```
!###################################################################
!#      PROGRAM TO FIND NUMERICAL SOLUTION OF A BAR                #
!#      GOVERNING EQUATION:  -du/dx(EA(x)* du/dx) + C * u = f(x)   #
!#      where EA = Eo+ E1*X                                        #
!#            C  = constant                                        #
!#            f  = Fo + F1 * X + F2 * X**2                         #
!#      BOUNDARY CONDITIONS: DISPLACEMENT AT ANY ENDS OF THE BAR  #
!#                                     or                          #
!#                            APPLIED FORCE AT ANT ENDS OF THE BAR #
!#                                     or                          #
!#                            SPRING MASS AT ANT ENDS OF THE BAR   #
!#                                     or                          #
!#                           ANY COMBINATION OF ABOVE THREE B.C's  #
!#*****************************************************************#
!#         NAME          MURALI KRUSHNA RAO K. V                   #
!#         Roll No.      Y9101068                                  #
!###################################################################

        implicit real(a-h,o-z)
!       MAXIMUM NUMBER OF NODES = maxnodes
!       MAXIMUM ORDER OF SHAPE FUNCTION PLOYNOMIAL = 9
!       MAXIMUM POINTS OF GAUSS QUADRATURE RULE    = 10
        parameter(maxnodes = 1000)
        common i,j,k,l,m,n,n1,h,h1,ajacob,aincob,nel,npts,barlt,xi,nn
        common alpha1,alpha2,dampcof,maxgqpts,ngqpts,igaus1,igaus2
        common maxgaus,ibtype1,ibtype2,bdata1(5),bdata2(5)
        common f1,f2,f3,eps,alpha,maxits,sorelax,iunknowns
        dimension ea(maxnodes,5),f(maxnodes,5),weight(15),gqpt(15)
        dimension zeta(15),sh(15,15),sh1(15,15)
        dimension stiffmt(maxnodes,maxnodes),x(maxnodes),xloc(maxnodes)
        dimension elementk(15,15),forcemt(maxnodes)
        dimension finalk(maxnodes,maxnodes),finalf(maxnodes)
        dimension soln(maxnodes),slope(maxnodes),exact(maxnodes)
        dimension der(maxnodes),eder(maxnodes)
        dimension a(maxnodes,maxnodes),b(maxnodes)
        character*30 key
        common input,twicese
        input =5
c
        read(input,*) key
        write(6,*)key
102     read(input,*) key
        if (key.eq.'done') then
          goto 101
        else if (key.eq.'order') then
          read(input,*)npts
        else if(key.eq.'numel') then
          read(input,*) nel
        else if(key.eq.'barlength') then
          read(input,*) barlt
        else if(key.eq.'initialPoint') then
          read(input,*) xi
        else if(key.eq.'EAconst1') then
          read(input,*) alpha1
        else if(key.eq.'EAconst2') then
         read(input,*) alpha2
        else if(key.eq.'Fconst1') then
         read(input,*) f1
```

```
        else if(key.eq.'Fconst2') then
         read(input,*) f2
        else if(key.eq.'Fconst3') then
         read(input,*) f3
        else if(key.eq.'dampCoeff') then
         read(input,*) dampcof
        else if (key.eq.'RelaxFact') then
         read(input,*) sorelax
        else if (key.eq.'epsilon') then
         read(input,*) eps
        else if(key.eq.'maxiter') then
         read(input,*) maxits
         end if
        go to 102
101     continue

        open(unit=9,file='bc.inp',status='old')
        read(9,*) ibtype1,bdata1(1), bdata1(2)
        read(9,*) ibtype2,bdata2(1), bdata2(2)
        close(9)
c
        igaus1 = npts
!       calculating (Na1, Na2) term's ploynomial order
        if(npts.gt.1.or.dampcof.ne.0) igaus2 = (npts*2)-2
!       evaluating the (Na,f) term's ploynomial order
        if(f2.ne.0) igaus1 = npts+1
        if(f3.ne.0) igaus1 = npts+2
!       calculating Max order of ploynomial for Max number of
!       gauss quadrature points
        if(igaus1.ge.igaus2) then
        maxgaus = igaus1
        else
        maxgaus = igaus2
        end if
        write(6,*)" max order of the polymonial",maxgaus
c
        ngqpts = (maxgaus+1)/2
!       roundoff the max gauss quadrature points for
!       maximum  odd order of the polynomial( e.g 2.5 = 2)
        if(mod(maxgaus,2) .eq.0) ngqpts = 1 + (maxgaus+1)/2
c
        if(ibtype1.eq.2.and.ibtype2.eq.2) then
        write(6,*)"-------------------------------------------------"
        write(6,*)" THE DETERMINANT OF THE STIFFNESS MATRIX IS ZERO "
        write(6,*)" FIX ONE END OF THE BAR TO GET SOLUTION "
        write(6,*) "i.e set ibtype =1 or ibtype2 = 1 "
        write(6,*)"-------------------------------------------------"
        goto 90
        end if

        if(ngqpts.ge.10) then
        write(6,*)"-------------------------------------------------"
        write(6,*) " maxinum GAUSS QUADRATURE RULE reached"
        write(6,*) " Decrease the order of ploynomial to get solution"
        write(6,*) "------------------------------------------------"
        goto 110
        end if
        write(6,*) " number of G-Q points",ngqpts
```

```
c
        h1     = barlt/nel
        aincob = (2.0*nel)/barlt
        ajacob =  barlt/(2.0*nel)
        nn     =  nel * npts + 1        !total number of nodes
        write(6,*) "total number of nodes ",nn
        write(6,*)"number of elements",nel
c
        call shapefun(npts,nel,barlt,ajacob,zeta,sh,sh1)
        open(unit =10, file ='ShapeCoeff.dat', status='unknown')
        do i=1,npts+1
                write(10,*)"------------------------------------------"
                write(10,*) "shape function",i, "is"
                write(10,*) (sh(i,k),'x**',npts-k+1,k=1,npts+1)
                write(10,*)"derivative of shape function",i,"is"
                write(10,*) (sh1(i,j),'x**',npts-j,j=1,npts)
                write(10,*)"------------------------------------------"
        end do
        close(10)
c
        call mesh(maxnodes,nn,xi,barlt,x)
        open(unit=11,file='mesh.dat',status='unknown')
        write(11,*)"number of nodes",nn
        write(11,*)"number of elements",nel
        write(11,*) "node no location"
        write(11,111)(i,x(i),i=1,nn)
111     format(i4,4x,1f9.5)
        close(11)
c
!       converting  EA as a function of X to zeta
        call matdata(maxnodes,nn,alpha1,alpha2,h1,x,ea)
        open(unit =12,file='young.dat',status='unknown')
        write(12,*) "location ea(1) ea(2) "
        do i=1,nn
        write(12,*) x(i),ea(i,1),ea(i,2)
        end do
        close(12)
c
!       converting F as a function of X to zeta
        call forcedata(maxnodes,f1,f2,f3,h1,nel,npts,x,f)
        open(unit=13,file='force.dat',status='unknown')
        write(13,*) "location f(1) f(2) f(3)"
        do i=1,nn
        write(13,*) x(i), f(i,1), f(i,2), f(i,3)
        end do
        close(13)
!       calculating Gauss quadrature points and weights
        call gauss(ngqpts,npts,weight,gqpt,maxgqpts)
        open(unit =14,file='GSpoints.dat',status='unknown')
        write(14,*) "number of  gauss quadrature pts",ngqpts
        write(14,*) "Gauss Quadrature points"
        write(14,*)" pointNo. location, weight "
         do i=1, ngqpts
        write(14,*) i,gqpt(i),weight(i)
        end do
        close(14)
c
!        Global stiffness matrix and load vector formation
```

```
      call elemstiff(maxnodes,ea,sh,sh1,f,ajacob,aincob,weight,gqpt,
     $                dampcof,barlt,npts,nn,nel,ngqpts,stiffmt,
     $                forcemt)
c
      call boundary(maxnodes,nn,ibtype1,ibtype2,
     $                bdata1,bdata2,stiffmt,forcemt,finalk,finalf)
c
      open(unit=15,file='forceVector.dat',status='unknown')
      write(15,*) "number of nodes is=",nn
      write(15,*)" force vector is"
      open(unit=16,file='Stiffmatrix.dat',status='unknown')
      write(16,*)"stiffness matrix of order",nn,'x',nn
      write(16,*) "stiffness matrix is"
      do i=1,nn
              write(16,161)(finalk(i,j),j=1,nn)
              write(15,151) finalf(i)
      end do
151   format(2x,1e12.5)
161   format(4(2x,1f9.5))
      close(15)
      close(16)
c
      CALL ELIMIN(FINALK,FINALF,MAXNODES,NN,SOLN)
c
      twicese = 0.0d0
      do i=1,nn
      twicese = twicese +finalf(i) * soln(i)
      end do
      write(6,*) "strain energy",twicese/2.0
c
      open(unit=17,file='se.dat',status='unknown',access='append')
      write(17,*)"number of nodes",nn
      write(17,*) "strain energy",twicese/2.0
      close(17)
c
      do i=1,nn
      exact(i) = - x(i)**2/2.0 + 11.0 *x(i)
      end do
      do i=1,nn-1
      eder(i) = -(x(i)+ x(i+1))/2.0+ 11.0
      der(i) = (soln(i+1) - soln(i))*2.0/h1
      end do

      open(unit=18,file="slope.dat",status="unknown")
      write(18,*)""
      write(18,*) "SLOPE AT MID POINT"
      write(18,*) "location,slope_exact,slope_fem"
      write(18,*)"number of nodes",nn
      do i=1,nn-1
      write(18,181)(x(i)+x(i+1))/2.0, eder(i),der(i)
      end do
181   format(1f9.3,2(2x,1f9.3))
      close(18)

      open(unit=19,file='disp.dat',status='unknown')
      write(19,*) 'SOLUTON OF ', nn,' X ', nn, " EQUATIONS BY GAUSS
     $ ELIMINATION METHOD is"
      write(19,*)"LOCATON  DISPLACEMENT_fem  DISPLACEMENT_exact"
```

```
        write(19,*)"number of nodes",nn
        do i=1,nn
        write(19,191) x(i),soln(i), exact(i)
        end do
191     format(1f9.3,2(2x,1f10.4))
        close(19)
c
100     format(a30)
110     continue
90      continue
        end
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        subroutine boundary(maxnodes,nn,ibtype1,ibtype2,bdata1,
     $                          bdata2,stiffmt,forcemt,condK,condF)
        dimension condK(maxnodes,maxnodes), condF(maxnodes)
        dimension stiffmt(maxnodes,maxnodes), forcemt(maxnodes)
        dimension bdata1(5),bdata2(5)
c
        do i=1,nn
                do j=1,nn
                condK(i,j) = stiffmt(i,j)
                end do
        condF(i) = forcemt(i)
        end do
!******************* bytype1 is specified at X=0 *******************
!       ibctype1 =1 for displacement boundary condition at x=0
!                               Uo  o-----------
!        Uo = bdata1(1)
        if (ibtype1.eq.1) then
        condF(1) = bdata1(1)
        do i=2,nn
                condF(i) = condF(i) - stiffmt(i,1)*bdata1(1)
                condK(i,1) = 0.0d0
                condK(1,i) = 0.0d0
        end do
        condK(1,1) = 1.0d0
        end if
!       ibctype1 =2 for force boundary condition at X=0
!                               P1 <------ ||---------
!       P1 = bdata1(1)
        if(ibtype1.eq.2) then
                condF(1) = condF(1) - bdata1(1)
!       ibctype1 =3 for spring load at X =0
!                                          ||----^^^^---
!                                                k1, delta1
!       k1       = bdata1(1)
!       delata1  = bdata1(2)
        else if (ibtype1.eq.3) then
                condK(1,1) = condK(1,1) + bdata1(1)
                condF(1)   = condF(1) + bdata1(1)*bdata1(2)
! ************   Implementation of Bctype1 is completed **************
! ************    Implementing  bytype2 at X=L          **************
!        ibctype2 =1 for displacement boundary condition at x=L
!                                 -----------o U_L
!       U_L = bdtata2(1)
        else if (ibtype2.eq.1) then
                condF(nn) = bdata2(1)
                do i=1,nn-1
```

```
                condF(i) = condF(i) - stiffmt(i,nn) * bdata2(1)
                condK(i,nn) = 0.0d0
                condK(nn,i) = 0.0d0
                end do
!       ibctype2 =2 for force boundary condition at X=0
!                                -----------|| -----> P2
!       P2 = bdata2(1)
        else if (ibtype2.eq.2) then
                condF(nn) = condF(nn) + bdata2(1)
!       ibctype2 =3 for spring load at X =0
!                                ----^^^^---||
!                                   k2, delta2
!       k2        = bdata2(1)
!       delata2  = bdata2(2)
        else if (ibtype2.eq.3) then
                condK(nn,nn) = condK(nn,nn) + bdata2(1)
                condF(nn) = condF(nn) - bdata2(1)*bdata2(2)
        end if
! ************  Implementing  bytype2 at X=L is completed  **************
        return
        end
c
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        subroutine elemstiff(maxnodes,ea,sh,sh1,f,ajacob,aincob,weight,
     $                  gqpt,dampcof,barlt,npts,nn,nel,ngqpts,globalk,globalf)
c
        implicit real(a-h,o-z)
        dimension sh(15,15),sh1(15,15),f(maxnodes,15),weight(15)
        dimension eleK(15,15),eleF(15),gqpt(15),ea(maxnodes,15)
        dimension globalk(maxnodes,maxnodes), globalf(maxnodes)
        dimension temp(maxnodes,15),temp1(maxnodes,15),ym(maxnodes,15)
        dimension ftemp(maxnodes,15),tempea(maxnodes,15)
        dimension tempfr(maxnodes,15)
        integer  ij,jk,itemp,icon
c
        ajacob = barlt/(2.0*nel)
!----------------------  INTIALISATION  ---------------------------------
        do  i=1,npts+1
                do j=1, ngqpts
                        temp(i,j) = 0.0d0
                        if (i.ge.2) then
                                temp1(i-1,j) = 0.0
                        end if
                        ym(i,j)      = 1.0d0
                        ftemp(i,j)   = 1.0d0
                end do
        end do
!-------------------------  COMPLETED  ----------------------------------
!------------ INITIALISION LOCAL STIFFNESS MATRIX AND LOAD VECTOR --------
        do i=1,npts+1
                do j=1,npts+1
                eleK(i,j) =0.0d0
                end do
                eleF(i)  =0.0d0
        end do
!---------------------------- COMPLETED ---------------------------------
!-------- INITIALISATION OF GLOBAL STIFFNESS MATRIX AND LOAD VECTOR ------
        do i = 1,nn
```

```
                do j=1,nn
                        globalk(i,j) = 0.0d0
                end do
        globalf(i) = 0.0d0
        end do
!------------------------------ COMPLETED --------------------------------
!------  EVALUATING LOCAL SHAPE FUNCTIONS AND IT'S DERIVATIVES AT THE-----
!----------              GAUSS QUADRATURE POINTS        ------------------
!--------------------------- shape functions -----------------------------
        do i=1,ngqpts
                do j=1,npts+1
                        do k=1,npts+1
                               temp(j,i) = temp(j,i) + sh(j,k)*
     $                         (gqpt(i)**(npts+1-k))
                        end do
                end do
        end do
!---------------------- shape function's derivatives ---------------------
        do i=1,ngqpts
               do j=1,npts+1
                         do k=1,npts
                               temp1(j,i) = temp1(j,i) + (sh1(j,k)*
     $                         gqpt(i)**(npts-k))
                        end do
                end do
        end do
!-------------------------------------------------------------------------
!------  VALUE OF EA AND F AT GAUSS QUADRATURE POINTS FOR ALL THE NODES --
        do i=1,nn
                do k=1,ngqpts
                        ym(i,k) = ea(i,1) + ea(i,2) * gqpt(k)
                        ftemp(i,k) = f(i,1) + f(i,2)*gqpt(k)+ f(i,3) *
     $                  gqpt(k)**2
                end do
        end do
!-------------------------------------------------------------------------
!------------ FORMATIONS OF A LOCAL STIFFNESS MATRIX and LOAD VECTOR for--
!------------each element AND ADD IT TO THE CORRESPONDING GLOBAL STIFFNESS-
!------------MATRIX AND LOAD VECTOR -------------------------------------
        l = 1                  !starting node number of an element
        m = npts+1             !ending node number of an element
        do itemp = 1,nel                       !number of elements
!------------ LOCAL STIFFNESS MATRIX for one element ----------
        do i=1,npts+1
        icon = (itemp-1)*npts + i
                         do j=1,ngqpts
                        tempea(i,j) = ym(icon,j)
                        tempfr(i,j) = ftemp(icon,j)
                        end do
        end do
c
        do i=1,npts+1
                do j=1,npts+1
                        do k=1,ngqpts
                        eleK(i,j) = eleK(i,j) + tempea(i,k)*
     $                                  temp1(i,k)*temp1(j,k)
     $                                  *aincob *weight(k) +
     $                                  dampcof*temp(i,k)*
```

```
     $                                 temp(j,k)*ajacob*weight(k)
                        end do
                end do
        end do
!-----------------  LOCAL LOAD VECTOR for one element------------
        do i=1,npts+1
                do j=1,ngqpts
                        eleF(i) = eleF(i) + tempfr(i,j)*temp(i,j)*
     $                          weight(j)*ajacob
                end do
        end do
!----------------------------------------------------------------
!*****************************  ASSEMBLY **********************************
!       Adding the local element stiffness matrix and load vector to
!       global stiffness matrix and load vector for one element
        do i=1, npts+1
                j=itemp
                ij = (j-1)*npts + i
                globalf(ij) = globalf(ij) + eleF(i)
                        do k = 1,npts+1
                                jk = (j-1)*npts + k
                                globalk(ij,jk) = globalk(ij,jk)+ eleK(i,k)
                        end do
        end do
!--------------------------------------------------------------------------
!------ setting LOCAL STIFFNESS MATRIX and LOAD VECTOR to ZERO for
!                       the next element -------------------------------
         do i=1,npts+1
                do j=1,npts+1
                eleK(i,j) =0.0d0
                end do
        eleF(i) = 0.0d0
        end do
!--------------------------------------------------------------------------
        l = l+ npts
        m = m+ npts
        end do
        return
        end
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        subroutine mesh(maxnodes,nn,xi,barlt,y)
        implicit real(a-h,o-z)
        dimension y(maxnodes)
        real  temp
        temp = 0.0d0
        temp = 1.0d0 *barlt/(nn-1)
!       xi    = starting point
!       barlt = length of the bar
        do i=1,nn
                y(i)  = xi + 1.0d0 *(i-1)* temp
        end do
c
        return
        end
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        subroutine gauss(ngqpts,npts,wght,qpt,maxn)
        dimension wght(15),qpt(15)
!       ONE POINT GAUSS QUADRATURE RULE
```

```
        if(ngqpts.eq.1) then
                qpt(1)   =  0.0d0
                wght(1)  =  2.0d0
!       TWO POINT GAUSS QUADRATURE RULE
        else if (ngqpts.eq.2) then
                qpt(1)   =   -sqrt(1.0/3)
                qpt(2)   =   -qpt(1)
                wght(1)  =   1.0d0
                wght(2)  =   wght(1)
!       THREE POINT GAUSS QUADRATURE RULE
        else if (ngqpts.eq.3) then
                qpt(1)   =   - sqrt(3.0d0/5)
                qpt(2)   =   0.0d0
                qpt(3)   =  -qpt(1)
                wght(1)  =   5.0d0/9
                wght(2)  =   8.0d0/9
                wght(3)  =   wght(1)
!       FOUR POINT GAUSS QUADRATURE RULE
        else if(ngqpts.eq.4) then
                qpt(1)   =     sqrt((3.0 + (2.0*sqrt(6.0/5)))/7.0)
                qpt(2)   =     sqrt((3.0 - (2.0*sqrt(6.0/5)))/7)
                qpt(3)   =    -qpt(2)
                qpt(4)   =    -qpt(1)
                wght(1)  =     (18.0 - sqrt(30.0))/36
                wght(2)  =     (18.0 + sqrt (30.0))/36
                wght(3)  =     wght(2)
                wght(4)  =     wght(1)
!       FIVE POINT GAUSS QUADRATURE RULE
        else if(ngqpts.eq.5) then
                 qpt(1)   =   (sqrt(5.0+(2*sqrt(10.0/7))))/3
                 qpt(2)   =   (sqrt(5.0-(2*sqrt(10.0/7))))/3
                 qpt(3)   =   0.0d0
                 qpt(4)   =   -qpt(2)
                 qpt(5)   =   -qpt(1)
                 wght(1)  =   (322.0 - (13.0 * sqrt(70.0)))/900
                 wght(2)  =   (322.0 + (13.0 * sqrt(70.0)))/900
                 wght(3)  =   128.0/225
                 wght(4)  =    wght(2)
                 wght(5)  =    wght(1)
!       SIX POINT GAUSS QUADRATURE RULE
        else if (ngqpts.eq.6) then
                qpt(1)   =      -.9324695142032520
                qpt(2)   =      -.6612093864662650
                qpt(3)   =      -.2386191860831970
                qpt(4)   =      -qpt(3)
                qpt(5)   =      -qpt(2)
                qpt(6)   =      -qpt(1)
                wght(1)  =      0.1713244923791700
                wght(2)  =      0.3607615730481390
                wght(3)  =      0.4679139345726910
                wght(4)  =      wght(3)
                wght(5)  =      wght(2)
                wght(6)  =      wght(1)
!       SEVEN POINT GAUSS QUADRATURE RULE
        else if (ngqpts.eq.7) then
                qpt(1)  =       -0.9491079123427590
                qpt(2)  =       -0.7415311855993940
                qpt(3)  =       -0.4058451513773970
```

```
                qpt(4)  =       0.0
                qpt(5)  =       -qpt(3)
                qpt(6)  =       -qpt(2)
                qpt(7)  =       -qpt(1)
                wght(1) =       0.1294849661688700
                wght(2) =       0.2797053914892770
                wght(3) =       0.3818300505051190
                wght(4) =       0.4179591836734690
                wght(5) =       wght(3)
                wght(6) =       wght(2)
                wght(7) =       wght(1)
!       EIGHT POINT GAUSS QUADRATURE RULE
        else if (ngqpts.eq.8) then
                qpt(1)  =       -0.9602898564975360
                qpt(2)  =       -0.7966664774136270
                qpt(3)  =       -0.5255324099163290
                qpt(4)  =       0.1834346424956500
                qpt(5)  =       -qpt(4)
                qpt(6)  =       -qpt(3)
                qpt(7)  =       -qpt(2)
                qpt(8)  =       -qpt(1)
                wght(1) =       0.1012285362903760
                wght(2) =       0.2223810344533740
                wght(3) =       0.3137066458778870
                wght(4) =       0.3626837833783620
                wght(5) =       wght(4)
                wght(6) =       wght(3)
                wght(7) =       wght(2)
                wght(8) =       wght(1)
!       NINE POINT GAUSS QUADRATURE RULE
        else if(ngqpts.eq.9) then
                qpt(1)  =       -0.9681602395076260
                qpt(2)  =       -0.8360311073266360
                qpt(3)  =       -0.6133714327005900
                qpt(4)  =       -0.3242534234038090
                qpt(5)  =       0.0
                qpt(6)  =       -qpt(4)
                qpt(7)  =       -qpt(3)
                qpt(8)  =       -qpt(2)
                qpt(9)  =       -qpt(1)
                wght(1) =       0.0812743883615740
                wght(2) =       0.1806481606948570
                wght(3) =       0.2606106964029350
                wght(4) =       0.3123470770400030
                wght(5) =       0.3302393550012600
                wght(6) =       wght(4)
                wght(7) =       wght(3)
                wght(8) =       wght(2)
                wght(9) =       wght(1)
!       TEN POINT GAUSS QUADRATURE RULE
        else if (ngqpts.eq.10) then
                qpt(1)   =      -0.9739065285171720
                qpt(2)   =      -0.8650633666889850
                qpt(3)   =      -0.6794095682990240
                qpt(4)   =      -0.4333953941292470
                qpt(5)   =      -0.1488743389816310
                qpt(6)   =      -qpt(5)
                qpt(7)   =      -qpt(4)
```

```
              qpt(8)   =      -qpt(3)
              qpt(9)   =      -qpt(2)
              qpt(10)  =      -qpt(1)
              wght(1)  =      0.0666713443086880
              wght(2)  =      0.1494513491505810
              wght(3)  =      0.2190863625159820
              wght(4)  =      0.2692667193099960
              wght(5)  =      0.2955242247147530
              wght(6)  =      wght(5)
              wght(7)  =      wght(4)
              wght(8)  =      wght(3)
              wght(9)  =      wght(2)
              wght(10) =      wght(1)
        end if
c
        return
        end
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        subroutine forcedata(maxnodes,f1,f2,f3,h1,nel,npts,x,force)
        implicit real(a-h,o-z)
        dimension force(maxnodes,5),x(maxnodes)
!        f     = f1 + f2 *x + f3 * x**2
!        f(i)  = f(i,1) + f(i,2) *zeta + f(i,3) * zeta**2
        do i=1,npts*nel+1
        force(i,1) = f1 + (f2*h1/2.0d0)+(f3*h1*h1/4.0d0)+
     $  x(i)*(f2+f3*x(i)+f3*h1)
        force(i,2) = (f2*h1/2.0d0)+(f3*h1*x(i))+(f3*h1*h1/2.0d0)
        force(i,3) = f3*h1**2/4.0d0
        end do
c
        return
        end

!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        subroutine matdata (maxnodes,nn,alpha1,alpha2,h1,x,young)
        implicit real (a-h,o-z)
        dimension young(maxnodes,5) ,x(maxnodes)
!       EA     = alpha1 + alpha2 * x
!       EA(i)  = young(i,1) + young(i,2) * zeta
        do i=1,nn
        young(i,1) = (alpha1 +(alpha2*h1)/2.0d0) +alpha2 *x(i)
        young(i,2) = alpha2*h1/2.0d0
        end do
c
        return
        end
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        subroutine shapefun(npts,nel,barlt,ajacob,zeta,b,b1)
        implicit real(a-h,o-z)
        dimension zeta(15),b(15,15),b1(15,15)
        common i,j,k,l,m,n,n1,n2,n3,n4,h
         real*8 const(15),c0(15),c1(15)
         real*8  c2(15),c3(15),c4(15),c5(10),c6(15)
         real*8  c7(15),c8(15),c9(15)
c
        do i=1,npts+1
                c0(i)  = 0.0d0
                c1(i)  = 0.0d0
```

```
                c2(i)  = 0.0d0
                c3(i)  = 0.0d0
                c4(i)  = 0.0d0
                c5(i)  = 0.0d0
                c6(i)  = 0.0d0
                c7(i)  = 0.0d0
                c8(i)  = 0.0d0
                c9(i)  = 0.0d0
                const(i) = 1.0d0
                do j=1,npts+1
                        b(i,j)  = 1.0d0
                        b1(i,j) = 1.0d0
                end do
        end do
!       INITIAL AND FINAL POINT
        do i=1,npts+1
        zeta(i) = 1.0 - 2.0*(i-1)/npts
        end do
c
        ajacob = barlt/(nel*2.0d0)
c
        do i=1,npts+1
        c0(i) = c0(i)+1.0d0

          do j=1,npts+1
            if(j.ne.i) then
               c1(i) = c1(i)+zeta(j)
                do k=j+1,npts+1
                  if (k.ne.i) then
                    c2(i) = c2(i) + zeta(j)*zeta(k)
                     do l=k+1,npts+1
                       if (l.ne.i) then
                         c3(i) = c3(i) + zeta(j)*zeta(k)*zeta(l)
                           do m=l+1,npts+1
                              if (m.ne.i) then
                                c4(i) = c4(i) + zeta(j)*zeta(k)*zeta(l)*
     $                                  zeta(m)
                                 do n = m+1,npts+1
                                   if (n.ne.i) then
                                     c5(i) = c5(i) + zeta(j)*zeta(k)
     $                                       *zeta(l)* zeta(m)*zeta(n)
                                        do n1 =n+1,npts+1
                                          if(n1.ne.i) then
                                            c6(i) = c6(i) + zeta(j)*
     $                                      zeta(k)*zeta(l)*zeta(m)*
     $                                      zeta(n)*zeta(n1)
                                              do n2 =n1+1,npts+1
                                                if(n2.ne.i) then
                                                  c7(i) = c7(i)+zeta(j)*
     $                                          zeta(k)*zeta(l)*zeta(m)*
     $                                         zeta(n)*zeta(n1)*zeta(n2)
                                                      do n3 =n2+1,npts+1
                                                        if(n3.ne.i) then
                                                          c8(i) = c8(i)
     $                               + zeta(j)*zeta(k)*zeta(l)*zeta(m)*
     $                               zeta(n)*zeta(n1)*zeta(n2)*zeta(n3)
                                                            do n4 =n3+1,
     $                                                          npts+1
```

```
                                                              if(n4.ne.
     $                                                         i) then
                                                                c9(i) =
     $                           c9(i)+ zeta(j)*zeta(k)*zeta(l)*zeta(m)*
     $                        zeta(n)*zeta(n1)*zeta(n2)*zeta(n3)*zeta(n4)
                                                              end if

                                                           end do
                                                        end if
                                                      end do
                                                  end if
                                                end do
                                            end if
                                          end do
                                     end if
                                   end do
                                end if
                             end do
                         end if
                       end do
                    end if
                  end do
              end if
            end do
c
           do j=1,npts+1
               if (j.ne.i) then
                  const(i) = const(i)*(-zeta(i) +zeta(j))
               end if
           end do
c
           b(i,1) = c0(i)/const(i)
           b(i,2) = c1(i)/const(i)
           b(i,3) = c2(i)/const(i)
           b(i,4) = c3(i)/const(i)
           b(i,5) = c4(i)/const(i)
           b(i,6) = c5(i)/const(i)
           b(i,7) = c6(i)/const(i)
           b(i,8) = c7(i)/const(i)
           b(i,9) = c8(i)/const(i)
           b(i,10) = c9(i)/const(i)
c
           do k=1,npts
                  b1(i,k) = b(i,k) * (npts-k+1)
           end do
c
          end do
          return
          end
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!************************************************************************
!*      SOLUTION OF LINEAR SYSTEM OF EQUATIONS BY THE                   *
!*                 GAUSS- ELIMINATION METHOD                            *
!************************************************************************
!*       A      = SYSTEM OF MATRIX (STIFFNESS MATRIX)                   *
!*       B      = VECTOR  CONTAINING THE INDEPENDENENT COEFFICIENTS     *
!*       SOLN   = ARRAY WHICH, AFTER SOLUTION, WILL CONTAIN THE VALUES  *
!*                 OF THE SYSTEM OF UNKNOWNS                            *
```

```
!*       N      = ORDER OF SYSTEM                                        *
!*************************************************************************
        subroutine elimin(A,B,MAXNODES,N,SOLN)
        DIMENSION A(MAXNODES,MAXNODES), B(MAXNODES),SOLN(MAXNODES)
        WRITE (6,*) ('****GAUSSIAN ELIMINATION ****')

! CONVERT TO UPPER TRIANGULAR FORM
        DO K = 1,N-1
                IF (ABS(A(K,K)).GT.1.E-6) THEN
                        DO I = K+1, N
                        X = A(I,K)/A(K,K)
                                DO J = K+1, N
                                A(I,J) = A(I,J) -A(K,J)*X
                                ENDDO
                        B(I) = B(I) - B(K)*X
                        ENDDO
                ELSE
                WRITE (6,*) 'ZERO PIVOT FOUND IN LINE:'
                WRITE (6,*) K
                STOP
                END IF
        ENDDO
!        BACK SUBSTITUTION
        DO I = N,1,-1
                SUM = B(I)
                IF (I.LT.N) THEN
                        DO J= I+1,N
                        SUM = SUM - A(I,J)*B(J)
                        ENDDO
                END IF
        B(I) = SUM/A(I,I)
        SOLN(I) = B(I)
        ENDDO
        ! PRINT THE RESULTS
!       write(6,*) ('SOLUTION VECTOR')
        CALL PRINTV(B,MAXNODES,N,6)
        END
!-----------------------------------------
        SUBROUTINE PRINTA(A,IA,M,N,ICH)
! WRITE A 2D ARRAY TO OUTPUT CHANNEL 'ICH'
        DIMENSION A(IA,IA)
        WRITE(ICH,2) (A(I,J),J=1,N)
!       ENDDO
2       FORMAT(2X,1E12.4)
        END SUBROUTINE PRINTA
!----------------------------------------
        SUBROUTINE PRINTV(VEC,MAXNODES,N,ICH)
! WRITE A COLUMN VECTOR TO CHANNEL 'ICH'
        DIMENSION VEC(MAXNODES)
        WRITE(ICH,*)"SOLUTION VECTOR"
        WRITE(ICH,1) (VEC(I),I=1,N)
1       FORMAT(1X,1E12.4)
        END SUBROUTINE PRINTV
!----------------------------------------
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```