APPENDIX C

A Finite Element Analysis Subroutine for Cracking Localization Problem

```
// The 4-noded quadrilateral smeared element with constant bandwidth
// for checking stability.
// VERSION 960109
// written by Pruettha NANAKORN
// developed by Aruz PETCHERDCHOO
// Node numbering:
//
//        3 ---- 2
//        |      |
//        |      |
//        |      |
//        4 ---- 1
//
// Data format:
//
// E nu thick ip nrg nsg
//
// E     = Young's Modulus
// nu    = Poisson's Ratio
// thick= thickness
// ip    = nonzero -> plane stress
//       = zero    -> plane strain
// nrg   = number of integration points in the first direction
// nsg   = number of integration points in the second direction
// strsCr = tensile strength of the material
// cntrlElmNum = control element number
// cntrlDofNum = control element degree of freedom
// RnFr = retention factor

#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <math.h>
#include <stdlib.h>
#include "util.h"
#include "matrix.h"
#include "node.h"
#include "element.h"
#include "function.h"
#include "user.h"
inline Real SIGN(Real a,Real b) {return (b>=0) ? fabs(a) : -fab (a);}
static Real sqrarg;
Real SQR(Real a) {return ((sqrarg=(a)) == 0.0) ? 0.0 : sqrarg*sqrarg;}
inline void SWAP(Real &c,Real &d) {Real y=c; c=d; d=y;}


// constants ************************************************************

static const Real gp[4][4]=
{ 0.,0.,0.,0.,
  0.,               0.,          0.,          0.,
  0.,      -sqrt(1./3.), sqrt(1./3.),          0.,
  0.,      -sqrt(3./5.),          0., sqrt(3./5.)
};
static const Real gw[4][4]=
{ 0.,0.,0.,0.,
  0.,               2.,          0.,          0.,
  0.,               1.,          1.,          0.,
  0.,            5./9.,       8./9.,       5./9.
};

// functions'
prototypes*****************************************************************

static Matrix Elastic(const Material &mate);
static Matrix Inelastic(const Material &mate, const Vector &angleS,const Control
&control,ofstream &ouf,const Vector *StrsT,const Node *node,Element &elem,const Vector
&Band);
static Matrix Crack(const Material &mate, const Vector &angleS,const Control
&control,ofstream &ouf,const Vector *StrsT,const Vector &BandAll,const int kk);
static void Shape4(Matrix &SHP,Matrix &DSHP,Real &detJ,const Node *node,Real rg,Real
sg);
static void Shp4(ofstream &ouf,Matrix &SHP,Matrix &DSHP,Real &detJ,const Vector
*nodeCoor,const Matrix &elmNode, int kk,Real rg,Real sg);
static Matrix BMatr(const Matrix &DSHP);
static void PrtStrainStress(ofstream &ouf,const CoreData &core,const Control
&control,const Node *node,const Vector &Strn,const Vector &Strs);
```

```
static Vector FindLambda(ofstream &ouf,Vector &Strs,const Vector *StrsT,const Vector
&DStrs, Real strsCr,const Control &control);
static Vector AnalyseLambda(ofstream &ouf,Vector Lambda, Vector LambdaE, Real
&lambdap,Real &lambdan,const Control &control);
static Vector Bandwidth(const Material &mate,const Vector &angleS,const Node
*node,const Control &control, Element &elem,Real rg,Real sg);
static void tred2(Matrix &a, int n, Vector &d, Vector &e);
static void tqli(Vector &d, Vector &e, int n , Matrix &z);
static Real pythag(Real a, Real b);
static void nrerror(char error_text[]);
static void PrtEigensystem(ofstream &ouf, int n, Vector &d, Matrix &a, Matrix
&CndnGloK, Vector &f);

//*****************************************************************************

void STAC4E(int isw,ifstream &inf,ofstream &ouf,const CoreData &core,
        Material &mate,Element &elem,Node *node,Control &control)
{
  register int i,j,k,l,ii,jj,kk,g;
  Real E,nu,RnFr,thick,strsCr,angle;
  Real M,N;                // used in calculating principal angle
  Real detJ,dV;
  int ip;
  Matrix C(3,3);
  Matrix SHP(1,4);
  Matrix DSHP(3,4);
  Matrix B(3,8);
  Vector Strn(3);
  Vector Strs(3);
  Vector DStrn(3);
  Vector DStrnCr(3);
  Vector DStrs(3);
  Vector LambdaE(2);
  Matrix K11(8,8);
  Matrix K22(8,8);
  Matrix Del(3,3);
  Matrix Dcr(3,3);
  Vector X(4),Y(4),coef(2);
  static int nrg,nsg;
  static int crelmno,combnCr,dofnum=0,nodeAll=0,nForc=0;
  static int row11,col11,row22,col22,cln22;
  static int row12,col12,row21,col21,cln12,rw21;
  static long  crElmNumP, crElmNumN;
  static Real  lambdap, lambdan;
  static Vector *StrsT=new Vector [core.nElem+1];
  static Vector *StrnT=new Vector [core.nElem+1];
  static Vector *DU=new Vector [core.nElem+1];
  static Vector *DSts=new Vector [core.nElem+1];
  static Vector *DStn=new Vector [core.nElem+1];
  static Vector *nodedofP=new Vector [core.nNode+1];
  static Vector *nodeBoun=new Vector [core.nNode+1];
  static Vector *nodeCoor=new Vector [core.nNode+1];
  static Vector LambdaP(core.nElem);
  static Vector LambdaN(core.nElem);
  static Vector crElmNumS(core.nElem);
  static Vector bandwidth(core.nElem);
  static Vector angleS(core.nElem);
  static Vector crangle(core.nElem);
  static Vector Band(1);
  static Vector BandAll(core.nElem);
  static Vector Lambda(2);
  static Real Load;
  static Real Deflect;
  static long cntrlElmNum;
  static long cntrlDofNum;
  static Real duCntrl;
  static Real lambdaf;
  static Real DETJ,Sum;
  static Matrix GloK11;
  static Matrix InvGloK11;
  static Matrix GloK12;
  static Matrix GloK21;
  static Matrix GloK22;
  static Matrix CndnGloK;
  static Matrix CnstrnGloK;
  static Matrix a;
  static Matrix caseCr;
```

```
static Matrix elmNode(core.nElem,4);
static int combnCnt;
static Vector combnChk;

switch (isw) {
case Mate:    // Read material properties.

  ouf<<"Four-Noded Quadrilateral Element.\n";
  mate.SetNData(10);
  inf>>E;
  inf>>nu;
  inf>>thick;
  inf>>ip;
  inf>>nrg>>nsg;
  inf>>strsCr;
  inf>>cntrlElmNum>>cntrlDofNum;
  inf>>RnFr;
  ouf<<"Young's Modulus ="<<E<<"\n";
  ouf<<"Poisson's Ratio ="<<nu<<"\n";
  ouf<<"Thickness       ="<<thick<<"\n";
  ouf<<"Plain Str"<<(ip?"ess":"ain")<<"\n";
  ouf<<"Use "<<nrg<<"x"<<nsg<<" integration points"<<"\n";
  ouf<<"Tensile Strength "<<strsCr<<"\n";
  ouf<<"cntrlElmNum = "<<cntrlElmNum<<endl;
  ouf<<"cntrlDofNum = "<<cntrlDofNum<<endl;
  ouf<<"Retention Factor ="<<RnFr<<"\n";

  mate.Data(1)=E*(1.+(1.-ip)*nu)/(1.+nu)/(1.-ip*nu);
  mate.Data(2)=nu*mate.Data(1)/(1.+(1.-ip)*nu);
  mate.Data(3)=E/2./(1.+nu);
  mate.Data(4)=thick;
  mate.Data(5)=nrg;
  mate.Data(6)=nsg;
  mate.Data(7)=strsCr;
  mate.Data(8)=RnFr;

  //initialize values
  Lambda(1)=+1.e30;
  Lambda(2)=-1.e30;
  Load=0.;
  Deflect=0.;
  elmNode=0.;
  Band(1)=0.;
  for(i=1;i<=core.nElem;i++){
    bandwidth(i)=0;
    crElmNumS(i)=0;
    BandAll(i)=0;
    StrsT[i].SetDimension(3);
    StrnT[i].SetDimension(3);
    DSts[i].SetDimension(3);
    DStn[i].SetDimension(3);
    StrsT[i]=0;
    StrnT[i]=0;
    DSts[i]=0;
    DStn[i]=0;
  }
  for(j=1;j<=core.nNode;j++){
    nodeBoun[j].SetDimension(2);
    nodedofP[j].SetDimension(2);
    nodeCoor[j].SetDimension(2);
    for(k=1;k<=2;k++){
      nodeBoun[j](k)=0;
      nodedofP[j](k)=0;
      nodeCoor[j](k)=0;
    }
  }

  break;


case Tang:    // Compute stiffness matrix.

  thick=mate.Data(4);
  nrg=(int) mate.Data(5);
  nsg=(int) mate.Data(6);
  if(crElmNumS(control.elmNum)==0){
    C=Elastic(mate);
```

```
    }
    else if(crElmNumS(control.elmNum)==1){
      Sum=0.;
      DETJ=0.;
      for(i=1;i<=nrg;i++){
          for(j=1;j<=nsg;j++){
            coef=Bandwidth(mate,angleS,node,control,elem,gp[nrg][i],gp[nsg][j]);
          Sum+=coef(1)*coef(2);
          DETJ+=coef(2);
          }
      }
      Band(1)=DETJ/Sum;
      C=Inelastic(mate,angleS,control,ouf,StrsT,node,elem,Band);
    }
    for(i=1;i<=nrg;i++){
      for(j=1;j<=nsg;j++){
        Shape4(SHP,DSHP,detJ,node,gp[nrg][i],gp[nsg][j]);
        B=BMatr(DSHP);
        dV=detJ*gw[nrg][i]*gw[nsg][j]*thick;
        elem.S+=(dV*B->*C*B);
      }
    }

    break;


  case Stre:    // Compute strain & stress.

    thick=mate.Data(4);
    strsCr=mate.Data(7);
    Shape4(SHP,DSHP,detJ,node,0.,0.);
    if(crElmNumS(control.elmNum)==0){
      C=Elastic(mate);
    }
    else if(crElmNumS(control.elmNum)==1){
      coef=Bandwidth(mate,angleS,node,control,elem,0.,0.);
      Band(1)=1/coef(1);
      C=Inelastic(mate,angleS,control,ouf,StrsT,node,elem,Band);
    }
    B=BMatr(DSHP);
    DStrn=B*elem.DU;
    DStn[control.elmNum]=DStrn;
    DStrs=C*DStrn;
    DU[control.elmNum]=elem.DU;
    DSts[control.elmNum]=DStrs;
    PrtStrainStress(ouf,core,control,node,DStrn,DStrs);
    if(crElmNumS(control.elmNum)==0){        //Find cracked element
      LambdaE=FindLambda(ouf,Strs,StrsT,DStrs,strsCr,control);
    }
    else if(crElmNumS(control.elmNum)==1){
      LambdaE(1)=1.e30;
      LambdaE(2)=-1.e30;
    }

    Lambda=AnalyseLambda(ouf,Lambda,LambdaE,lambdap,lambdan,control);

    //  store lambdap and lambdan
    LambdaP(control.elmNum)=lambdap;
    LambdaN(control.elmNum)=lambdan;

    //  keep du-control with coefficient
    if(control.elmNum==cntrlElmNum){
      duCntrl=(elem.DU(cntrlDofNum));
    }

    // keep disp and forc boundary conditions
    if(control.time==1){
      for(i=1;i<=4;i++){
        elmNode(control.elmNum,i)=elem.Node(i);
        for(j=1;j<=core.nNode;j++)
          if(elem.Node(i)==j){
            for(k=1;k<=2;k++)
              if(node[i].Boun(k)!=0) nodeBoun[j](k)=1;
          }
      }
      for(i=1;i<=4;i++){
        for(j=1;j<=core.nNode;j++)
```

```
      if(elem.Node(i)==j){
        for(k=1;k<=2;k++)
          if(node[i].Forc(1,k)!=0){
            nodeBoun[j](k)=1;
          }
      }
  }
}

//  find cracked element
if(control.elmNum==core.nElem){
  if(duCntrl>0){
    lambdaf=Lambda(1);
    for(i=1;i<=core.nElem;i++){
      StrsT[i]=StrsT[i]+(lambdaf*DSts[i]);
      StrnT[i]=StrnT[i]+(lambdaf*DStn[i]);
      if(LambdaP(i)-Lambda(1)<1e2 && crElmNumS(i)!=1){
        M=(2*StrsT[i](3))/(StrsT[i](1)-StrsT[i](2));
        N=(atan(M))/2;
        angle=(180*N*7)/22;
        angleS(i)=angle;
        crElmNumS(i)=1;
      }
    }
  }
  else if(duCntrl<0){
    lambdaf=Lambda(2);
    for(i=1;i<=core.nElem;i++){
      StrsT[i]=StrsT[i]+(lambdaf*DSts[i]);
      StrnT[i]=StrnT[i]+(lambdaf*DStn[i]);
      if(LambdaN(i)-Lambda(2)>-1e2 && crElmNumS(i)!=1){
        M=(2*StrsT[i](3))/(StrsT[i](1)-StrsT[i](2));
        N=(atan(M))/2;
        angle=(180*N*7)/22;
        angleS(i)=angle;
        crElmNumS(i)=1;
      }
    }
  }


  crelmno=0;
  combnCr=1;
  for(i=1;i<=core.nElem;i++)
    if(crElmNumS(i)==1){
      crelmno=crelmno+1;
      combnCr=2*combnCr;
    }
  if(control.time==1){
    for(g=1;g<=core.nNode;g++)
      for(j=1;j<=2;j++)
        if(nodeBoun[g](j)==0){
          dofnum=dofnum+1;
          nodedofP[g](j)=dofnum;
        }
  }



  //  Clear Lambda
  Lambda(1)=+1.e30;
  Lambda(2)=-1.e30;
}
if(control.time==1){
  for(k=1;k<=core.nElem;k++){
    Sum=0.;
    DETJ=0.;
    for(i=1;i<=nrg;i++){
      for(j=1;j<=nsg;j++){
        coef=Bandwidth(mate,angleS,node,control,elem,gp[nrg][i],gp[nsg][j]);
        Sum+=coef(1)*coef(2);
        DETJ+=coef(2);
      }
    }
    BandAll(k)=DETJ/Sum;
  }
  for(j=1;j<=4;j++){
```

```
      for(k=1;k<=2;k++){
        nodeCoor[(int) elmNode(control.elmNum,j)](k)=node[j].Coor(k);
      }
    }
  }

  break;


case Usr7:  // Find eigenvalue for checking stability

  if(control.elmNum==1){
    thick=mate.Data(4);
    nrg=(int) mate.Data(5);
    nsg=(int) mate.Data(6);
    static int x11,x12,x21,x22,fixdof,tmp1,tmp2;
    static int stRow,stCol,Cnsn,CnRowCol;
    static int stCol1,stCol2,stCol4,stCol6,stCol8,stRow1,stRow6,stRow8,stRoTmp;
    static int RowCol1,RowCol2,TmpCrelmno;

    // construct combination crack
    caseCr.SetDimension(combnCr,core.nElem);
    combnChk.SetDimension(combnCr);
    for(i=1;i<=combnCr;i++)  combnChk(i)=1;
    caseCr=0.;
    tmp1=combnCr;
    tmp2=1;
    for(j=1;j<=core.nElem;j++){
      if(crElmNumS(j)==1){
        tmp1=tmp1/2;
        tmp2=tmp2*2;
        for(k=1;k<=tmp2;k++){
          for(i=tmp1*k-tmp1+1;i<=tmp1*k;i++){
            if(k%2==1) caseCr(i,j)=1;
            else if(k%2==0) caseCr(i,j)=0;
          }
        }
      }
    }
    for(jj=1;jj<=combnCr-1;jj++){
      TmpCrelmno=0;
      for(ii=1;ii<=core.nElem;ii++){
        if(caseCr(jj,ii)==1){
          TmpCrelmno=TmpCrelmno+1;
        }
      }

      RowCol1=dofnum;
      RowCol2=8*TmpCrelmno;
      GloK11.SetDimension(RowCol1,RowCol1);
      InvGloK11.SetDimension(RowCol1,RowCol1);
      GloK12.SetDimension(RowCol1,RowCol2);
      GloK21.SetDimension(RowCol2,RowCol1);
      GloK22.SetDimension(RowCol2,RowCol2);
      CndnGloK.SetDimension(RowCol2,RowCol2);
      GloK11=0.; GloK12=0.;
      GloK21=0.; GloK22=0.;
      InvGloK11=0.; CndnGloK=0.;
      row11=0; col11=0; row12=0; col12=0;
      row21=0; col21=0; row22=0; col22=0;
      cln12=0; rw21=0; cln22=0;
      x11=0,x12=0,x21=0,x22=0;
      for(kk=1;kk<=core.nElem;kk++){
        K11=0.,K22=0.;
        if(caseCr(jj,kk)==1){
          Del=Elastic(mate);
          Dcr=Crack(mate,angleS,control,ouf,StrsT,BandAll,kk);
          for(i=1;i<=nrg;i++){
            for(j=1;j<=nsg;j++){
              Shp4(ouf,SHP,DSHP,detJ,nodeCoor,elmNode,kk,gp[nrg][i],gp[nsg][j]);
              B=BMatr(DSHP);
              dV=detJ*gw[nrg][i]*gw[nsg][j]*thick;
              K11+=(dV*B->*Del*B);
              K22+=(dV*B->*Dcr*B);
            }
          }
          cln12=x12*8;
```

```
    x12=x12+1;
    int fac=0;
    for(k=1;k<=4;k++){
      for(g=1;g<=2;g++){
        fac=fac+1;
        if(nodedofP[(int) elmNode(kk,k)](g)!=0){
          row12= (int) nodedofP[(int) elmNode(kk,k)](g);
          col12=cln12;
          for(j=1;j<=8;j++){
            col12=col12+1;
            GloK12(row12,col12)=(-K11(fac,j));
          }
        }
      }
    }
    GloK21=Transpose(GloK12);

    cln22=x22*8;
    x22=x22+1;
    for(i=1;i<=8;i++){
      row22=row22+1;
      col22=cln22;
      for(j=1;j<=8;j++){
        col22=col22+1;
        GloK22(row22,col22)=K11(i,j)+K22(i,j);
      }
      ouf<<endl;
    }
  }
  else if(caseCr(jj,kk)==0){
    Del=Elastic(mate);
    for(i=1;i<=nrg;i++){
      for(j=1;j<=nsg;j++){
        Shp4(ouf,SHP,DSHP,detJ,nodeCoor,elmNode,kk,gp[nrg][i],gp[nsg][j]);
        B=BMatr(DSHP);
        dV=detJ*gw[nrg][i]*gw[nsg][j]*thick;
        K11+=(dV*B->*Del*B);
      }
    }
  }
  int fac1=0;
  int fac2=0;
  for(k=1;k<=4;k++){
    for(g=1;g<=2;g++){
      fac1=fac1+1;
      if(nodedofP[(int) elmNode(kk,k)](g)!=0){
        row11= (int) nodedofP[(int) elmNode(kk,k)](g);
        fac2=0;
        for(i=1;i<=4;i++){
          for(j=1;j<=2;j++){
            fac2=fac2+1;
            if(nodedofP[(int) elmNode(kk,i)](j)!=0){
              col11= (int) nodedofP[(int) elmNode(kk,i)](j);
              GloK11(row11,col11)+=K11(fac1,fac2);
            }
          }
        }
      }
    }
  }

  if(kk==core.nElem){
    InvGloK11=Inverse(GloK11);
    CndnGloK=GloK22-GloK21*InvGloK11*GloK12;      //Condense stiffness

    // For constraint of x- and y-translation, and rotation
    stRow=0,stCol=0,Cnsn=0,CnRowCol=0;
    stRow1=0,stRow6=0,stRow8=0,stRoTmp=0;
    stCol1=0,stCol2=0,stCol4=0,stCol6=0,stCol8=0;
    CnRowCol=5*TmpCrelmno;
    CnstrnGloK.SetDimension(CnRowCol,CnRowCol);
    a.SetDimension(CnRowCol,CnRowCol);
    CnstrnGloK=0.; a=0.;
    for(i=1;i<=TmpCrelmno;i++){
      stRow1=8*i-7; stRow6=8*i-2; stRow8=8*i;
      for(k=8*i-5;k<=8*i;k++){
        if(k!=8*i-4){
```

```
                if(k%2==1) stRoTmp=8*i-7;
                else if(k==stRow6) stRoTmp=8*i-4;
                else if(k==stRow8) stRoTmp=8*i-6;
                stRow=stRow+1;
                stCol=0;
                for(g=1;g<=TmpCrelmno;g++){
                  for(j=8*g-5;j<=8*g;j++){
                    if(j!=8*g-4){
                      stCol=stCol+1;
                      stCol1=8*g-7;
                      stCol2=8*g-6; stCol4=8*g-4; stCol6=8*g-2; stCol8=8*g;
                      if(j%2==1){
                        CnstrnGloK(stRow,stCol)=CndnGloK(k,j)-CndnGloK(k,stCol1)-
                          CndnGloK(stRoTmp,j)+CndnGloK(stRoTmp,stCol1);
                      }
                      else if(j==stCol6){
                        CnstrnGloK(stRow,stCol)=CndnGloK(k,j)-CndnGloK(k,stCol4)-
                          CndnGloK(stRoTmp,j)+CndnGloK(stRoTmp,stCol4);
                      }
                      else if(j==stCol8){
                        CnstrnGloK(stRow,stCol)=CndnGloK(k,j)-CndnGloK(k,stCol2)-
                          CndnGloK(stRoTmp,j)+CndnGloK(stRoTmp,stCol2);
                      }
                    }
                  }
                }
              }
            }
          }

          //Find the eigenvalue
          a=CnstrnGloK;
          int NP=0;
          NP=CnRowCol;
          Vector d(NP);
          Vector e(NP);
          Vector f(NP);
          tred2(a,NP,d,e);
          tqli(d,e,NP,a);
          PrtEigensystem(ouf,NP,d,a,CnstrnGloK,f);
          ouf<<endl;
          for(i=1;i<=NP;i++){
            if(d(i)<0){
              combnChk(jj)=0;
            }
          }
        }
      }
    }
    combnCnt=-1;                          //-1 means do not count the case of no crack
    for(j=1;j<=combnCr;j++){
      if(combnChk(j)==1) combnCnt=combnCnt+1;
    }
    i=0;
    for(j=1;j<=combnCr-1;j++){        //-1 means do not count the case of no crack
      if(combnChk(j)==1){
        cout<<endl<<"check which combination is stable = "<<j<<endl;
        cout<<"check which element is cracked = ";
        for(ii=1;ii<=core.nElem;ii++){
          cout<<caseCr(j,ii)<<" ";
        }
        cout<<endl;
      }
    }
    cout<<endl<<"Number  of stable cases = "<<combnCnt<<endl;
  }

  break;


case FacU:  // Factor DU

  control.userFacU=lambdaf;

  break;

default:
```

```
    cerr<<"A Macro is not available in this element."<<endl;

  }
}

// *************************************************************************

Matrix Elastic(const Material &mate)
{
  Matrix CE(3,3);
  CE=0.;
  CE(1,1)=CE(2,2)=mate.Data(1);
  CE(1,2)=CE(2,1)=mate.Data(2);
  CE(3,3)=mate.Data(3);

  return CE;

}

// *************************************************************************

Matrix Inelastic(const Material &mate,const Vector &angleS,const Control
&control,ofstream &ouf,const Vector *StrsT,const Node *node,Element &elem,const Vector
&Band)
{
  Matrix CI(3,3);
  Matrix Cl(3,3);
  Matrix N(3,2);
  Matrix Ntran(2,3);
  Matrix Cin(2,2);
  Matrix Cn(2,2);
  Matrix LDStn(2,1);
  Matrix Stst(3,1);
  Matrix StrsTN(2,1);

  CI=0.;
  Cl=0.;
  N=0.;
  Cn=0.;
  Ntran=0.;
  Cin=0.;
  Real z,be,E;
  int i,j,k;
  Cl(1,1)=Cl(2,2)=mate.Data(1);
  Cl(1,2)=Cl(2,1)=mate.Data(2);
  Cl(3,3)=mate.Data(3);

  z=angleS(control.elmNum)*22/(7*180);
  Ntran(1,1)=cos(z)*cos(z); Ntran(1,2)=sin(z)*sin(z); Ntran(1,3)=2*cos(z)*sin(z);
  Ntran(2,1)=-sin(z)*cos(z); Ntran(2,2)=sin(z)*cos(z); Ntran(2,3)=cos(z)*cos(z)-sin
(z)*sin(z);

  N=Transpose(Ntran);

  for(i=1;i<=3;i++){
    Stst(i,1)=StrsT[control.elmNum](i);
  }

  StrsTN=Ntran*Stst;
  Cn(1,1)=-10*Band(1)/0.025;
  Cn(2,2)=0;

  Cin=Inverse(Cn+(Ntran*Cl*N));
  CI=Cl-(Cl*N*Cin*Ntran*Cl);

  return CI;
}

// *************************************************************************

Matrix Crack(const Material &mate,const Vector &angleS,const Control &control,ofstream
&ouf,const Vector *StrsT,const Vector &BandAll, const int kk)
{
  Matrix Nhatran(3,2);
  Matrix Nhat(2,3);
  Matrix Ntran(2,3);
  Matrix CnC(2,2);
```

```
  Matrix Dcrack(3,3);
  Matrix Ststt(3,1);
  Matrix StrsTN(2,1);
  Real z;
  Nhat=0.;
  Nhatran=0.;
  Ntran=0.;
  int i,j,k;

  z=angleS(control.elmNum)*22/(7*180);
  Nhat(1,1)=cos(z)*cos(z); Nhat(1,2)=sin(z)*sin(z); Nhat(1,3)=cos(z)*sin(z);
  Nhat(2,1)=-2*sin(z)*cos(z); Nhat(2,2)=2*sin(z)*cos(z); Nhat(2,3)=cos(z)*cos(z)-sin
(z)*sin(z);
  Nhatran=Transpose(Nhat);

  Ntran(1,1)=cos(z)*cos(z); Ntran(1,2)=sin(z)*sin(z); Ntran(1,3)=2*cos(z)*sin(z);
  Ntran(2,1)=-sin(z)*cos(z); Ntran(2,2)=sin(z)*cos(z); Ntran(2,3)=cos(z)*cos(z)-sin
(z)*sin(z);

  for(i=1;i<=3;i++){
    Ststt(i,1)=StrsT[control.elmNum](i);
  }
  StrsTN=Ntran*Ststt;
  CnC(1,1)=-0.16667;
  CnC(2,2)=0.001;
  Dcrack=Nhatran*CnC*Nhat;

  return Dcrack;
}

// ************************************************************************

void Shape4(Matrix &SHP,Matrix &DSHP,Real &detJ,const Node *node,Real rg,Real sg)
{
  register int k,j,i;
  const Real R[]={0,1,1,-1,-1};
  const Real S[]={0,-1,1,1,-1};
  Matrix J(2,2);
  Matrix JINV(2,2);

  // Form the shape functions and their derivatives in natural coordinates.
  for(i=1;i<=4;i++){
    SHP(1,i)=0.25*(1+rg*R[i])*(1+sg*S[i]);
    DSHP(1,i)=0.25*(1+sg*S[i])*R[i];
    DSHP(2,i)=0.25*(1+rg*R[i])*S[i];
  }

  // Construct Jacobian Matrix.
  J=0.;
  for(i=1;i<=2;i++){
    for(j=1;j<=2;j++){
      for(k=1;k<=4;k++) J(i,j)+=DSHP(i,k)*node[k].Coor(j);
    }
  }

  detJ=J(1,1)*J(2,2)-J(1,2)*J(2,1);
  JINV(1,1)= J(2,2)/detJ;   JINV(1,2)=-J(1,2)/detJ;
  JINV(2,1)=-J(2,1)/detJ;   JINV(2,2)= J(1,1)/detJ;

  DSHP=JINV*DSHP;

}

// ************************************************************************

void Shp4(ofstream &ouf,Matrix &SHP,Matrix &DSHP,Real &detJ,const Vector
*nodeCoor,const Matrix &elmNode,int kk,Real rg,Real sg)
{
  register int k,j,i;
  const Real R[]={0,1,1,-1,-1};
  const Real S[]={0,-1,1,1,-1};
  Matrix J(2,2);
  Matrix JINV(2,2);

  // Form the shape functions and their derivatives in natural coordinates.
  for(i=1;i<=4;i++){
    SHP(1,i)=0.25*(1+rg*R[i])*(1+sg*S[i]);
```

```
    DSHP(1,i)=0.25*(1+sg*S[i])*R[i];
    DSHP(2,i)=0.25*(1+rg*R[i])*S[i];
  }

  // Construct Jacobian Matrix.
  J=0.;
  for(i=1;i<=2;i++){
    for(j=1;j<=2;j++){
      for(k=1;k<=4;k++) J(i,j)+=DSHP(i,k)*nodeCoor[(int) elmNode(kk,k)](j);
    }
  }

  detJ=J(1,1)*J(2,2)-J(1,2)*J(2,1);
  JINV(1,1)= J(2,2)/detJ;    JINV(1,2)=-J(1,2)/detJ;
  JINV(2,1)=-J(2,1)/detJ;    JINV(2,2)= J(1,1)/detJ;

  DSHP=JINV*DSHP;

}

// ***************************************************************************

Matrix BMatr(const Matrix &DSHP)
{
  register int node;
  int j;
  Matrix B(3,8);
  B=0.;
  for(node=1;node<=4;node++){
    j=(node-1)*2+1;
    B(1,j)=DSHP(1,node);
    B(2,j+1)=DSHP(2,node);
    B(3,j)=DSHP(2,node); B(3,j+1)=DSHP(1,node);
  }

  return B;
}

// ***************************************************************************

void PrtStrainStress(ofstream &ouf,const CoreData &core,const Control &control,
                     const Node *node,const Vector &Strn,const Vector &Strs)
{
  register int i;
  Real xc=0.,yc=0.;

  for(i=1;i<=4;i++){
    xc+=node[i].Coor(1);
    yc+=node[i].Coor(2);
  }
  xc/=4.;
  yc/=4.;

  if(control.elmNum==1){
    ouf<<"STRAIN STRESS::\n\n";
    for(i=1;i<=2;i++) ouf<<setw(10)<<i<<" COOR";
    for(i=1;i<=3;i++) ouf<<setw(10)<<i<<" STRN";
    ouf<<"\n";
    ouf<<"                                  ";
    for(i=1;i<=3;i++) ouf<<setw(10)<<i<<" STRS";
    ouf<<"\n\n";
  }

  ouf.precision(6);
  ouf.flags(FIX_POINT);
  ouf<<setw(3)<<control.elmNum<<setw(15)<<xc<<setw(15)<<yc;
  ouf.flags(SCIENTIC);
  for(i=1;i<=3;i++) ouf<<setw(15)<<Strn(i);
  ouf<<"\n";
  ouf<<"                                  ";
  for(i=1;i<=3;i++) ouf<<setw(15)<<Strs(i);
  ouf<<"\n";
  if(control.elmNum==core.nElem) ouf<<endl;

}
```

```
// ************************************************************************

Vector FindLambda(ofstream &ouf,Vector &Strs,const Vector *StrsT, const Vector &DStrs,
Real strsCr,const Control &control)
{
  register int i;
  Vector Lambda(2);
  Real a,b,c,d,e,f,g,A,B,C,D,aa,bb,cc,dd;

  Strs(1)=StrsT[control.elmNum](1);
  Strs(2)=StrsT[control.elmNum](2);
  Strs(3)=StrsT[control.elmNum](3);
  A = (Strs(1)+Strs(2))/2;
  B = (DStrs(1)+DStrs(2))/2;
  C = (Strs(1)-Strs(2))/2;
  D = (DStrs(1)-DStrs(2))/2;
  aa = (B*B)-(D*D)-(DStrs(3)*DStrs(3));
  bb = (-1)*2*((strsCr*B)-(A*B)+(C*D)+(Strs(3)*DStrs(3)));
  cc = (strsCr*strsCr)-(2*strsCr*A)+(A*A)-(C*C)-(Strs(3)*Strs(3));
  if(aa<1.e-15 && aa>-1.e-15){
    g=-(cc/bb);
    if(g>0){
      Lambda(1)=g;
      Lambda(2)=-1.e+10;}
    else{
      Lambda(1)=1e+10;
      Lambda(2)=g;}
  }
  else{
    dd = sqrt((bb*bb)-(4*aa*cc));
    e = (-bb+dd)/(2*aa);
    f = (-bb-dd)/(2*aa);
    Lambda(1)=Max(e,f);
    Lambda(2)=Min(e,f);}

  return Lambda;
}

// ************************************************************************

Vector AnalyseLambda(ofstream &ouf,Vector Lambda, Vector LambdaE, Real &lambdap,
                     Real &lambdan,const Control &control)
{
  Vector LambdaT(2);

  if(LambdaE(2)>0){
    lambdap=LambdaE(2);
    lambdan=-1.e30;
  }
  else if(LambdaE(1)<0){
    lambdap=1.e30;
    lambdan=LambdaE(1);
  }
  else{
    lambdap=LambdaE(1);
    lambdan=LambdaE(2);
  }

  if(lambdap<Lambda(1)){
    LambdaT(1)=lambdap;
  }
  else{
    LambdaT(1)=Lambda(1);
  }

  if(lambdan>Lambda(2)){
    LambdaT(2)=lambdan;
  }
  else{
    LambdaT(2)=Lambda(2);
  }

  return LambdaT;

}
```

```
// ***************************************************************************

Vector Bandwidth(const Material &mate,const Vector &angleS,const Node *node,const
Control &control,Element &elem,Real rg,Real sg)
{
  Real zeta,Xc,Yc,detJ;
  Vector X(4),Y(4),Xb(4),Yb(4),DNI(4),DNII(4);
  Vector f(4),coef(2),Xm(2),Ym(2);
  Matrix SHP(1,4);
  Matrix DSHP(3,4);
  register int i,j,k;
  const Real R[]={0,1,1,-1,-1};
  const Real S[]={0,-1,1,1,-1};
  Matrix J(2,2);
  Matrix JINV(2,2);

  zeta=angleS(control.elmNum)*22/(7*180);
  if(zeta<1e-10 && zeta>-1e-10){zeta=0;}
  for(i=1;i<=4;i++){
    X(i)=node[i].Coor(1);
    Y(i)=node[i].Coor(2);
  }
  for(i=1;i<=2;i++){
    j=2*i-1;
    k=j+1;
    Xm(i)=(X(j)+X(k))/2;
    Ym(i)=(Y(j)+Y(k))/2;
  }
  Xc=(Xm(1)+Xm(2))/2;
  Yc=(Ym(1)+Ym(2))/2;
  for(i=1;i<=4;i++){
    Xb(i)=X(i)-Xc;
    Yb(i)=Y(i)-Yc;
    if((Xb(i)*cos(zeta)+Yb(i)*sin(zeta))<0){f(i)=0;}
    else {f(i)=1;}
  }
  for(i=1;i<=4;i++){
    DSHP(1,i)=0.25*(1+sg*S[i])*R[i];
    DSHP(2,i)=0.25*(1+rg*R[i])*S[i];
  }
  J=0.;
  for(i=1;i<=2;i++){
    for(j=1;j<=2;j++){
      for(k=1;k<=4;k++) J(i,j)+=DSHP(i,k)*node[k].Coor(j);
    }
  }
  detJ=J(1,1)*J(2,2)-J(1,2)*J(2,1);
  JINV(1,1)= J(2,2)/detJ;    JINV(1,2)=-J(1,2)/detJ;
  JINV(2,1)=-J(2,1)/detJ;    JINV(2,2)= J(1,1)/detJ;
  DSHP=JINV*DSHP;
  coef(1)=0.;
  for(i=1;i<=4;i++){

    DNI(i)=DSHP(1,i);
    DNII(i)=DSHP(2,i);
    coef(1)+=(DNI(i)*cos(zeta)+DNII(i)*sin(zeta))*f(i);
  }
  coef(2)=detJ;

  return coef;
}

//***************************************************************************

void nrerror(char error_text[])

{
  cout<<endl<<"Numerical Recipes run-time error...."<<endl;
  cout<<error_text<<endl;
  cout<<"...now exit to system.."<<endl;
  exit(1);
}
```

```
// ***************************************************************************

void tred2(Matrix &a, int n, Vector &d, Vector &e)

{
  int i,j,k,l;
  Real scale,hh,h,g,f;

  for(i=n;i>=2;i--){
    l=i-1;
    h=scale=0.;
    if(l>1){
      for(k=1;k<=l;k++)
        scale+=fabs(a(i,k));
      if(scale==0.)
        e(i)=a(i,l);
      else{
        for(k=1;k<=l;k++){
          a(i,k)/=scale;
          h+=a(i,k)*a(i,k);
        }
        f=a(i,l);
        g=(f>=0.?-sqrt(h):sqrt(h));
        e(i)=scale*g;
        h-=f*g;
        a(i,l)=f-g;
        f=0.;
        for(j=1;j<=l;j++){
          a(j,i)=a(i,j)/h;
          g=0.;
          for(k=1;k<=j;k++)
            g+=a(j,k)*a(i,k);
          for(k=j+1;k<=l;k++)
            g+=a(k,j)*a(i,k);
          e(j)=g/h;
          f+=e(j)*a(i,j);
        }
        hh=f/(h+h);
        for(j=1;j<=l;j++){
          f=a(i,j);
          e(j)=g=e(j)-hh*f;
          for(k=1;k<=j;k++)
            a(j,k)-=(f*e(k)+g*a(i,k));
        }
      }
    }else
      e(i)=a(i,l);
    d(i)=h;
  }
  d(1)=0.;
  e(1)=0.;
  for(i=1;i<=n;i++){
    l=i-1;
    if(d(i)){
      for(j=1;j<=l;j++){
        g=0.0;
        for(k=1;k<=l;k++)
          g+=a(i,k)*a(k,j);
        for(k=1;k<=l;k++)
          a(k,j)-=g*a(k,i);
      }
    }
    d(i)=a(i,i);
    a(i,i)=1.0;
    for(j=1;j<=l;j++) a(j,i)=a(i,j)=0.;
  }
}

//***************************************************************************

void tqli(Vector &d, Vector &e, int n , Matrix &z)

{

  int m,l,iter,i,k;
  Real s,r,p,g,f,dd,c,b;
```

```
  for(i=2;i<=n;i++) e(i-1)=e(i);
  e(n)=0.;
  for(l=1;l<=n;l++){
    iter=0;
    do {
      for(m=l;m<=n-1;m++){
        dd=fabs(d(m))+fabs(d(m+1));
        if((Real)(fabs(e(m))+dd)==dd) break;
      }
      if(m!=l){
        if(iter++==30) nrerror("Too many iterations in tqli");
        g=(d(l+1)-d(l))/(2.0*e(l));
        r=pythag(g,1.0);
        g=d(m)-d(l)+e(l)/(g+SIGN(r,g));
        s=c=1.0;
        p=0.;
        for(i=m-1;i>=l;i--){
          f=s*e(i);
          b=c*e(i);
          e(i+1)=(r=pythag(f,g));
          if(r==0.0){
            d(i+1)-=p;
            e(m)=0.0;
            break;
          }
          s=f/r;
          c=g/r;
          g=d(i+1)-p;
          r=(d(i)-g)*s+2.0*c*b;
          d(i+1)=g+(p=s*r);
          g=c*r-b;
          for(k=1;k<=n;k++){
            f=z(k,i+1);
            z(k,i+1)=s*z(k,i)+c*f;
            z(k,i)=c*z(k,i)-s*f;
          }
        }
        if(r==0.0 && i>=l) continue;
        d(l)-=p;
        e(l)=g;
        e(m)=0.0;
      }
    }while(m!=l);
  }
}

//************************************************************************

Real pythag(Real a, Real b)

{
  Real absa,absb;
  absa=fabs(a);
  absb=fabs(b);
  if(absa>absb) return absa*sqrt(1.0+SQR(absb/absa));
  else return (absb == 0.0 ? 0.0 : absb*sqrt(1.0+SQR(absa/absb)));
}

// ************************************************************************

void PrtEigensystem(ofstream &ouf, int n, Vector &d, Matrix &a, Matrix &CndnGloK,
Vector &f)

{
  register int i,j,k;

  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++){
      f(j)=0.0;
      for(k=1;k<=n;k++)
        f(j)+=(CndnGloK(j,k)*a(k,i));
    }
    ouf.precision(6);
    ouf.flags(SCIENTIC);
    ouf<<endl<<setw(15)<<"Eigenvalue"<<setw(6)<<i<<setw(15)<<d(i)<<endl;

    ouf<<endl<<setw(20)<<"vector"<<setw(20)<<"mtrx*vect"<<setw(20)<<"ratio"<<endl;
```

```
    for(j=1;j<=n;j++){
      if(fabs(a(j,i))<le-6)
        ouf<<setw(20)<<a(j,i)<<setw(20)<<f(j)<<setw(20)<<"div. by 0"<<endl;
      else
        ouf<<setw(20)<<a(j,i)<<setw(20)<<f(j)<<setw(20)<<f(j)/a(j,i)<<endl;
    }
  }
}
```