

70-line 3D finite deformation elastoplastic finite-element code

William M. Coombs, Roger S. Crouch and Charles E. Augarde

School of Engineering and Computing Sciences, Durham University,

South Road, Durham. DH1 3LE. UK.

2010

Abstract

Few freeware FE programs offer the capabilities to include 3D finite deformation inelastic continuum analysis; those that do are typically expressed in tens of thousands of lines. This paper offers for the first time compact MATLAB scripts forming a complete finite deformation elasto-plastic FE program. The key modifications required to an infinitesimal FE program in order to include geometric non-linearity are described and the entire code given.

1 INTRODUCTION

The Finite-Element (FE) method has transformed geomechanical analysis. A number of open source codes now encourage researchers to extend or modify the basic algorithms, yet most 3D codes are expressed in tens of thousands of lines of C, C++ or fortran, requiring a significant time investment from potential new developers. Few freeware FE programs offer the capability to include 3D finite deformation analysis. Researchers are faced with writing their own algorithms from scratch or mastering very lengthy codes which are typically understandable only by those close to the original development. However, high level computational environments, such as MATLAB, allow engineers, scientists and mathematicians to produce powerful numerical analysis scripts rapidly. By using lean, efficient algorithms and subfunctions, it is possible to write the main routine of an elasto-plastic finite deformation FE program within a single page. Once a program spills onto multiple pages the ability to easily visualise the program structure is lost and the opportunity for error detection is reduced. Transparent programs facilitate re-analysis, adjustment, improvement and experimentation, resulting in polished robust algorithms. These programming ‘gems’ should be clear, easy to read, check, edit and modify. This strategy is inspired by Trefethen’s cry for cleaner, shorter code, within the philosophy of ‘one page, ten digit, five second’ algorithms (?).

A three dimensional MATLAB finite deformation FE code has been developed by the first author at Durham University, with the intention of analysing geotechnical problems subject to large deformations and strains. The program given here uses fully integrated 8-noded isoparametric hexahedral elements and a Prandtl-Reuss constitutive model, including the appropriate consistent tangent to ensure asymptotic quadratic convergence of the global Newton-Raphson iterations. The compact, modular algorithm allows alternative isotropic constitutive models (such as those based on a critical state) to be easily incorporated without modification to the overall program.

2 CONCEPTUAL MODEL

The following sections present the modifications to and equations required for an infinitesimal FE program in order to include geometric non-linearity. Section 2.2 describes the implemented combined force-displacement control FE program and should be read in conjunction with Algorithms I and II. All program segments are given, except the simple set-up file (line 3 of Algorithm I, highlighted in grey) which has been omitted for the sake of brevity. The overall program structure is summarised in Figure 1¹.

2.1 Modifications to infinitesimal theory

The following modifications are required when implementing an updated Lagrangian large strain FE code, compared with the equivalent infinitesimal linear elastic program

¹The numbers in Figure 1 refer to the MATLAB code lines in Algorithm I.

- The primary internal variable is the deformation gradient, $[F]$.
- The derivatives of the shape functions are calculated with respect to the updated nodal coordinates.
- The non-symmetric material spatial tangent modulus, $[a]$, and the full strain(9-component)–displacement matrix, $[G]$, are used to form the element stiffness matrix.
- An inelastic constitutive model is included.
- The global equilibrium equation is solved using the Newton–Raphson scheme.

Unlike infinitesimal theory, within a finite deformation framework there exists a choice for the stress and strain measures. However, certain combinations provide advantages when moving between infinitesimal and large strain theories. The implemented FE code uses a logarithmic strain–Kirchhoff stress relationship along with an exponential map for the plastic flow equation to allow the implementation of standard small strain constitutive algorithms within a finite deformation framework without modification². This implementation preserves the isochoric property of traceless strains and satisfies the incompressibility of J2 plastic flow theory exactly.

2.2 Non-linear FE code

The deformation gradient provides the fundamental link between the current and the reference configurations

$$[F] = \left[\frac{\partial \{x\}}{\partial \{X\}} \right] = \left[[1] + \frac{\partial \{u\}}{\partial \{X\}} \right], \quad (1)$$

where $\{x\}$ and $\{X\}$ are the coordinates of the same point in the current and reference configurations, respectively, $\{u\}$ is the displacement between the configurations and $[1]$ is the rank three identity matrix. Taking the polar decomposition of (1) we obtain

$$[F] = [v][R] \quad (2)$$

where $[R]$ is an orthogonal rotation matrix and $[v]$ is the left stretch matrix, given by

$$[v] = \sqrt{[b]} = \sqrt{[F][F]^T} \quad (3)$$

where $[b]$ is the left Cauchy–Green strain matrix. Note the square root of $[b]$ is obtained using spectral decomposition into principal values [see ?) for details] and using the inverse decomposition to recover the full six component symmetric matrix. We define the logarithmic strain measure as

$$[\varepsilon] = \ln[v] = \frac{1}{2} \ln[b], \quad (4)$$

where the logarithm of $[b]$ is obtained using spectral decomposition in a way analogous to the square root. Using the multiplicative decomposition of the deformation gradient, initially proposed by ?), into elastic and plastic components we can equivalently define the elastic logarithmic strain as

$$[\varepsilon^e] = \frac{1}{2} \ln[b^e], \quad (5)$$

where $[b^e]$ is the elastic left strain matrix. As the deformation proceeds within a boundary value problem, we update the deformation gradient via

$$[F_{n+1}] = [\Delta F][F_n], \quad (6)$$

where $[F_n]$ is the deformation gradient from the previous converged load step. The increment in the deformation gradient, for a given element nodal displacement increment, is given by

$$[\Delta F] = \left[[1] - \sum_{i=1}^n [\Delta u_i][N_{i,x}]^T \right]^{-1} \quad (7)$$

²See ?), amongst others, for more details on the recovery of the infinitesimal format of the stress return algorithms.

where $[\Delta u_i]$ are the nodal displacement increments and $[N_{i,x}]$ are the derivatives of the shape functions with respect to the updated nodal coordinates. From this deformation gradient increment, we can obtain the trial elastic left Cauchy-Green strain matrix

$$[b_{tr}^e] = [\Delta F] [b_n^e] [\Delta F]^T, \quad (8)$$

where $[b_n^e]$ is the left Cauchy-Green strain matrix from the previous converged load step. The trial elastic strain, $\{\varepsilon_{tr}^e\}$, is obtained from combining (8) with (5) and is subsequently used as the input into the small strain isotropic constitutive model (along with any internal variables). The constitutive model will return the update elastic logarithmic strain $\{\varepsilon^e\}$, internal variables and the Kirchhoff stress, defined as

$$\{\tau\} = J\{\sigma\}, \quad (9)$$

where J is the determinant of the deformation gradient and $\{\sigma\}$ is the true, or Cauchy, stress. The constitutive model should also return the small strain consistent (algorithmic) tangent stiffness matrix $[D^{alg}] = [\partial\tau/\partial\varepsilon_{tr}^e]$.

The above steps can be summarised into

1. Calculate the increment in the deformation gradient $[\Delta F]$ using (7), lines 48–50
2. Form the trial elastic left Cauchy-Green strain matrix $[b_{tr}^e]$ using (8) and calculate the trial elastic logarithmic strain $\{\varepsilon_{tr}^e\}$ from (5), lines 52–58
3. Use the standard small strain constitutive model (line 52) to return the updated elastic logarithmic strain $\{\varepsilon^e\}$, Kirchhoff stress $\{\tau\}$, internal variables and the consistent elasto-plastic tangent modulus $[D^{alg}] = [\partial\tau/\partial\varepsilon_{tr}^e]$
4. Calculate the updated true Cauchy stress $\{\sigma\}$ via the rearrangement of (9) using `formDsig.m`, line 53.

Once the Cauchy stress is updated, the element stiffness matrix (lines 23 and 61) is obtained from

$$[k^e] = \sum_{i=1}^{n_{gp}} [G_i]^T [a_i] [G_i] [J_i] w_i \quad (10)$$

where n_{gp} is the number of Gauss points, $[G]$ is the strain(9–component)–displacement matrix

$$[G] = \begin{bmatrix} N_{1,x} & 0 & 0 & \dots & 0 \\ 0 & N_{1,y} & 0 & \dots & 0 \\ 0 & 0 & N_{1,z} & \dots & N_{n,z} \\ N_{1,y} & 0 & 0 & \dots & 0 \\ 0 & N_{1,x} & 0 & \dots & 0 \\ 0 & N_{1,z} & 0 & \dots & 0 \\ 0 & 0 & N_{1,y} & \dots & N_{n,y} \\ 0 & 0 & N_{1,x} & \dots & N_{n,x} \\ N_{1,z} & 0 & 0 & \dots & 0 \end{bmatrix}. \quad (11)$$

$[J]$ is the Jacobian matrix obtained from the derivatives of the shape functions and the updated nodal coordinates and w is the weight function. $[a]$ is the consistent spatial tangent modulus, given by

$$[a] = \frac{1}{2J} [D^{alg}] [L] [B^a] - [S], \quad (12)$$

where

$$\begin{aligned} [D^{alg}] &= \frac{\partial\{\tau\}}{\partial\{\varepsilon_{tr}^e\}}, & [L] &= \frac{\partial \ln[b_{tr}^e]}{\partial [b_{tr}^e]}, \\ [B^a]_{ijkl} &= \delta_{ik} (b_{tr}^e)_{jl} + \delta_{jk} (b_{tr}^e)_{il}, \\ [S]_{ijkl} &= \sigma_{il} \delta_{jk}. \end{aligned}$$

$[D^{alg}]$ is the consistent tangent from the *unmodified* small strain constitutive model. $[L]$ is determined as a particular case of the derivative of a general symmetric second order tensor function with respect to its argument³. `formDsig.m`, line 60, returns the spatial tangent $[a]$ and $[L]$ is implemented in `parDerGen.m`.

The element internal forces (line 64) are then calculated using

$$\{f^{int}\} = \sum_{i=1}^{n_{gp}} [B_i]^T \{\sigma_i\} [J_i] w_i \quad (13)$$

where $[B]$ is the standard strain(6-component)–displacement matrix.

The preceding operations are performed for all Gauss points within each element over lines 41–66 in Algorithm I. The remaining FE implementation is summarised in Figure 1. The subfunctions are explained in Section 4.

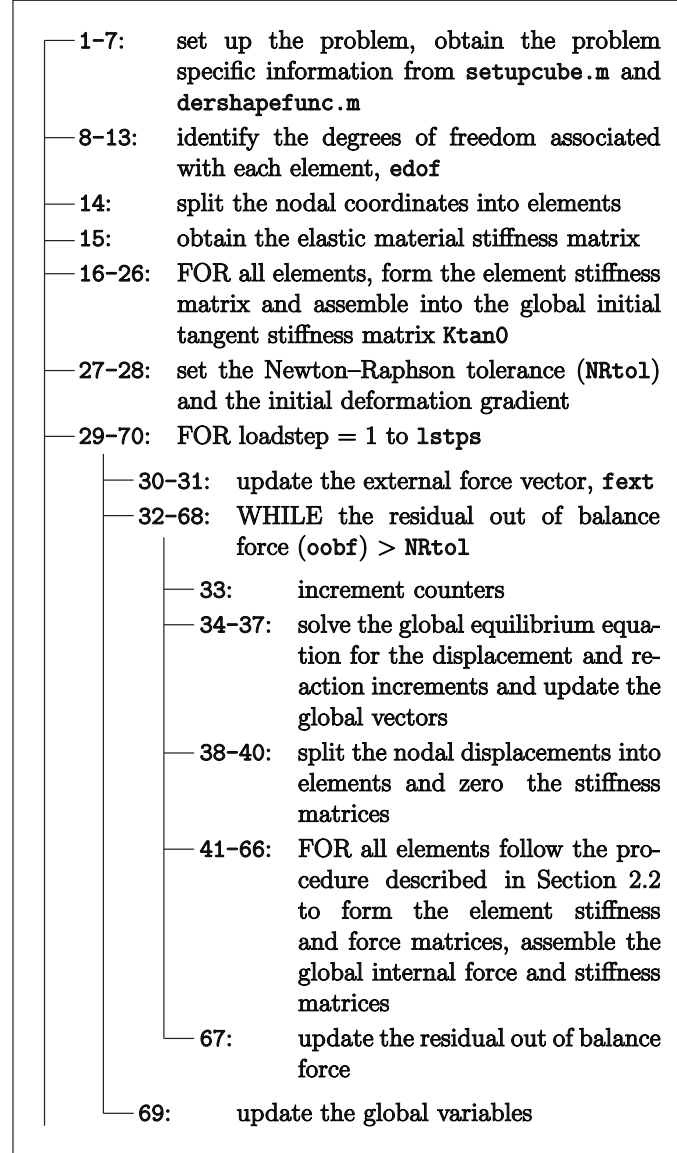


Figure 1: Main program structure

³Traditionally the derivative of a tensor function with respect to its argument has been solved by considering the spectral decomposition of the tensor function and using the product rule to obtain the derivative. However, calculation of the derivative in the case of repeated eigenvalues requires the use of eigen-projections to overcome the non-uniqueness of the eigenvalues. Refer to `parDerGen.m` and see ?), amongst others, for more details.

3 CONCLUSION

This paper offers for the first time the compact MATLAB scripts for a 3D finite deformation elasto–plastic FE program. The main code is only 70 lines in length, comfortably fitting on one page. The associated subfunctions are contained within an additional page⁴.

Through the use of an updated Lagrangian logarithmic strain–Kirchhoff stress formulation (with an exponential map for the plastic flow equation) the code allows for the incorporation existing isotropic small strain constitutive models without modification.

The extension of this code to include anisotropic constitutive models, based on the notion of a back–stress, is possible within 10 additional lines in the main program and making small extensions to the existing subfunctions.

4 PROGRAM NOTES

The main program and associated subfunctions are given in Algorithms I and II, respectively. Input parameters from `setupcube.m` are described in Table 1.

Item	Format	Description
nels	integer	No. elements
nodes	integer	No. nodes
nDoF	integer	No. degrees of freedom
coord	$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_{\text{nodes}} & y_{\text{nodes}} & z_{\text{nodes}} \end{bmatrix}$	Nodal coordinates
etopol	$\begin{bmatrix} 1 & n_1^1 & \dots & n_{\text{nen}}^1 \\ 2 & n_1^2 & \dots & n_{\text{nen}}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \text{nels} & n_1^{\text{nels}} & \dots & n_{\text{nen}}^{\text{nels}} \end{bmatrix}$	Element topology
fext0	$\begin{bmatrix} f_{ext}^1 \\ \vdots \\ f_{ext}^{\text{nDoF}} \end{bmatrix}$	Total external force vector to be applied in equal increments
bc	$\begin{bmatrix} i & u_i \end{bmatrix}$	Boundary conditions, displacement u_i for degree of freedom i
nen	integer	No. nodes per element
ngp	integer	No. Gauss points
lstps	integer	No. loadsteps
E	real	Young’s modulus
v	real	Poisson’s ratio
fc	real	Yield stress

Table 1: `setupcube.m` input parameters.

`dershapefunc.m`, line 4, returns the derivatives of the shape functions with respect to the element local coordinate system (`dNr`), the local coordinates (`xsi`, `eta`, `zet`) and the weights (`wp`) associated with the Gauss points for a fully integrated 8–noded isoparametric hexahedral element.

⁴The complete MATLAB .m files are available from the first author on request, email w.m.coombs@durham.ac.uk.

Xsplit.m, lines 14, 38 and 39, splits nodal coordinates (or displacements) into element contributions (ex, ey, ez), given the element degrees of freedom (edof), the nodal coordinates (coord), the degrees of freedom (DoF), the number of nodes per element (nen) and the number of elements (nels).

VMconst.m, lines 15 and 59, contains the constitutive model; in this case a Prandtl-Reuss (von Mises perfect plasticity) model that returns updated Kirchhoff stress (kirSig), the consistent tangent modulus (D) and elastic strain (epsE) for a given trial elastic strain (epsEtr).

formBG.m, lines 22 and 47, forms the strain(6 and 9-component)–displacement matrices, B and G, from the derivatives of the shape functions with respect to the updated nodal coordinates (dNx).

assem.m, lines 25 and 65, directs the contributions from the element stiffness matrices (ke) into the global stiffness matrix (Ktan).

solveq.m, lines 34 and 35, solves the global equilibrium equation $[K]\{\Delta d\} = \{\Delta f\}$ for the unknown incremental displacements and reactions (dduvw and dreact).

formDsig.m, line 60, forms the spatial tangent operator (a) and the updated Cauchy stress (sig).

parDerGen.m returns the partial derivative of a general symmetric second order tensor function with respect to its argument, L.

ACKNOWLEDGEMENT

The authors are most grateful for the support obtained from the UK EPSRC grant EP/D07711/01.

```

1 clear;
2 NRitmax=500000; tNRits=0; gNRits=0; NRtol=1e-12; bml=[1 1 1 0 0 0].';
3 [nels,nodes,nDoF,coord,etopol,fext0,bc,nen,ngp,lstps,E,v,fc]=setupcube;
4 [wp,xsi,eta,zet,dNr]=dershapefunc(ngp);
5 edof=zeros(nels,((nen*3)+1)); DoF=[(1:3:nDoF-2).'(2:3:nDoF-1).'(3:3:nDoF).'];
6 Ktan0=zeros(nDoF); fint=zeros(nDoF,1); rct=fint; uvwold=fint; uvw=fint;
7 epsEold=zeros(6,1,nels,ngp); epsE=epsEold; sig=epsE; Fold=zeros(3,3,nels,ngp);
8 for nel=1:nels
9     edof(nel,1)=nel;
10    for node=1:nen
11        edof(nel,(3*node)-1:(3*node)+1)=DoF(etopol(nel,node),:);
12    end
13 end
14 [ex,ey,ez]=Xsplt(edof,coord,DoF,nen,nels);
15 D=VMconst(zeros(6,1),E,v,fc);
16 for nel=1:nels
17     ke=zeros((nen*3),(nen*3));
18     JT=dNr*[ex(nel,:); ey(nel,:); ez(nel,:)].';
19     for gp=1:ngp
20         Fold(1:3,1:3,nel,gp)=eye(3);
21         indx=[(3*gp)-2:(3*gp)-1;(3*gp)];
22         [B]=formBG((JT(indx,:))\dNr(indx,:),nen);
23         ke=ke+(B'*D*B*det(JT(indx,:))*wp(gp));
24     end
25     Ktan0=assem(edof(nel,:),Ktan0,ke);
26 end
27 NRtol=max([max([abs(fext0); abs(Ktan0(bc(:,1),bc(:,1))*bc(:,2))])*NRtol 1e-6]);
28 k=0; F=Fold;
29 for lstp=1:lstps
30     fext=(lstp/lstps)*fext0;
31     oobf=rct+fext-fint; obfN=2*NRtol; NRit=0;
32     while ((NRit<NRitmax)&&(obfN>NRtol))
33         NRit=NRit+1; tNRits=tNRits+1; k=k+1; gNRits=max(gNRits,NRit);
34         if (NRit==1); [dduvw,dreact]=solveq(Ktan0,oobf,[bc(:,1) (1/lstps)*bc(:,2) ]);
35         else [dduvw,dreact]=solveq(Ktan ,oobf,[bc(:,1) zeros(size(bc,1),1)]);
36         end
37         uvw=uvw+dduvw; rct=rct+dreact; duvw=uvw-uvwold;
38         [eu,ev,ew]=Xsplt(edof,[ uvw(DoF(:,1)), uvw(DoF(:,2)), uvw(DoF(:,3))],DoF,nen,nels);
39         [du,dv,dw]=Xsplt(edof,[duvw(DoF(:,1)),duvw(DoF(:,2)),duvw(DoF(:,3))],DoF,nen,nels);
40         Ktan=zeros(nDoF); fint=zeros(nDoF,1); felem=zeros((nen*3),nels);
41         for nel=1:nels
42             JT=dNr*[ex(nel,:)+eu(nel,:); ey(nel,:)+ev(nel,:); ez(nel,:)+ew(nel,:)].';
43             ke=zeros((nen*3),(nen*3));
44             for gp=1:ngp
45                 indx=[(3*gp)-2:(3*gp)-1;3*gp];
46                 detJ=det(JT(indx,:)); dNx=(JT(indx,:))\dNr(indx,:);
47                 [B,G]=formBG(dNx,nen);
48                 ddF=zeros(3);
49                 for i=1:(size(dNx,2)); ddF=ddF+[du(nel,i);dv(nel,i);dw(nel,i)]*dNx(:,i).'; end
50                 dF=inv(eye(3)-ddF);
51                 F(:,,nel,gp)=dF*Fold(:,,nel,gp);
52                 e=epsEold(:,,nel,gp); eps=[e(1) e(4)/2 e(6)/2; e(4)/2 e(2) e(5)/2; e(6)/2 e(5)/2 e(3)];
53                 [V,ePr]=eig(eps); ePr=[ePr(1); ePr(5); ePr(9)]; Be=exp(2*ePr);
54                 Be=Be(1)*V(:,1)*V(:,1)'+Be(2)*V(:,2)*V(:,2)'+Be(3)*V(:,3)*V(:,3)';
55                 BeTr=dF*Be*dF';
56                 [V,BePr]=eig(BeTr); BePr=[BePr(1); BePr(5); BePr(9)]; BePr=0.5*log(BePr);
57                 etr=BePr(1)*V(:,1)*V(:,1)'+BePr(2)*V(:,2)*V(:,2)'+BePr(3)*V(:,3)*V(:,3)';
58                 epsEtr=[etr(1) etr(5) etr(9) 2*etr(2) 2*etr(6) 2*etr(7)'];
59                 [D,kirSig,epsE(:,,nel,gp)]=VMconst(epsEtr,E,v,fc);
60                 [a,sig(:,,nel,gp)]=formDsig(BeTr,kirSig,D,F(:,,nel,gp));
61                 ke=ke+(G'*a*G)*detJ*wp(gp);
62                 felem(:,nel)=felem(:,nel)+(B'*sig(:,,nel,gp)*detJ*wp(gp));
63             end
64             fint(edof(nel,2:(nen*3)+1))=fint(edof(nel,2:(nen*3)+1))+felem(:,nel);
65             Ktan=assem(edof(nel,:),Ktan,ke);
66         end
67         oobf=fext+rct-fint; obfN=sqrt(oobf'*oobf);
68     end
69     uvwold=uvw; epsEold=epsE; Fold=F;
70 end

```

Algorithm I: Main 3D finite deformation elasto–plastic FE code.

```

function [wp,xsi,eta,zet,dNr]=dershapefunc(ngp)
xsi=[-1 -1 1 1 -1 -1 1 1]'/sqrt(3); eta=[-1 -1 -1 -1 1 1 1 1]'/sqrt(3); zet=[-1 1 1 -1 -1 1 1 -1]'/sqrt(3);
wp=ones(8,1); r2=ngp*3;
dNr(1:3:r2,1)=-1/8*(1-eta).*(1-zet); dNr(2:3:r2+1,1)=-1/8*(1-xsi).*(1-zet); dNr(3:3:r2+2,1)=-1/8*(1-xsi).*(1-eta);
dNr(1:3:r2,2)=-1/8*(1-eta).*(1+zet); dNr(2:3:r2+1,2)=-1/8*(1-xsi).*(1+zet); dNr(3:3:r2+2,2)= 1/8*(1-xsi).*(1-eta);
dNr(1:3:r2,3)= 1/8*(1-eta).*(1+zet); dNr(2:3:r2+1,3)=-1/8*(1+xsi).*(1+zet); dNr(3:3:r2+2,3)= 1/8*(1+xsi).*(1-eta);
dNr(1:3:r2,4)= 1/8*(1-eta).*(1-zet); dNr(2:3:r2+1,4)=-1/8*(1+xsi).*(1-zet); dNr(3:3:r2+2,4)=-1/8*(1+xsi).*(1-eta);
dNr(1:3:r2,5)=-1/8*(1+eta).*(1-zet); dNr(2:3:r2+1,5)= 1/8*(1-xsi).*(1-zet); dNr(3:3:r2+2,5)=-1/8*(1-xsi).*(1+eta);
dNr(1:3:r2,6)=-1/8*(1+eta).*(1+zet); dNr(2:3:r2+1,6)= 1/8*(1-xsi).*(1+zet); dNr(3:3:r2+2,6)= 1/8*(1-xsi).*(1+eta);
dNr(1:3:r2,7)= 1/8*(1+eta).*(1+zet); dNr(2:3:r2+1,7)= 1/8*(1+xsi).*(1+zet); dNr(3:3:r2+2,7)= 1/8*(1+xsi).*(1+eta);
dNr(1:3:r2,8)= 1/8*(1+eta).*(1-zet); dNr(2:3:r2+1,8)= 1/8*(1+xsi).*(1-zet); dNr(3:3:r2+2,8)=-1/8*(1+xsi).*(1+eta);

function [K]=assem(edof,K,Ke)
[nie,n]=size(edof); t=edof(:,2:n);
for i=1:nie; K(t(i,:),t(i,:))=K(t(i,:),t(i,:))+Ke; end

function [d,R]=solveq(K,f,bc)
fdof=[1:size(K)']; d=zeros(size(fdof)); fdof(bc(:,1))=[];
s=K(fdof,fdof)\(f(fdof)-K(fdof,bc(:,1))*bc(:,2));
d(bc(:,1))=bc(:,2); d(fdof)=s; R=K*d-f;

function [ex,ey,ez]=Xsplt(edof,coord,DoF,nen,nel)
nn=zeros(1,nen); ex=zeros(nel,nen); ey=ex; ez=ex;
for i=1:nel
    for j=1:nen;
        nn(j)=find((DoF(:,1)-edof(i,j*3-1))==0);
    end
    ex(i,:)=coord(nn,1)'; ey(i,:)=coord(nn,2)'; ez(i,:)=coord(nn,3)';
end

function [Dalg,sigma,epsE]=VMconst(epsEtr,E,v,fc)
tol=1e-15; maxit=25; bm1=[1 1 1 0 0 0]';
Ce=[-ones(3)*v+(1+v)*eye(3) zeros(3); zeros(3) 2*(1+v)*eye(3)]/E; De=inv(Ce);
sigma=De*epsEtr; s=sigma-sum(sigma(1:3))/3*bm1; j2=(s'*s+s(4:6)'*s(4:6))/2;
f=sqrt(3*j2)/fc-1; epsE=epsEtr; Dalg=De;
if (f>tol)
    b=zeros(7,1); b(7)=f; itnum=0; dgam=0;
    dj2=s; dj2(4:6)=2*dj2(4:6); ddj2=[eye(3)-ones(3)/3 zeros(3); zeros(3) 2*eye(3)];
    df=sqrt(3)/(2*fc*sqrt(j2))*dj2; ddf=sqrt(3)/2/fc*(-dj2*dj2'/(2*j2^(3/2))+ddj2/sqrt(j2));
    while (itnum<maxit) && ((norm(b(1:6))>tol) || (abs(b(7))>tol)); itnum=itnum+1;
    A=[eye(6)+dgam*ddf*De df; df'*De 0];
    dx=-inv(A)*b; epsE=epsE+dx(1:6); dgam=dgam+dx(7);
    sigma=De*epsE; s=sigma-sum(sigma(1:3))/3*bm1; j2=(s'*s+s(4:6)'*s(4:6))/2;
    dj2=s; dj2(4:6)=2*dj2(4:6); ddj2=[eye(3)-ones(3)/3 zeros(3); zeros(3) 2*eye(3)];
    df=sqrt(3)/(2*fc*sqrt(j2))*dj2; ddf=sqrt(3)/2/fc*(-dj2*dj2'/(2*j2^(3/2))+ddj2/sqrt(j2));
    b=[epsE-epsEtr+dgam*df; sqrt(3*j2)/fc-1];
    end
    B=inv([Ce eye(6); dgam*ddf -eye(6)]);
    Dalg=B(1:6,1:6)-B(1:6,7:12)*(df*df')*B(1:6,1:6)/([df' zeros(1,6)]*B*[zeros(6,1); df]);
end

function [B,G]=formBG(dNx,nen)
B=zeros(6,nen*3);
B(1,1:3:end)=dNx(1,:);
B(2,2:3:end)=dNx(2,:);
B(3,3:3:end)=dNx(3,:);
B(4,1:3:end)=dNx(2,:);
B(4,2:3:end)=dNx(1,:);
B(5,2:3:end)=dNx(3,:);
B(5,3:3:end)=dNx(2,:);
B(6,1:3:end)=dNx(3,:);
B(6,3:3:end)=dNx(1,:);
G=zeros(9,nen*3);
G(1:3,:)=B(1:3,:);
G(4,1:3:end)=dNx(2,:);
G(5,2:3:end)=dNx(1,:);
G(6,2:3:end)=dNx(3,:);
G(7,3:3:end)=dNx(2,:);
G(8,3:3:end)=dNx(1,:);
G(9,1:3:end)=dNx(3,:);

function [a,sig]=formDsig(Be,kirSig,D,F)
sig=1/det(F)*kirSig;
[BeVec,BePr]=eig(Be); BePr=[BePr(1) BePr(5) BePr(9)]';
L=parDerGen(Be,BeVec,BePr,log(BePr),1./BePr);
S=zeros(9); B=zeros(9); B(1,[1 4 9])=2*Be([1 2 3]);
B(2,[2 5 6])=2*Be([5 4 6]); B(3,[3 7 8])=2*Be([9 8 7]);
B(4,[1 2 4 5 6 9])=Be([4 2 5 1 3 6]); B(5,:)=B(4,:);
B(6,[2 3 5 6 7 8])=Be([8 6 7 9 5 4]); B(7,:)=B(6,:);
B(8,[1 3 4 7 8 9])=Be([7 3 8 2 1 9]); B(9,:)=B(8,:);
S(1,[1 4 9])=sig([1 4 6]); S(2,[2 5 6])=sig([2 4 5]);
S(3,[3 7 8])=sig([3 5 6]); S(4,[2 5 6])=sig([4 1 6]);
S(5,[1 4 9])=sig([4 2 5]); S(6,[3 7 8])=sig([5 2 4]);
S(7,[2 5 6])=sig([5 6 3]); S(8,[1 4 9])=sig([6 5 3]);
S(9,[3 7 8])=sig([6 4 1]);
D9=[D(1:3,1:3) D(1:3,[4 4 5 5 6 6]);
     D([4 4 5 5 6 6],1:3) D([4 4 5 5 6 6],[4 4 5 5 6 6])];
a=1/(2*det(F))*D9*L*B-S;

function [L]=parDerGen(X,eV,eP,yP,yd)
tol=1e-9; s=zeros(5,1); Is=[eye(3) zeros(3); zeros(3) eye(3)/2]; bm1=[1 1 1 0 0 0]';
if (abs(eP(1))<tol && abs(eP(2))<tol && abs(eP(3))<tol); L=Is;
elseif abs(eP(1)-eP(2))<tol && abs(eP(1)-eP(3))<tol; L=yd(1)*Is;
elseif abs(eP(1)-eP(2))<tol || abs(eP(2)-eP(3))<tol || abs(eP(1)-eP(3))<tol
    x=[X(1) X(5) X(9) X(2) X(6) X(7)]';
    s(1,1)=(yP(1)-yP(3))/(eP(1)-eP(3))-2*(yd(3))/(eP(1)-eP(3));
    s(2,1)=2*eP(3)*(yP(1)-yP(3))/(eP(1)-eP(3))-2*(eP(1)+eP(3))/(eP(1)-eP(3))*yd(3);
    s(3,1)=2*(yP(1)-yP(3))/(eP(1)-eP(3))-3*(yd(1)+yd(3))/(eP(1)-eP(3))-2;
    s(4,1)=eP(3)*s(3,1); s(5,1)=eP(3)^2*s(3,1);
    dX2dX=[2*X(1) 0 0 X(2) 0 X(3); 0 2*X(5) 0 X(2) X(6) 0; 0 0 2*X(9) 0 X(6) X(3);
           X(2) X(2) 0 (X(1)+X(5))/2 X(3)/2 X(6)/2; 0 X(6) X(6) X(3)/2 (X(5)+X(9))/2 X(2)/2;
           X(3) 0 X(3) X(6)/2 X(2)/2 (X(1)+X(9))/2];
    L=s(1,1)*dX2dX-s(2,1)*Is-s(3,1)*(x*x')+(s(4,1)*(x*bm1'+bm1*x')-s(5,1)*(bm1*bm1'));
else
    D=[(eP(1)-eP(2))*(eP(1)-eP(3)); (eP(2)-eP(1))*(eP(2)-eP(3)); (eP(3)-eP(1))*(eP(3)-eP(2))];
    alfa=0; bta=0; g=zeros(3,1); eD=zeros(6,3);
    for i=1:3; alfa=alfa+yP(i)*eP(i)/D(i); bta=bta+yP(i)/D(i)*det(X);
        for j=1:3; g(i)=g(i)+yP(j)*eP(j)/D(j)*(det(X)/eP(j)-eP(i)^2)/eP(i)^2; end
        esq=eV(:,i)*eV(:,i)'; eD(:,i)=[esq(1,1) esq(2,2) esq(3,3) esq(1,2) esq(2,3) esq(3,1)]';
    end
    y=inv(X);
    Ib=[y(1)^2 y(2)^2 y(7)^2 y(1)*y(2) y(2)*y(7) y(1)*y(7); y(2)^2 y(5)^2 y(6)^2 y(5)*y(2) y(5)*y(6) y(2)*y(6);
        y(7)^2 y(6)^2 y(9)^2 y(6)*y(7) y(9)*y(6) y(9)*y(7);
        y(1)*y(2) y(5)*y(2) y(6)*y(7) (y(1)*y(5)+y(2)^2)/2 (y(2)*y(6)+y(5)*y(7))/2 (y(1)*y(6)+y(2)*y(7))/2;
        y(2)*y(7) y(5)*y(6) y(9)*y(6) (y(2)*y(6)+y(5)*y(7))/2 (y(9)*y(5)+y(6)^2)/2 (y(9)*y(2)+y(6)*y(7))/2;
        y(1)*y(7) y(2)*y(6) y(9)*y(7) (y(1)*y(6)+y(2)*y(7))/2 (y(9)*y(2)+y(6)*y(7))/2 (y(9)*y(1)+y(7)^2)/2];
    L=alfa*Is-bta*Ib+(yd(1)+g(1))*eD(:,1)*eD(:,1)'+(yd(2)+g(2))*eD(:,2)*eD(:,2)'+(yd(3)+g(3))*eD(:,3)*eD(:,3)';
end
L=[L(1:3,1:3) L(1:3,[4 4 5 5 6 6]); L([4 4 5 5 6 6],1:3) L([4 4 5 5 6 6],[4 4 5 5 6 6])];

```

Algorithm II: Subfunctions: dershapefunc.m, assem.m, solveq.m, Xsplt.m, formDsig.m, VMconst.m, formBG.m and parDerGen.m