6th European Conference on Computational Mechanics (ECCM 6)
7th European Conference on Computational Fluid Dynamics (ECFD 7)
1115 June 2018, Glasgow, UK

A COMPACT GEOMETRICALLY NONLINEAR FE SHELL CODE FOR CARDIAC ANALYSIS: SHELL FORMULATION

YEW YAN WONG AND ROGER S CROUCH

City, University of London
10 Northampton Square, EC1V 8EN
Yew.Wong.1@city.ac.uk

Key words: Cardiac Analysis, Shell Finite Elements, Large Deformations

Abstract. There are few free open-source FE programs for 3D geometrically nonlinear shell elements that allow users to modify or extend the code. This paper presents such a MATLAB code which draws heavily on the work of Coombs[1] and Coombs *et al*[2].

1 INTRODUCTION

Cardiovascular disease is the largest cause of death in humans and account for 45% of all loss of life in Europe in 2017[3]; with cardiac arrhythmias being the most troublesome medical condition. The cause of cardiac arrhythmias is still not that well understood. Clinical studies are impeded by invasive and expensive in-vivo cardiac experimentation[4,5]. As an alternative approach, mathematical modelling[6,7,8] coupled with the advancement in high-resolution MRI scans are increasingly being used to assist in the study of cardiac behaviour[9,10,11,12]. However, these simulations can be computationally expensive. Considerable effort has been put into trying to optimise analyses by taking advantage of parallel processors[4,5]. Yet a recent analysis with 60 million degrees of freedom still required 50 minutes of run time to compute a single cardiac cycle with 127 Xeon processor cores when using conventional 3D solid finite elements[12].

Lack of open source codes hinders research development in the field. The study described here makes use of shell finite elements which reduces the requirement for many hexahedra or tetrahedral elements through the thickness of the heart muscle walls. In this paper, a compact three dimensional MATLAB code for *Total-Lagrangian Finite Element Analyses* (FEA) using 9-noded shell elements, is presented. Code performance is compared here against hexahedral[2] elements using a cantilever subjected to a transverse end load and an end moment. The shell elements are currently being extended to introduce a layered anisotropic formulation such that the different zones through the heart wall can be represented.

2 METHODS

2.1 Electro-mechanical coupling

Cardiac muscle tissue is highly orientated, with muscle fibres aligned in near-parallel bundles. These fibres contract and relax following changes in the electrical membrane potential. The latter can be modelled by two different means; (i) use of local *ionic concentration* models, which account for the intracellular movement of ions or (ii) phenomenological models which duplicate the membrane potential behaviour seen in *electrocardiogram* (ECG) scans. The second approach allows for significant savings on computational time. A one-dimensional coupled electro-mechanical *finite difference* code that propagates the membrane potential through a *monodomain* model has already been constructed by the first author. This is currently being extended to the shell element formulation.

2.2 Formulation of the Total-Lagrangian approach for shell elements

The key concept of the *continuum based degenerated shell finite element* formulation is to capture information of the through-thickness bending behaviour by the intrudction of rotational degree of freedoms. The 3D coordinate field is given as follows[14]:

$$\left\{{}^{t}x\right\} = \sum_{k=1}^{N} h^k \left\{{}^{t}x^k\right\} + \frac{\zeta}{2}\sum_{k=1}^{N} a^k h^k \left\{{}^{t}V_n^k\right\} \tag{1}$$

where ζ gives the local coordinates of the $z-$axis, $\{{}^{t}x^k\}$, h^k, a^k and $\{{}^{t}V_n^k\}$ is the coordinates, *shape functions* (made-up of membrane local coordinates ξ and η), the thickness of the shell and the thickness direction vectors at node k (N being the total number of nodes in the elements) at time t respectively. Two different angles (θ_x and θ_y) are used to identify the initial thickness direction vector with respect to the global $x-$axis and $y-$axis at each node. The initial thickness direction vector can then be expressed as:

$$\left\{{}^{0}V_n^k\right\} = \begin{Bmatrix} \cos(\theta_x^k) \\ \sin(\theta_x^k)\cos(\theta_y^k) \\ \sin(\theta_x^k)\sin(\theta_y^k) \end{Bmatrix} \tag{2}$$

The displacements are given by $\{u\} = \{{}^{t+\Delta t}x\} - \{{}^{t}x\}$. The thickness direction vector can be expressed in terms of a rotation about the $x-$axis α, and a rotation about the $y-$axis, β as $\{{}^{t}V_n^k\} = -\{{}^{t}V_2^k\}\alpha^k + \{{}^{t}V_1^k\}\beta^k$. Thus, the displacement becomes:

$$\left\{{}^{t}u\right\} = \sum_{k=1}^{N} h^k \left\{{}^{t}u^k\right\} + \frac{\zeta}{2}\sum_{k=1}^{N} a^k h^k(-\alpha^k\{{}^{t}V_2^k\} + \beta^k\{{}^{t}V_1^k\}) \tag{3}$$

When large deformation and rotation arise in a *geometrically nonlinear* FEA, the *constitutive matrix* $[D]$, is updated in order to follow the deformed configuration by means of a rotation matrix $[Q]$, which is recorded in equations (5.119) and (5.120) in the work by Bathe[15].

The strain-displacement matrix can then be formed as follows:

$$\begin{Bmatrix} u_{i,{}^t x_1} \\ u_{i,{}^t x_2} \\ u_{i,{}^t x_3} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u_i}{\partial {}^t x_1} \\ \frac{\partial u_i}{\partial {}^t x_2} \\ \frac{\partial u_i}{\partial {}^t x_3} \end{Bmatrix} = \sum_{k=1}^{N} \begin{bmatrix} {}^t h_{k,1} & {}^t g_{1it}^k G_1^k & {}^t g_{2it}^k G_1^k \\ {}^t h_{k,2} & {}^t g_{1it}^k G_2^k & {}^t g_{2it}^k G_2^k \\ {}^t h_{k,3} & {}^t g_{1it}^k G_3^k & {}^t g_{2it}^k G_3^k \end{bmatrix} \begin{Bmatrix} u_i^k \\ \alpha^k \\ \beta^k \end{Bmatrix} \tag{4}$$

where $\{{}_t h_k\}$, $\{g_1^k\}$ and $\{g_2^k\}$ and $\{{}_t G^k\}$ are defined below.

$$\begin{aligned} \left\{{}^t h_k\right\} &= \begin{Bmatrix} ({}^t J^{-1})_{11} \frac{dN^k}{d\xi} + ({}^t J^{-1})_{12} \frac{dN^k}{d\eta} \\ ({}^t J^{-1})_{21} \frac{dN^k}{d\xi} + ({}^t J^{-1})_{22} \frac{dN^k}{d\eta} \\ ({}^t J^{-1})_{31} \frac{dN^k}{d\xi} + ({}^t J^{-1})_{32} \frac{dN^k}{d\eta} \end{Bmatrix} \\ \left\{{}^t g_1^k\right\} &= -\frac{1}{2} a^k \left\{{}^t V_2^k\right\} \qquad\qquad \left\{{}^t g_2^k\right\} = \frac{1}{2} a^k \left\{{}^t V_1^k\right\} \\ \left\{{}^t G^k\right\} &= t \begin{Bmatrix} \zeta\left(({}^t J^{-1})_{11} \frac{dN^k}{d\xi} + ({}^t J^{-1})_{12} \frac{dN^k}{d\eta}\right) + ({}^t J^{-1})_{13} N^k \\ \zeta\left(({}^t J^{-1})_{21} \frac{dN^k}{d\xi} + ({}^t J^{-1})_{22} \frac{dN^k}{d\eta}\right) + ({}^t J^{-1})_{23} N^k \\ \zeta\left(({}^t J^{-1})_{31} \frac{dN^k}{d\xi} + ({}^t J^{-1})_{32} \frac{dN^k}{d\eta}\right) + ({}^t J^{-1})_{33} N^k \end{Bmatrix} \end{aligned} \tag{5}$$

In *Total-Lagrangian* formulations, the *Green-Lagrange* strain can be divided into linear and nonlinear components as shown in equation (19) of Bathe *et al*[16]. To take into account of this, the strain-displacement matrix is divided into a linear component $[B_L]$, and a nonlinear component $[B_{NL}]$, which can both be found in Table 4 of Bathe and Bolourchi[14] (as ${}^t_0\mathbf{B}_{L0}$ and ${}^t_0\mathbf{B}_{L1}$ respectively). However, the nonlinear strain-displacement matrix work reported here is computed by obtaining the product of an auxiliary matrix $[A]$ (expressed in equation (7.24) of Stegmann[16]) and the nonlinear strain-displacement transformation matrix, $[G_{NL}]$ known as ${}^t_0\mathbf{B}_{NL}$ by Bathe and Bolourchi[14] in Table 4 or $[\mathbf{G}]$ in equation (7.26) by Stegmann[16].

The *Green-Lagrange* strain $\{\epsilon\}$, can then be determined as $\{\epsilon\} = \{{}^n\epsilon\} + \{{}^t\epsilon\}$, where $\{{}^n\epsilon\}$ is the strain state corresponding to the previous iteration and $\{{}^t\epsilon\}$ is the strain increment. The strain increment can determined as shown in Table 6.2 (as ${}_0e_{ij}$) by Bathe[14]. The second-Piola Kirchoff stress can now determined from as $\{\sigma\} = \{{}^n\sigma\} + [D_{sh}]\{{}^t\epsilon\}$, where $[D_{sh}]$ is the constitutive matrix of the deformed configuration[13].

Having formed the strain-displacement matrix, $[B]$, the constitutive matrix, $[D_{sh}]$ and the nonlinear strain-displacement transformation matrix, $[G_{NL}]$, the tangent stiffness matrix, $[K_e]$ can now be computed as:

$$\left[K_e\right] = \sum_{i=1}^{nGp} \Big([B]^T [D_{sh}][B] + [G_{NL}]^T [H][G_{NL}] \Big) |J| w_i \tag{6}$$

where $|J|$ is the determinant of the *Jacobian* and w_i is the *Gauss weighting* for Gauss point i. The nodal internal force can now be determined as:

$$\left\{f_{int}\right\} = \sum_{i=1}^{nGp} \left[B\right]^T \left\{\sigma\right\} \left|J\right| w_i \tag{7}$$

The difference between the applied external force and the internal force from the internal stresses is then calculated as:

$$\{f_{oob}\} = \{f_{ext}\} + \{f_{rct}\} - \{f_{int}\} \tag{8}$$

If this out-of-balance force is within a pre-defined tolerance, then that load step is deemed to have converged and the analysis moves on to the next load step. However, if it is not within the tolerance, the displacement is updated by determining the displacement variation $\{\delta u\}$ by making use of *Newton Raphson* iterations with a new tangent stiffness matrix and the out-of-balance force.

$$[K]\{\delta u\} = \{f_{oob}\} \tag{9}$$

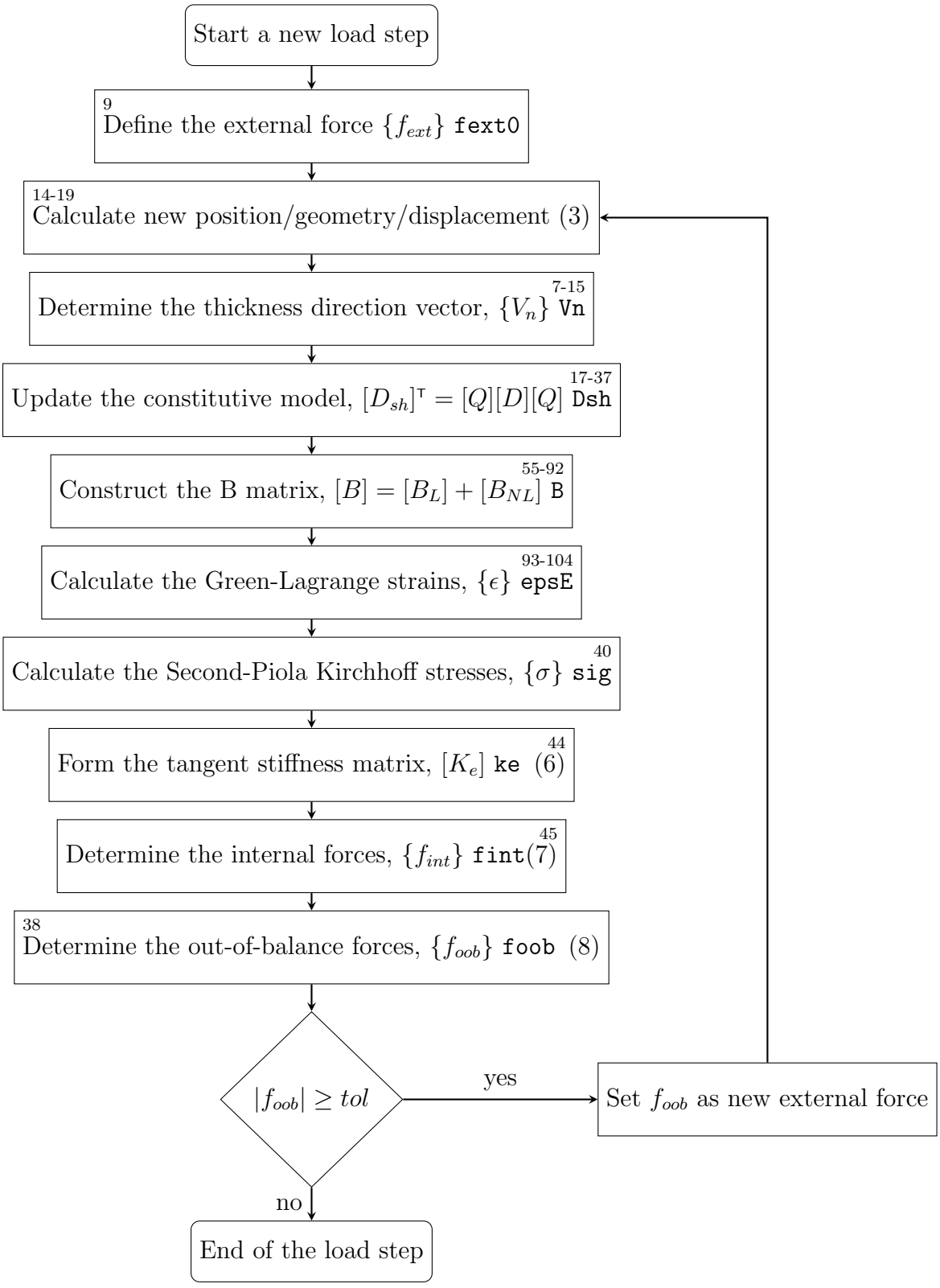
The procedure of the FEA and the variable update sequence is summarised in Figure 1. The number in the top left of each process refers to the line number of the main MATLAB script (Listing 1) whereas the number in the top right refers to the line number of the MATLAB function files (Listing 2).

The parameters required to define the problem and the initial geometry needs to be stored in a M-script *inputfile.m* with the parameters explained in Table 1 below.

Table 1: inputfile.m parameters

Notation	Format	Description
`coord`	$\begin{bmatrix} x_i & y_i & z_i & \theta_{ix} & \theta_{iy} & t \end{bmatrix}$	Nodal coordinates, initial thickness rotation angles and shell thickness
`etpl`	$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$	Element topology
`fext0`	$\left\{ f_{ext}^{1} \cdots f_{ext}^{nDOF} \right\}^{\mathsf{T}}$	Total applied external force
`bc`	$\begin{bmatrix} i & u_i \end{bmatrix}$	Boundary conditions: degrees of freedom i having defined displacements of u_i
`lstps`	integer scalar	Number of load steps
`NRitmax`	integer scalar	Maximum number of iterations per load step
`NRtol`	real scalar	Out-of-balance force tolerance for convergence
`E`	real scalar	Young's modulus (Pascal)
`nu`	real scalareger	Poisson's ratio
`ka`	real scalar	Shear factor

Figure 1: Finite element procedure



```
[coord,etpl,fext0,bc,lstps,NRitmax,NRtol,E,nu,ka]=inputfile;
nels=size(etpl,1); nodes=size(coord,1); nDoF=nodes*5;
neDoF=(9*5)^2; krow=zeros(neDoF*nels,1); kcol=krow; kval=krow;
uvw=zeros(nDoF,1); uvwold=uvw; fint=uvw; react=uvw;
fd=(1:nDoF); fd(bc(:,1))=[];
epsEn=zeros(6,8,nels); epsE=epsEn; sigN=epsEn; sig=epsEn;
Vn=zeros(9,3,nels); oVn=Vn; oL=zeros(8,9,nels); L=oL;
for lstp=0:lstps
  fext=(lstp/lstps)*fext0; foob=react+fext-fint;
  foobnorm=2*NRtol; NRit=0;
  while ((NRit<NRitmax)&&(foobnorm>NRtol))
    NRit=NRit+1; fint=zeros(nDoF,1); dreact=fint; dduvw=fint;
    if lstp>=1
      Kt=sparse(krow,kcol,kval,nDoF,nDoF);
      dduvw(bc(:,1))=(1+sign(1-NRit))*bc(:,2)/lstps;
      dduvw(fd)=Kt(fd,fd)\(foob(fd)-Kt(fd,bc(:,1))*dduvw(bc(:,1)));
      dreact(bc(:,1))=Kt(bc(:,1),:)*dduvw-foob(bc(:,1));
    end
    uvw=uvw+dduvw; react=react+dreact; duvw=uvw-uvwold;
    for nel=1:nels
      ed=reshape(ones(5,1)*etpl(nel,:)*5-(5-1:-1:0).'*ones(1,9),1,9*5);
      if lstp==0
        elcoord=coord(etpl(nel,:),:);
        phi=elcoord(:,4); psi=elcoord(:,5);
        oVn(:,:,nel)=[cos(psi) sin(psi).*cos(phi) sin(psi).*sin(phi)];
      end
      [ke,felem,epsE(:,:,nel),Vn(:,:,nel),sig(:,:,nel),L(:,:,nel)]=...
      shell(coord(etpl(nel,:),:),uvw(ed),duvw(ed),epsEn(:,:,nel),...
            oVn(:,:,nel),E,nu,ka,sigN(:,:,nel),oL(:,:,nel));
      if lstp==0
        ct=(nel-1)*neDoF+1:nel*neDoF;
        krow(ct)=reshape(ed.'*ones(1,9*5),neDoF,1);
        kcol(ct)=reshape(ones(9*5,1)*ed  ,neDoF,1);
      end
      kval((nel-1)*neDoF+1:nel*neDoF)=reshape(ke,neDoF,1);
      fint(ed)=fint(ed)+felem;
    end
    foob=fext+react-fint; foobnorm=norm(foob)/norm(fext+react+eps);
    fprintf('%4i %4i %6.3e\n',lstp,NRit,foobnorm);
  end
  uvwold=uvw; epsEn=epsE; sigN=sig; oL=oL+L;
end
```

Listing 1: Main script

```
function [ke,fint,epsE,Vn,sig,L]=shell(nodeData,uvw,duvw,epsEn,oVn,...
                                       E,nu,ka,sigN,oL)
coord=nodeData(:,1:3); t=nodeData(:,6); epsE=zeros(6,8);
ke=zeros(9*5); fint=zeros(9*5,1); dxr=zeros(3); [wp,GpLoc]=GpPos();
ex=[1 0 0].'; ey=[0 1 0].'; ez=[0 0 1].';
V1=zeros(9,3); V2=V1; g1=V1; g2=V1;
xsi=[-1; -1; -1; 0; 1; 1; 1; 0; 0]; eta=[-1; 0; 1; 1; 1; 0; -1; -1; 0];
dNr=dershapefunc(xsi,eta); dnr=dNr(1:2:end,:); dns=dNr(2:2:end,:);
dsp=reshape(uvw-duvw,5,[])'; elcoord=coord+dsp(:,1:3);
for n=1:9
  Vn(n,:)=cross((dnr(n,:)*elcoord)/norm(dnr(n,:)*elcoord),...
                (dns(n,:)*elcoord)/norm(dns(n,:)*elcoord));
  V=cross(ey,Vn(n,:).').';
  V1(n,:)=V/norm(V); V2(n,:)=cross(Vn(n,:).',V1(n,:).').';
  g1(n,:)=-0.5*t(n)*V2(n,:); g2(n,:)= 0.5*t(n)*V1(n,:);
end
D=[[1 nu; nu 1; 0 0] zeros(3,4); zeros(3) eye(3)*ka*(1-nu)*0.5];
D=(E/(1-nu^2))*D; D(4,4)=D(4,4)/ka;
for Gp=1:8
  xsi=GpLoc(Gp,1); eta=GpLoc(Gp,2); zet=GpLoc(Gp,3);
  N=shapefunc(xsi,eta); dNr=dershapefunc(xsi,eta);
  dxr(1:2,:)=(dNr*(coord+0.5*zet*(oVn.*(t*ones(1,3)))));
  dxr(3,:)=(0.5*(N.*t')*oVn); VnGp=(N*oVn).';
  sdir=dxr(:,2)/norm(dxr(:,2));
  er=cross(sdir,VnGp); er=er/norm(er);
  es=cross(VnGp,er);   es=es/norm(es);
  et=VnGp/norm(VnGp);
  l1=(ex.'*er); m1=(ey.'*er); n1=(ez.'*er);
  l2=(ex.'*es); m2=(ey.'*es); n2=(ez.'*es);
  l3=(ex.'*et); m3=(ey.'*et); n3=(ez.'*et);
  Q=[l1*l1     m1*m1   n1*n1 l1*m1       m1*n1       n1*l1       ;
     l2*l2     m2*m2   n2*n2 l2*m2       m2*n2       n2*l2       ;
     l3*l3     m3*m3   n3*n3 l3*m3       m3*n3       n3*l3       ;
     2*l1*l2 2*m1*m2 2*n1*n2 l1*m2+l2*m1 m1*n2+m2*n1 n1*l2+n2*l1 ;
     2*l2*l3 2*m2*m3 2*n2*n3 l2*m3+l3*m2 m2*n3+m3*n2 n2*l3+n3*l2 ;
     2*l3*l1 2*m3*m1 2*n3*n1 l3*m1+l1*m3 m3*n1+m1*n3 n3*l1+n1*l3];
  Dsh=Q.'*D*Q;
  [B,GNL,epsEt,L(Gp,:),detJ]=formB(GpLoc(Gp,:),coord,oVn,t,g1,g2,...
                                   duvw,oL(Gp,:));
  epsE(:,Gp)=epsEn(:,Gp)+epsEt; sig(:,Gp)=sigN(:,Gp)+(Dsh*epsEt);
  H=[sig(1,Gp)*eye(3) sig(4,Gp)*eye(3) sig(6,Gp)*eye(3);
     sig(4,Gp)*eye(3) sig(2,Gp)*eye(3) sig(5,Gp)*eye(3);
     sig(6,Gp)*eye(3) sig(5,Gp)*eye(3) sig(3,Gp)*eye(3)];
  ke=ke+((B.'*Dsh*B)+(GNL.'*H*GNL))*detJ*wp(Gp);
  fint=fint+B.'*sig(:,Gp)*detJ*wp(Gp);
end

function [wp,GpLoc]=GpPos()
wp=ones(8,1); g2=1/sqrt(3); xsi=[-1 -1 1 1 -1 -1 1 1].'*g2;
eta=[-1 -1 -1 -1 1 1 1 1].'*g2; zet=[-1 1 1 -1 -1 1 1 -1].'*g2;
GpLoc=[xsi eta zet];

```

```
function [B,GNL,epsEt,L,detJ]=formB(GpLoc,coord,Vn,t,g1,g2,duvw,oL)
xsi=GpLoc(1); eta=GpLoc(2); zet=GpLoc(3);
N=shapefunc(xsi,eta); dNr=dershapefunc(xsi,eta); dxr=zeros(3);
dxr(1:2,:)=(dNr*(coord+0.5*zet*(Vn.*(t*ones(1,3)))));
dxr(3,:)=(0.5*(N.*t')*Vn);
detJ=det(dxr); invJ=inv(dxr); dNx=dxr\[dNr; zeros(1,9)];
G=zet*(invJ(:,1:2)*dNr)+invJ(:,3)*N;
B9=zeros(9,9*5); BNL=zeros(6,9*5); GNL=B9; L=zeros(1,9);
B9([1 4 9],1:5:end)=dNx; B9([5 2 6],2:5:end)=dNx;
B9([8 7 3],3:5:end)=dNx;
B9(:,4:5:end)=g1(:,[1 2 3 1 2 2 3 3 1])'.*G([1 2 3 2 1 3 2 1 3],:);
B9(:,5:5:end)=g2(:,[1 2 3 1 2 2 3 3 1])'.*G([1 2 3 2 1 3 2 1 3],:);
BL=B9([1:3 5 7 9],:); BL(4:6,:)=BL(4:6,:)+B9([4 6 8],:);
for n=1:3
  ct=3*(n-1)+1:3*(n-1)+3;
  for m=1:9
    m5=5*(m-1);
    L(:,ct)=L(:,ct)+([dNx(:,m) g1(m,n).*ones(3,1).*G(:,m)...
                      g2(m,n).*ones(3,1).*G(:,m)]*[duvw((m5)+n);
                      duvw((m5)+4); duvw((m5)+5)])';
  end
end
tL=oL+L;
A=[tL(:,1:3:9)  zeros(1,3)  zeros(1,3);
    zeros(1,3) tL(:,2:3:9)  zeros(1,3);
    zeros(1,3)  zeros(1,3) tL(:,3:3:9);
   tL(:,2:3:9) tL(:,1:3:9)  zeros(1,3);
    zeros(1,3) tL(:,3:3:9) tL(:,2:3:9);
   tL(:,3:3:9)  zeros(1,3) tL(:,1:3:9)];
for n=1:9
  ct=5*(n-1)+1:5*(n-1)+5;
  GNL(:,ct)=[dNx(1,n)*eye(3) g1(n,:)'.*G(1,n).*ones(3,1)...
             g2(n,:)'.*G(1,n).*ones(3,1);
             dNx(2,n)*eye(3) g1(n,:)'.*G(2,n).*ones(3,1)...
             g2(n,:)'.*G(2,n).*ones(3,1);
             dNx(3,n)*eye(3) g1(n,:)'.*G(3,n).*ones(3,1)...
             g2(n,:)'.*G(3,n).*ones(3,1)];
  BNL(:,ct)=A*GNL(:,ct);
end
B=BL+BNL;
epsEt=[(L(1)+ L(1)+oL(1)* L(1)+oL(4)*L(4)+oL(7)*L(7)+L(1)*oL(1)+...
        L(4)*oL(4)+ L(7)*oL(7)+ L(1)*L(1)+ L(4)*L(4)+L(7)* L(7))/2;
       (L(5)+ L(5)+oL(2)* L(2)+oL(5)*L(5)+oL(8)*L(8)+L(2)*oL(2)+...
        L(5)*oL(5)+ L(8)*oL(8)+ L(2)*L(2)+ L(5)*L(5)+L(8)* L(8))/2;
       (L(9)+ L(9)+oL(3)* L(3)+oL(6)*L(6)+oL(9)*L(9)+L(3)*oL(3)+...
        L(6)*oL(6)+ L(9)*oL(9)+ L(3)*L(3)+ L(6)*L(6)+L(9)* L(9))/2;
        L(2)+ L(4)+oL(1)* L(2)+oL(4)*L(5)+oL(7)*L(8)+L(1)*oL(2)+...
        L(4)*oL(5)+ L(7)*oL(8)+ L(1)*L(2)+ L(4)*L(5)+L(7)* L(8);
        L(8)+ L(6)+oL(3)* L(2)+oL(6)*L(5)+oL(9)*L(8)+L(3)*oL(2)+...
        L(6)*oL(5)+ L(9)*oL(8)+ L(3)*L(2)+ L(6)*L(5)+L(9)* L(8);
        L(3)+ L(7)+oL(1)* L(3)+oL(4)*L(6)+oL(7)*L(9)+L(1)*oL(3)+...
        L(4)*oL(6)+ L(7)*oL(9)+ L(1)*L(3)+ L(4)*L(6)+L(7)* L(9)];
```

```
function [N]=shapefunc(xsi,eta)
N(:,1)= xsi.*(xsi-1).*eta.*(eta-1)      /4;
N(:,2)=-xsi.*(xsi-1).*(eta+1).*(eta-1) /2;
N(:,3)= xsi.*(xsi-1).*eta.*(eta+1)      /4;
N(:,4)=-(xsi+1).*(xsi-1).*eta.*(eta+1) /2;
N(:,5)= xsi.*(xsi+1).*eta.*(eta+1)      /4;
N(:,6)=-xsi.*(xsi+1).*(eta+1).*(eta-1) /2;
N(:,7)= xsi.*(xsi+1).*eta.*(eta-1)      /4;
N(:,8)=-(xsi+1).*(xsi-1).*eta.*(eta-1) /2;
N(:,9)=(xsi+1).*(xsi-1).*(eta+1).*(eta-1);

function [dNr]=dershapefunc(xsi,eta)
r2=size(xsi,1)*2;
dNr(1:2:r2  ,1)= eta.*(eta-1).*(2*xsi-1)     /4;
dNr(1:2:r2  ,2)=-(eta+1).*(eta-1).*(2*xsi-1)/2;
dNr(1:2:r2  ,3)= eta.*(eta+1).*(2*xsi-1)     /4;
dNr(1:2:r2  ,4)=-eta.*(eta+1).*xsi             ;
dNr(1:2:r2  ,5)= eta.*(eta+1).*(2*xsi+1)     /4;
dNr(1:2:r2  ,6)=-(eta+1).*(eta-1).*(2*xsi+1)/2;
dNr(1:2:r2  ,7)= eta.*(eta-1).*(2*xsi+1)     /4;
dNr(1:2:r2  ,8)=-eta.*(eta-1).*xsi             ;
dNr(1:2:r2  ,9)= 2*(eta+1).*(eta-1).*xsi       ;
dNr(2:2:r2+1,1)= xsi.*(xsi-1).*(2*eta-1)     /4;
dNr(2:2:r2+1,2)=-xsi.*(xsi-1).*eta             ;
dNr(2:2:r2+1,3)= xsi.*(xsi-1).*(2*eta+1)     /4;
dNr(2:2:r2+1,4)=-(xsi+1).*(xsi-1).*(2*eta+1)/2;
dNr(2:2:r2+1,5)= xsi.*(xsi+1).*(2*eta+1)     /4;
dNr(2:2:r2+1,6)=-xsi.*(xsi+1).*eta             ;
dNr(2:2:r2+1,7)= xsi.*(xsi+1).*(2*eta-1)     /4;
dNr(2:2:r2+1,8)=-(xsi+1).*(xsi-1).*(2*eta-1)/2;
dNr(2:2:r2+1,9)= 2*(xsi+1).*(xsi-1).*eta       ;
```

Listing 2: Function files

3 RESULTS

Two benchmark problems[17] were used to validate the accuracy of the shell finite elements, which are cantilevers exposed to (i) a transverse end load and (ii) an end moment.

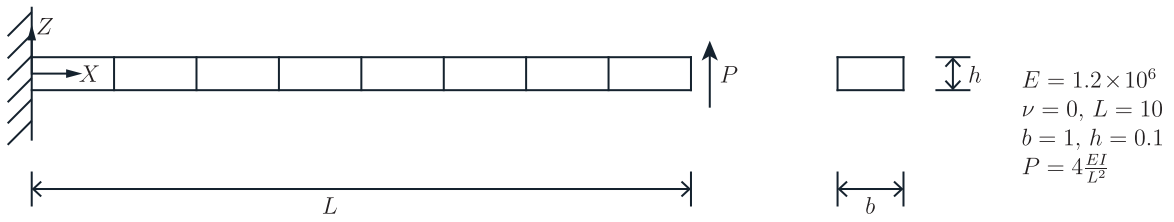


Figure 2: Mesh for end-loaded cantilever

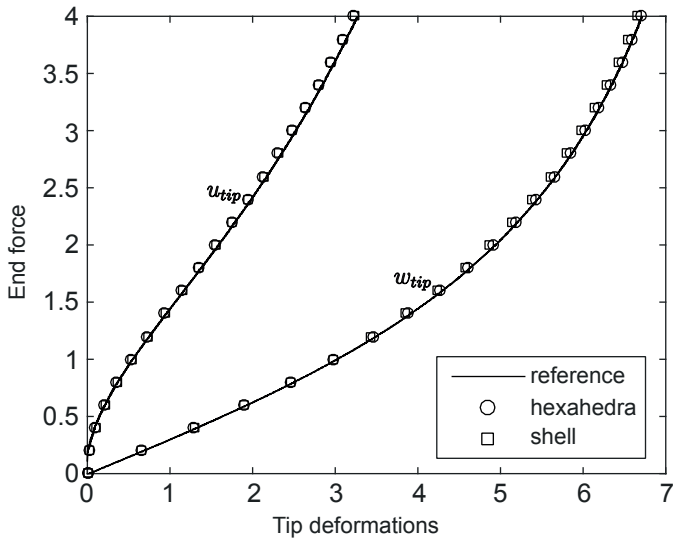


Figure 3: Load deflection curves for a cantilever subjected to an end point load

The first calibration problem was set-up as shown in Figure 2 was simulated with both hexahedra and shell elements. 500 hexahedral elements[2] were required, leading to a computational run time of 305s, whilst shell elements only required 8 elements with a computational run time of 28s to obtain results that are in close agreement with the benchmark solution. These results are shown in Figure 3.

The second calibration problem was set-up as shown in Figure 4 and its results are shown in Figure 5. Here an end moment was applied and the shell undergoes very large rotations.

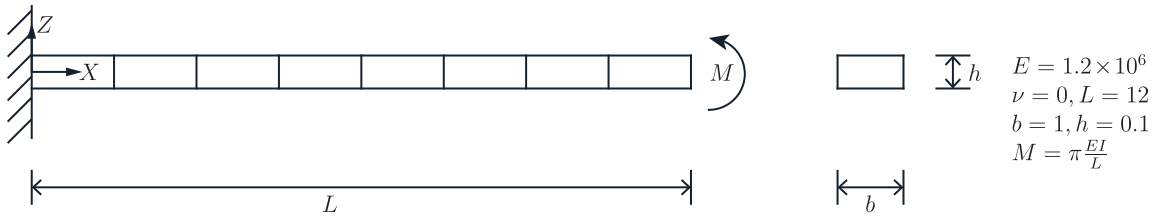


Figure 4: Mesh for moment-loaded cantilever

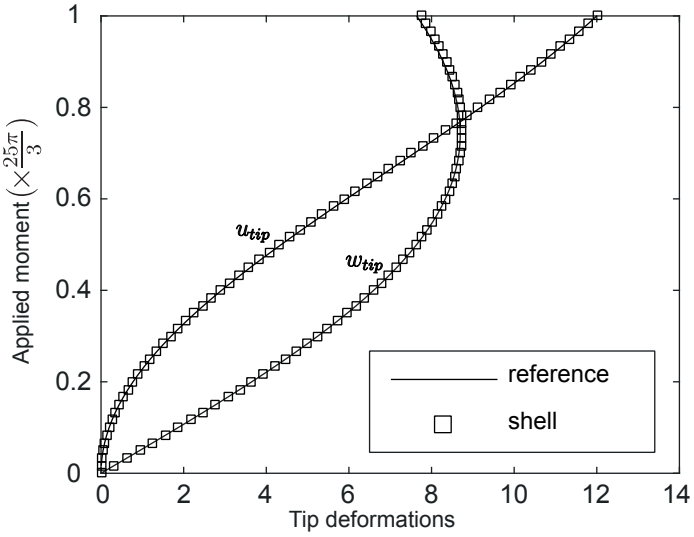


Figure 5: Load deflection curves for a cantilever shell subjected to an end moment

4 CONCLUSIONS

The analyses comfirm what has been known for many years, that shell elements can reduce the computational time significantly whilst still retaining the precision of hexahedra elements in bending cases. The compact MATLAB scripts given here provide engineers with a useful research tool to investigate the behaviour of shell structures. They are to be used in a coupled electro-mechanical analysis of a human heart as part of the first author's PhD investigation. We are greatly indebted to Dr William Coombs of Durham University for providing us with his geometrically linear shell element code. This formed a crucial starting point for the development of the *Total-Lagrangian* analysis.

REFERENCES

[1] Coombs, W.M. *et al.* (2011) Finite Deformation of Particulate Geomaterials: Frictional and Anisotropic Critical State Elasto-plasticity, Durham theses, Durham University.

[2] Coombs, W.M. *et al.* 70-line 3D Finite Deformation Elastoplastic Finite-element Code. *Numerical Methods in Geotechnical Engineering : Pro. of the Seventh European Conference on Num. Met. in Geo. Eng.* London: Taylor & Francis (2010) 151–156.

[3] European Heart Network. *European Cardiovascular Disease Statistics* (2017).

[4] Ruth, A.S. *Electro-Mechanical Large Scale Computational Models of the Ventricular Myocarduium* PhD thesis, Universitat Politècnica de Catalunya (2004).

[5] Antonioletti, M. *et al.* BeatBox-HPC Simulation Environment for Biophysically and Anatomically Realistic Cardiac Electrophysiology. *PLOS ONE* (2017), 12(5).

[6] Ten-Tusscher, K.H. *et al.* A Model for Human Ventricular Tissue. *Am. J. Physiol. Heart Circ. Physiol.* (2004) **286**(4): H1573–H1589.

[7] Iyer, V. *et al.* A Computational Model of the Human Left-Ventricular Epicardial Myocyte. *Biophys. J.* (2004) **87**(3): 1507–1525.

[8] Bueno-Orovio, A. *et al.* Minimal Model for Human Ventricular Action Potentials in Tissue. *J. Theor. Biol.* (2008) **253**(3): 544–560.

[9] Vigmond, E.J. *et al.* Computational Tools for Modeling Electrical Activity in Cardiac Tissue. *J. Electrocardiol.* (2003) **36**(Suppl): 69–74.

[10] Pitt-Francis, J. *et al.* Chaste: Using Agile Programming Techniques to Develop Computational Biology Software. *Philos. Trans. A. Math. Phys. Eng. Sci.* (2008) **366**(1878): 3111-3136.

[11] Clerx, M. *et al.* Myokit: A Simple Interface to Cardiac Cellular Electrophysiology.*Prog. Biophys. Mol. Biol.* (2016) **120**(1-3): 100–114.

[12] Okada, J.I. *et al.* Absence of Rapid Propagation through the Purkinje Network as a Potential Cause of Line Block in the Human Heart with Left Bundle Branch Block. *Front PHysiol.* (2018) **9**: 56.

[13] Bathe, K.J. and Bolourchi S. A Geometric and Material Nonlinear Plate and Shell Element. *Comp. & Struc.* (1980) **11**: 23–48.

[14] Bathe, K.J. *Finite Element Procedures.* 2nd Edition, 2014.

[15] Bathe, K.J. *et al.* Finite Element Formulations for Large Deformation Dynamic Analysis *Int. Jour. for Num. Meth. In Eng.* (1975) **9**: 353–386.

[16] Stegmann, J. *Finite Elements for Geometric Non-Linear Analysis Of Composite Laminates and Sandwich Structures.* MSc thesis, Aalborg University (2001).

[17] Sze, K. *et al.* Popular Benchmark Problems for Geometric Nonlinear Analysis of Shells. *Finite Elements In Analysis And Design* (2004) **40**(11): 1551-1569.