+(y2-y1)**Two)
          Elem(ie)%CosA        = (x2-x1)/Elem(ie)%Length
          Elem(ie)%SinA        = (y2-y1)/Elem(ie)%Length
          Elem(ie)%GlbDOF(1:3) = Joint(Elem(ie)%JointNo(1))%GDOF(1:3)
          Elem(ie)%GlbDOF(4:6) = Joint(Elem(ie)%JointNo(2))%GDOF(1:3)
        end do
       return
    end subroutine   SetElemProp
    !=======================
    subroutine TransMatrix(ET,CosA,SinA)
    !=======================
    !Creat the transform matric of the element ET
        REAL(rkind),INTENT(out):: ET(:,:)
        REAL(rkind),INTENT(IN) :: CosA,SinA
    ! ET could be 2x2, 3x3 or 6x6 depending on size(ET)
        ET=Zero
        ET(1,1:2)=(/CosA,SinA/)
        ET(2,1:2)=(/-SinA,CosA/)
        IF(SIZE(ET,1)>2) ET(3,3)=One
        IF(SIZE(ET,1)>3) ET(4:6,4:6)=ET(1:3,1:3)
        return
    end subroutine TransMatrix
end module TypeDef
```

```
!************************
module BandMat
!************************
    USE NumKind
    USE TypeDef,ONLY:typ_Element
    implicit none

    private
    PUBLIC::SetMatBand,DelMatBand,VarBandSolv

    TYPE,PUBLIC::typ_Kcol
        REAL(rkind),POINTER::row(:)
    END TYPE typ_Kcol

    contains

    !========================
    subroutine SetMatBand(Kcol,Elem)
    !========================
        TYPE(typ_Kcol),INTENT(in out):: Kcol(:)
        TYPE(typ_Element), INTENT(in):: Elem(:)
        INTEGER(ikind)                :: minDOF,ELocVec(6)
        INTEGER(ikind)                :: ie,j,NGlbDOF,NElem
        INTEGER(ikind)                :: row1(SIZE(Kcol,1))

        NGlbDOF = SIZE(Kcol,1)
        NElem   = SIZE(Elem,1)
        row1    = NGlbDOF

        !确定各列始行码，放在数组 row1(:)中
        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

        !为各列的半带宽分配空间并初始化
        do j=1,NGlbDOF
            ALLOCATE(Kcol(j)%row(row1(j):j))
            Kcol(j)%row=Zero !清零
        end do
        return
    end subroutine SetMatBand
```

```
!=======================
subroutine DelMatBand(Kcol)
!=======================
    TYPE(typ_Kcol),INTENT(in out):: Kcol(:)
    INTEGER(ikind)                :: j,NGlbDOF
    NGlbDOF = SIZE(Kcol,1)
    do j=1,NGlbDOF
        deallocate (Kcol(j)%row)
    end do
    return
end subroutine DelMatBand


!=======================
subroutine VarBandSolv(Disp,Kcol,GLoad)
!=======================
    type (typ_KCol),      intent(in out)   :: Kcol(:)
    real (rkind),         intent(out)      :: Disp(:)
    real (rkind),         intent(in)       :: GLoad(:)
    integer (ikind)                        :: i,j,row1j,row_1,NCol
    real(rkind)                            :: s,Diag(size(Kcol,dim=1))

    NCol=size(Kcol,1)
    Diag(:)=(/(Kcol(j)%row(j),j=1,NCol)/)
    do j=2,NCol
        row1j=lbound(Kcol(j)%row,1)
        do i=row1j,j-1
            row_1=max(row1j,lbound(Kcol(i)%row,1))
            s=sum(Diag(row_1:i-1)*Kcol(i)%row(row_1:i-1)*Kcol(j)%row(row_1:i-1))
            Kcol(j)%row(i)=(Kcol(j)%row(i)-s)/Diag(i)
        end do
        s=sum(Diag(row1j:j-1)*Kcol(j)%row(row1j:j-1)**Two)
        Diag(j)=Diag(j)-s
    end do

    Disp(:) = GLoad(:)
!...[ 6-5-3 节的代码：其中 GP 换为 Disp ]
    do j=2,NCol
        row1j=lbound(Kcol(j)%row,1)
        Disp(j)=Disp(j)-sum(Kcol(j)%row(row1j:j-1)*Disp(row1j:j-1))
    end do
    !...[ 6-5-4 节的代码：其中 GP 换为 Disp ]
    Disp(:)=Disp(:)/Diag(:)
    do j=NCol,1,-1
        row1j=lbound(Kcol(j)%row,1)
        Disp(row1j:j-1)=Disp(row1j:j-1)-Disp(j)*Kcol(j)%row(row1j:j-1)
    end do
```

```
    return
    end subroutine VarBandSolv
end module BandMat


!****************************
module DispMethod
!****************************
!The displacement method in structure machemism
    USE NumKind
    USE TypeDef
    USE BandMat
    implicit none

    CONTAINS
    !========================
    subroutine SolveDisp(Disp,Elem,Joint,JLoad,ELoad)
    !========================
        REAL(rkind),INTENT(OUT)          ::Disp(:)
        TYPE(typ_Element),INTENT(in)   ::Elem(:)
        TYPE(typ_Joint)   ,INTENT(in)   ::Joint(:)
        TYPE(typ_JointLoad),INTENT(in)::JLoad(:)
        TYPE(typ_Elemload),INTENT(in) ::ELoad(:)

        TYPE(typ_Kcol),allocatable      ::Kcol(:)
        REAL(rkind),allocatable          ::GLoad(:)
        INTEGER(ikind)                    ::NGlbDOF
        NGlbDOF=size(Disp,dim=1)
        allocate(GLoad(NGlbDOF))
        allocate(Kcol(NGlbDOF))

        call SetMatBand(Kcol,Elem)
        call GLoadVec(GLoad,Elem,JLoad,ELoad,Joint)
        call GStifMat(Kcol,Elem)
        call VarBandSolv(Disp,Kcol,GLoad)
        call DelMatBand(Kcol)!free the memory space occupied by Kcol
        deallocate(GLoad)      !free the memory space occupied by GLoad
        return
    end subroutine SolveDisp
    !========================
    subroutine GStifMat(Kcol,Elem)
    !========================
    !Creat the Globle Stiffness Matrix for the problem
        TYPE(typ_Kcol),INTENT(in out)    :: Kcol(:)
        TYPE(typ_Element),INTENT(in)     :: Elem(:)
        integer (ikind)                  :: ie,j,JGDOF,NElem
        real (rkind)                     :: EK(6,6),ET(6,6)
```

```
    integer (ikind)                    :: ELocVec(6)
    NElem=size(Elem,1)

    do ie=1,NElem
     !Create the Element Stiffness Matrix
       call EStifMat(EK,Elem(ie)%Length,Elem(ie)%EI,Elem(ie)%EA)
     !Create the Transform Matrix
       call TransMatrix(ET,Elem(ie)%CosA,Elem(ie)%SinA)
     !Transform the Element stiffness Matrix to Globle coordinate system
       EK=matmul(transpose(ET),matmul(EK,ET))
     !Get the Element Location Vector
       ELocVec=Elem(ie)%GlbDOF
     !Integrate the element stiffness matrix to the Global Stiffness matrix
     !according to their Element Location Vectors.
    do j=1,6
          JGDOF=ELocVec(j)
          if(JGDOF==0) cycle
          where (ELocVec<=JGDOF.and.ELocVec>0)
              Kcol(JGDOF)%row(ELocVec)=Kcol(JGDOF)%row(ELocVec)+EK(:,j)
          end where
       end do
    end do
    return
end subroutine   GStifMat
!========================
SUBROUTINE GLoadVec(GLoad,Elem,JLoad,ELoad,Joint)
!========================
    REAL(rkind),            INTENT(OUT):: GLoad(:)
    TYPE(typ_Element),   INTENT(in) :: Elem(:)
    TYPE(typ_JointLoad),INTENT(in) :: JLoad(:)
    TYPE(typ_Elemload), INTENT(in) :: ELoad(:)
    TYPE(typ_Joint)     , INTENT(in) :: Joint(:)

  INTEGER(ikind) :: i             !temporary Loop variable
  INTEGER(ikind) :: NJLoad      !Number of Joint Load
  INTEGER(ikind) :: NELoad      !Number of Element Load
  INTEGER(ikind) :: ELocVec(6),ve(6),GDOF(3),vj,LodDOF
  REAL(rkind)      :: F0(6)       !Fix End Forces
  REAL(rkind)      :: ET(6,6)
  INTEGER(ikind) :: ElemNo,JointNo
   NJLoad = SIZE(JLoad,1)
   NELoad = SIZE(ELoad,1)
   GLoad   = Zero                  !Initialize
   ve(:)   = (/1,2,3,4,5,6/) !An assistant vector
   !Deal with every element load
   do i=1,NELoad
      ElemNo=ELoad(i)%ElemNo
```

```
            ELocVec(:)=Elem(ElemNo)%GlbDOF(:) !Get the Element Location Vector
            F0=Zero     !Initialize
          !Calculate the element fixend forces vector F0(6) under the local coordinate system
            CALL EFixendF(F0,ELoad(i)%Indx,ELoad(i)%Pos,ELoad(i)%LodVal,Elem(ElemNo))
          !Create the Transform Matrix
            CALL TransMatrix(ET,Elem(ElemNo)%CosA,Elem(ElemNo)%SinA)
          !Transform the fixend forces vector to Globle coordinate system
            F0=matmul(transpose(ET),F0)
          !Integrate F0 to the Global Load Vector
            where (ELocVec>0)
                GLoad(ELocVec)=GLoad(ELocVec)-F0(ve)
            end where
        end do

       !Deal with every joint load
        do i=1,NJLoad
            JointNo = JLoad(i)%JointNo          !The joint where the load is applied
            GDOF(:) = Joint(JointNo)%GDOF(:) !The Global DOF of the joint
            LodDOF    = JLoad(i)%LodDOF             !The direction of the load (local coordinate
system)
            vj=GDOF(LodDOF)                          !The direction of the load (global coordinate
system)
          !Integrate JLoad to the Global Load Vector
            GLoad(vj)=GLoad(vj)+JLoad(i)%LodVal
        end do
        return
    END SUBROUTINE     GLoadVec
    !=======================
    SUBROUTINE EStifMat(EK,Elen,EI,EA)
    !=======================
    !Create the element Stiffness Matrix EK
        REAL(rkind),INTENT(out)      ::EK(6,6)
        REAL(rkind),INTENT(in)       ::Elen,EI,EA
        INTEGER(ikind)               ::i,j

        EK=Zero !Initialize
    !Generate the upper triangle part of EK
        EK(1,1) = EA/Elen
        EK(1,4) = -EA/Elen
        EK(2,2) = Twelve*EI/Elen**Three
        EK(2,3) = Six*EI/Elen**Two
        EK(2,5) = -Twelve*EI/Elen**Three
        EK(2,6) = Six*EI/Elen**Two
        EK(3,3) = Four*EI/Elen
        EK(3,5) = -Six*EI/Elen**Two
        EK(3,6) = Two*EI/Elen
        EK(4,4) = EA/Elen
```

```
    Ek(5,5) = Twelve*EI/Elen**Three
    EK(5,6) = -Six*EI/Elen**Two
    EK(6,6) = Four*EI/Elen
    !Mirror the upper triangle part to get the complete matrix
    do j=1,6
        do i=j+1,6
            EK(i,j)=EK(j,i)
        end do
    end do
    return
END SUBROUTINE   EStifMat
!=======================
subroutine EFixendF(F0,Indx,a,q,Elem)
!=======================
!Calculate the fixend forces vector F0 of an element
    REAL(rkind)     ,     INTENT(out)::F0(6)
    INTEGER(ikind),     INTENT(in) ::Indx
    REAL(rkind)     ,     INTENT(in) ::a        !the relative distance
    REAL(rkind)     ,     INTENT(in) ::q        !the magnitude of the load
    TYPE(typ_Element),INTENT(in) ::Elem
    real                                ::l
    l=Elem%length
    select case (Indx)
        case (1) !Transverse uniform pressure
            F0(1) = Zero
            F0(2) = -q*(a*l)/Two*(Two-Two*a**Two+a**Three)
            F0(3) = -q*(a*l)**Two/Twelve*(Six-Eight*a+Three*a**Two)
            F0(4) = Zero
            F0(5) = -q*(a*l)*a**Two/Two*(Two-a)
            F0(6) = q*(a*l)**Two*a/Twelve*(Four-Three*a)
        case (2) !Transverse concentrated force
            F0(1) = Zero
            F0(2) = -q*(One-Two*a+a**Two)*(One+Two*a)
            F0(3) = -q*(a*l)*(One-Two*a+a**Two)
            F0(4) = Zero
            F0(5) = -q*a**Two*(Three-Two*a)
            F0(6) = q*a**Two*(One-a)*l
        case (3) !axial displacement of the supports
        if(a<One/Two)then
          F0(1)=Elem%EA*q/l
          F0(4)=-F0(1)
        else
          F0(1)=-Elem%EA*q/l
          F0(4)=-F0(1)
        end if
          F0(2) = Zero
          F0(3) = Zero
```

```
            F0(5) = Zero
            F0(6) = Zero
        case (4) !transvers displacement of the supports
            if(a<One/Two)then
              F0(2) = Twelve*Elem%EI*q/l**Three
              F0(3) = Six*Elem%EI*q/l**Two
              F0(5) = -F0(2)
              F0(6) = F0(3)
            else
              F0(2) = -Twelve*Elem%EI*q/l**Three
              F0(3) = -Six*Elem%EI*q/l**Two
              F0(5) = -F0(2)
              F0(6) = F0(3)
            end if
            F0(1) = Zero
            F0(4) = Zero
    end select
    return
end subroutine EFixendF
!=======================
subroutine ElemDisp(EDisp,ie,Disp,Elem)
!=======================
!Get the Element Displacement from the global result Disp()
!in the Global Coordinate System
    REAL(rkind)         ,intent(out) :: EDisp(:)   !Element displacement to be generated
    REAL(rkind)         ,intent(in)  :: Disp(:)    !Global displacement solve
    INTEGER(ikind)      ,intent(in)  :: ie         !Index of the element to be dealt with
    TYPE(typ_Element),intent(in)  :: Elem(:)    !All the Elements
    INTEGER(ikind)                   :: i          !loop variable
    do i=1,6
        if(Elem(ie)%GlbDOF(i)==0)then
            EDisp(i)=Zero
        else
            EDisp(i)=Disp(Elem(ie)%GlbDOF(i))
        end if
    end do
    return
end subroutine ElemDisp
!=======================
subroutine ElemForce(EForce,ie,Disp,Elem,ELoad)
!=======================
!Get the Element end forces from the global result
!in the Local Coordinate System
!Here we have to repeat the function contained in the subroutine ElemDisp
!in consideration of independence between the calculations of displacement and forces
    REAL(rkind)         ,intent(out):: EForce(:)
    REAL(rkind)         ,intent(in) :: Disp(:)
```

```
        INTEGER(ikind)      ,intent(in) :: ie
        TYPE(typ_Element) ,intent(in) :: Elem(:)
        TYPE(typ_ElemLoad),intent(in) :: Eload(:)
        REAL(rkind)                   :: EDisp(6)
        REAL(rkind)                   :: ET(6,6),EK(6,6)
        REAL(rkind)                   :: EP(6)              !Fix end Forces
        INTEGER(ikind)                :: NELoad             !Number of Element Load
        INTEGER(ikind)                :: i                  !loop variable
        INTEGER(ikind)                :: ElemNo             !Where the element load is applied
        !Creat a transform matrix
        CALL TransMatrix(ET,ELem(ie)%CosA,Elem(ie)%SinA)
        !Get the element displacement
        do i=1,6
           if(Elem(ie)%GlbDOF(i)==0)then
              EDisp(i)=Zero
           else
              EDisp(i)=Disp(Elem(ie)%GlbDOF(i))
           end if
        end do
          !Creat the element stiffness matrix
        CALL EStifMat(EK,Elem(ie)%Length,Elem(ie)%EI,Elem(ie)%EA)
          !Calculate the element end forces caused by displacement
        EForce=matmul(EK,matmul(ET,EDisp))
          !Calculate the element end forces caused by element load
          !Which is the same as end fix forces
          NELoad = SIZE(ELoad,1)
          EP       = Zero                !Initialize
          do i=1,NELoad
           if(ELoad(i)%ElemNo==ie)then
             ElemNo=ELoad(i)%ElemNo
             !Calculate the element fixend forces vector F0(6) under the local coordinate system
             CALL EFixendF(EP,ELoad(i)%Indx,ELoad(i)%Pos,ELoad(i)%LodVal,Elem(ElemNo))
             !Add the two part to get the whole element end forces
             EForce(:)    = EForce(:)+EP(:)
           end if
          end do
          return
    end subroutine ElemForce
end module DispMethod

!*****************************
program SM_90          !main program
!*****************************
    USE NumKind
    USE TypeDef
    USE DispMethod
    implicit none
```

```
    TYPE(typ_Element)   ,ALLOCATABLE :: Elem(:)
    Type(typ_Joint)      ,AllOCATABLE :: Joint(:)
    TYPE(typ_JointLoad),ALLOCATABLE :: JLoad(:)
    TYPE(typ_Elemload) ,ALLOCATABLE :: ELoad(:)
    REAL(rkind)            ,ALLOCATABLE :: Disp(:)
    INTEGER(ikind)                      :: ProbType
    INTEGER(ikind)                      :: NElem,NJoint,NGlbDOF,NJLoad,NELoad
    INTEGER(ikind)                      :: i,ie
    call InputData()          !See below
    if(ProbType==1)then
        call SetElemProp(Elem,Joint)
        call SolveDisp(Disp,Elem,Joint,JLoad,Eload)
    end if
    call OutputResutls()    !See below

    stop

    contains
    !=======================
    subroutine InputData()
    !=======================
      OPEN(5,FILE="SM90.IPT",ACTION="READ",POSITION="REWIND",STATUS="OLD")
      READ(5,*) ProbType
      READ(5,*) NElem,NJoint,NGlbDOF,NJLoad,NELoad
      ALLOCATE(Elem(NElem))
      ALLOCATE(Joint(NJoint))
      ALLOCATE(JLoad(NJLoad))
      ALLOCATE(ELoad(NELoad))
      ALLOCATE(Disp(NGlbDOF))
      Disp=Zero
      READ(5,*) (Joint(i),i=1,NJoint)
      READ(5,*) (Elem(ie)%JointNo,Elem(ie)%EA,Elem(ie)%EI,ie=1,NElem)
      if(NJLoad>0)READ(5,*)(JLoad(i),i=1,NJLoad)
      if(NELoad>0)READ(5,*)(ELoad(i),i=1,NELoad)

      return
    end subroutine InputData
    !=======================
    subroutine OutputResutls()
    !=======================
      REAL(rkind)       :: EDisp(6),EForce(6),EDispLoad(6)
      REAL(rkind)       :: ET(6,6)
      INTEGER(ikind)   :: ie

OPEN(8,FILE="SMCAI90.OUT",ACTION="WRITE",POSITION="REWIND",STATUS="REPLACE")
      WRITE(8,*) "10 0"
```

```
      do ie=1,size(Elem)
       CALL ElemDisp(EDisp,ie,Disp,Elem)
       EDispLoad=Zero !Displacement Load Initialize
       !Create the Transform Matrix
           CALL TransMatrix(ET,Elem(ie)%CosA,Elem(ie)%SinA)
       !get the EDispLoad(6) under the local coordinate system
       do i=1,NELoad
           if(ELoad(i)%ElemNo==ie)then
               if(ELoad(i)%Indx==3)then
                   if(ELoad(i)%Pos==0)then
                       EDispLoad(1)=EDispLoad(1)+ELoad(i)%LodVal
                   else
                       EDispLoad(4)=EDispLoad(4)+ELoad(i)%LodVal
                   end if
               else if(ELoad(i)%Indx==4)then
                   if(ELoad(i)%Pos==0)then
                       EDispLoad(2)=EDispLoad(2)+ELoad(i)%LodVal
                   else
                       EDispLoad(5)=EDispLoad(5)+ELoad(i)%LodVal
                   end if
               end if
           end if
       end do
       !Transform the EDispLoad(6) to global coordinate system
       EDispLoad=matmul(transpose(ET),EDispLoad)
       !Add the two part of displacement together
       EDisp=EDisp+EDispLoad
       WRITE(8,*) EDisp
      end do
      do ie=1,size(Elem)
       CALL ElemForce(EForce,ie,Disp,Elem,ELoad)
       WRITE(8,*) -EForce(1),EForce(2:6)
      end do
      deallocate(JLoad)
      deallocate(Joint)
      deallocate(Elem)
      deallocate(ELoad)
      deallocate(Disp)
      return
     end subroutine OutputResutls
end program   SM_90
```

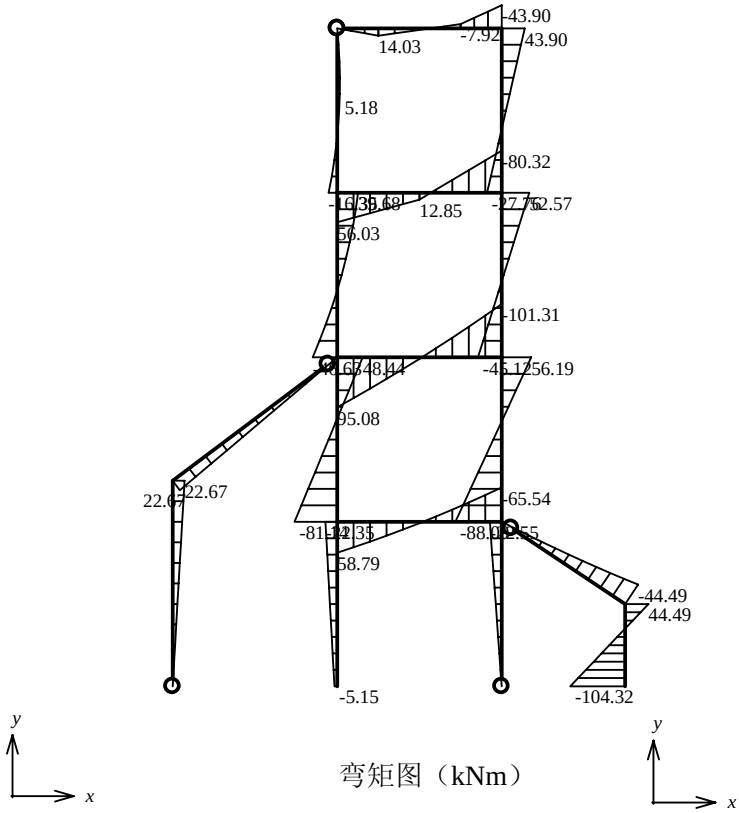
```
TITLE,平面刚架大作业
变量定义,l1=4,l2=4,l3=3,h1=5,h2=4,h3=2,P1=15,P2=10,P3=25,q1=6,q2=3,q3=2
结点,1,0,0
结点,3,L1,0
结点,8,L1+L2,0
结点,13,L1+L2+L3,0
结点生成,4,1,3,8,5,0,H2
结点,2,0,H1
结点,14,L1+l2+l3,H3
单元,3,4,1,1,1,1,1,1
单元,4,9,1,1,1,1,1,1
单元,8,9,1,1,1,1,1,1
单元生成,2,1,3,1
单元,6,7,1,1,1,1,1,0
单元,7,12,1,1,0,1,1,1
单元,11,12,1,1,1,1,1,1
单元,1,2,1,1,0,1,1,1
单元,2,5,1,1,1,1,1,0
单元,9,14,1,1,0,1,1,1
单元,13,14,1,1,1,1,1,1
结点支承,1,4,0,0,0
结点支承,3,6,0,0,-0.01,0
结点支承,8,4,0,0,0
结点支承,13,6,0,0,0,0
结点荷载,2,1,p1,0
结点荷载,7,1,p2,0
单元荷载,11,1,p3,1/4,90
单元荷载,11,1,p3,3/4,90
单元荷载,8,1,p3,1/2,90
单元荷载,7,3,q1,0,1,90
单元荷载,10,3,q1,0,1,90
单元荷载,5,3,q2,0,1,90
单元荷载,2,3,q2,0,1,90
单元荷载,15,3,q3,0,1,90
单元材料性质,1,1,1E5,1.5E4,0,0,-1
单元材料性质,2,2,1E6,1E4,0,0,-1
单元材料性质,3,4,1E5,1.5E4,0,0,-1
单元材料性质,6,7,1E5,1.5E4,0,0,-1
单元材料性质,9,10,1E5,1.5E4,0,0,-1
单元材料性质,12,13,1E5,1.5E4,0,0,-1
单元材料性质,16,16,1E5,1.5E4,0,0,-1
单元材料性质,5,5,1E6,1E4,0,0,-1
单元材料性质,8,8,1E6,1E4,0,0,-1
单元材料性质,11,11,1E6,1E4,0,0,-1
单元材料性质,14,15,1E6,1E4,0,0,-1
END
```

```
1
16 17 35 2 9
0.000000000000000E+00 0.000000000000000E+00 0 0 27
4.000000000000000 0.000000000000000E+00 0 0 0
8.000000000000000 0.000000000000000E+00 0 0 7
11.00000000000000 0.000000000000000E+00 0 0 0
4.000000000000000 4.000000000000000 1 2 3
8.000000000000000 4.000000000000000 4 5 6
8.000000000000000 4.000000000000000 4 5 32
4.000000000000000 8.000000000000000 8 9 10
4.000000000000000 8.000000000000000 8 9 31
8.000000000000000 8.000000000000000 11 12 13
4.000000000000000 12.00000000000000 14 15 16
8.000000000000000 12.00000000000000 17 18 19
4.000000000000000 16.00000000000000 20 21 22
4.000000000000000 16.00000000000000 20 21 23
8.000000000000000 16.00000000000000 24 25 26
0.000000000000000E+00 5.000000000000000 28 29 30
11.00000000000000 2.000000000000000 33 34 35
2 5 100000.0000000000 15000.00000000000
5 6 1000000.000000000 10000.00000000000
3 6 100000.0000000000 15000.00000000000
5 8 100000.0000000000 15000.00000000000
8 10 1000000.000000000 10000.00000000000
6 10 100000.0000000000 15000.00000000000
8 11 100000.0000000000 15000.00000000000
11 12 1000000.000000000 10000.00000000000
10 12 100000.0000000000 15000.00000000000
11 13 100000.0000000000 15000.00000000000
14 15 1000000.000000000 10000.00000000000
12 15 100000.0000000000 15000.00000000000
1 16 100000.0000000000 15000.00000000000
16 9 1000000.000000000 10000.00000000000
7 17 1000000.000000000 10000.00000000000
4 17 100000.0000000000 15000.00000000000
16 1 15.00000000000000
13 1 10.00000000000000
11 2 0.2500000000000000 -25.00000000000000
11 2 0.7500000000000000 -25.00000000000000
8 2 0.5000000000000000 -25.00000000000000
7 1 1.000000000000000 -6.000000000000000
10 1 1.000000000000000 -6.000000000000000
5 1 1.000000000000000 -3.000000000000000
2 1 1.000000000000000 -3.000000000000000
15 1 1.000000000000000 -2.000000000000000
1 3 0 -1.000000000000000E-02
```

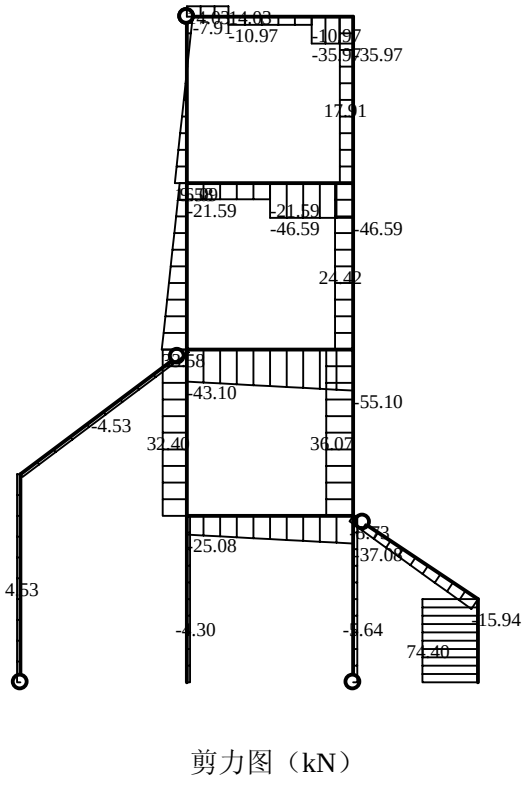
10 0

0.000000000000000E+000 -1.000000000000000E-002 0.000000000000000E+000
5.806349647923325E-003 -6.883153556935361E-003 -3.667634231706848E-003
5.806349647923325E-003 -6.883153556935361E-003 -3.667634231706848E-003
5.659564184663665E-003 -4.479038256895431E-003 -3.419020805095024E-003
0.000000000000000E+000 0.000000000000000E+000 -4.128261667013628E-004
5.659564184663665E-003 -4.479038256895431E-003 -3.419020805095024E-003
5.806349647923325E-003 -6.883153556935361E-003 -3.667634231706848E-003
4.071455763000208E-002 -4.769614016893997E-003 -8.027130799981729E-003
4.071455763000208E-002 -4.769614016893997E-003 -8.027130799981729E-003
4.066796866779876E-002 -9.985345124934248E-003 -7.672892235816372E-003
5.659564184663665E-003 -4.479038256895431E-003 -3.419020805095024E-003
4.066796866779876E-002 -9.985345124934248E-003 -7.672892235816372E-003
4.071455763000208E-002 -4.769614016893997E-003 -8.027130799981729E-003
7.808255174506967E-002 -4.467130771121871E-003 -6.820757138059562E-003
7.808255174506967E-002 -4.467130771121871E-003 -6.820757138059562E-003
7.805651581031681E-002 -1.328782837070637E-002 -6.679712677156500E-003
4.066796866779876E-002 -9.985345124934248E-003 -7.672892235816372E-003
7.805651581031681E-002 -1.328782837070637E-002 -6.679712677156500E-003
7.808255174506967E-002 -4.467130771121871E-003 -6.820757138059562E-003
0.106911674498708 -5.028165677663321E-003 -6.867209130251221E-003
0.106911674498708 -5.028165677663321E-003 -3.248222990235054E-003
0.106840022888649 -1.472679346416493E-002 -4.527524859406099E-003
7.805651581031681E-002 -1.328782837070637E-002 -6.679712677156500E-003
0.106840022888649 -1.472679346416493E-002 -4.527524859406099E-003
0.000000000000000E+000 0.000000000000000E+000 -8.715535348235721E-003
3.727963285233438E-002 -1.090408400032044E-004 -4.936709014929183E-003
3.727963285233438E-002 -1.090408400032044E-004 -4.936709014929183E-003
4.071455763000208E-002 -4.769614016893997E-003 7.315304850306170E-004
5.659564184663665E-003 -4.479038256895431E-003 3.250735254213754E-003
7.295181554056404E-003 -1.375287755997148E-003 -3.988421218420878E-003
0.000000000000000E+000 0.000000000000000E+000 0.000000000000000E+000
7.295181554056404E-003 -1.375287755997148E-003 -3.988421218420878E-003
77.9211610766160 -4.30008416856667 5.15346003176735
77.9211610766160 4.30008416856667 -22.3537967060340
-36.6963658149150 -25.0826725755819 -58.7868787176933
-36.6963658149150 37.0826725755819 -65.5438115846342
-111.975956422386 -5.63661494698812 -7.105427357601002E-015
-111.975956422386 5.63661494698812 -22.5464597879525
52.8384885010341 32.3962816463483 81.1406754237273
52.8384885010341 -32.3962816463483 48.4444511616657
-11.6472405508302 -43.0955905566674 -95.0767775237482
-11.6472405508302 55.0955905566674 -101.305584702921
-137.657671700970 36.0691267536908 88.0902713725868
-137.657671700970 -36.0691267536908 56.1862356421766
7.56208114430315 33.5781137971453 46.6323263620825

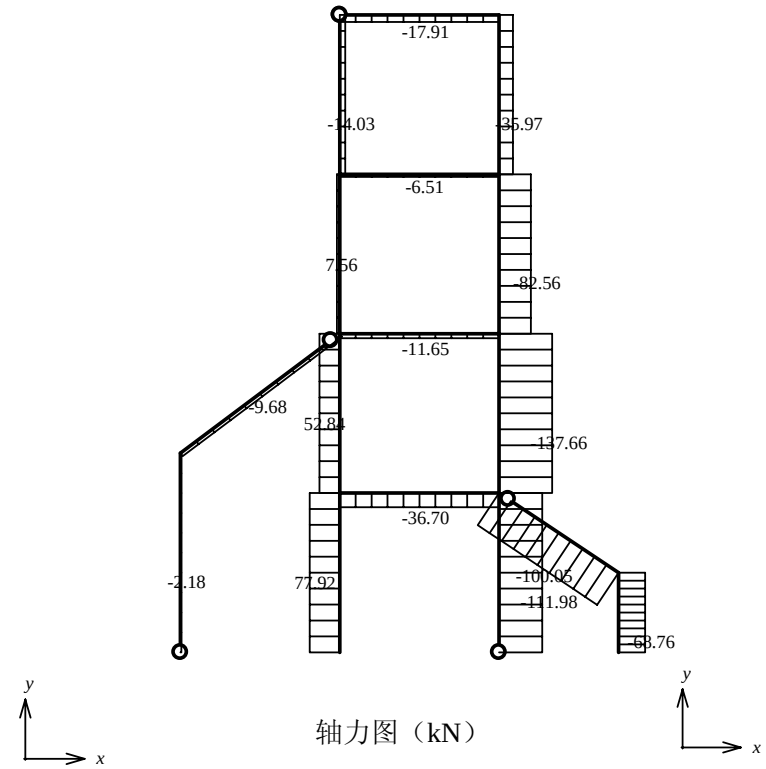
7.56208114430315	-9.57811379714533	39.6801288264987
-6.50898368821800	-21.5879538078393	-56.0285187679362
-6.50898368821800	46.5879538078393	-80.3232964634209
-82.5620811443031	24.4218862028596	45.1193490607447
-82.5620811443031	-24.4218862028596	52.5681957506937
-14.0258726635362	16.0870974853593	16.3483899414374
-14.0258726635362	7.91290251464066	-5.684341886080801E-014
-17.9129025146394	14.0258726635362	-1.065814103640150E-014
-17.9129025146394	35.9741273364638	-43.8965093458552
-35.9741273364639	17.9129025146456	27.7551007127272
-35.9741273364639	-17.9129025146456	43.8965093458552
-2.18081680006409	4.53459159996785	0.000000000000000E+000
-2.18081680006409	-4.53459159996785	22.6729579998392
-9.68081680006162	-4.53459159996783	-22.6729579998392
-9.68081680006162	4.53459159996783	2.309263891220326E-014
-100.049914893719	-8.73346296462250	3.552713678800501E-015
-100.049914893719	15.9445654502792	-44.4889484136427
-68.7643877998574	74.4021075517993	104.315266689956
-68.7643877998574	-74.4021075517993	44.4889484136427



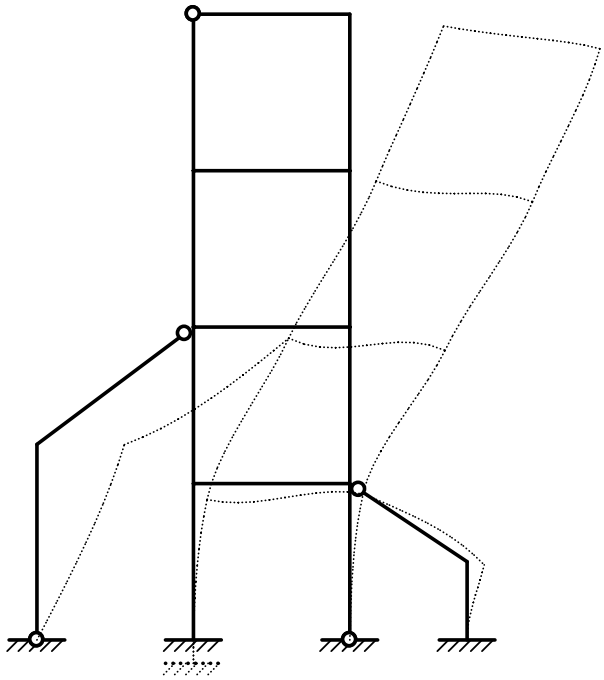
弯矩图（kNm）



剪力图（kN）



轴力图（kN）



位移图

以上结果与结构力学求解器 1.5 计算结果完全一致。