Technical University of Catalonia

INTERNSHIP

Computational Support of Traction Force Microscopy (TFM) in Cell and Tissue Mechanics

M.Sc. in Computational Mechanics UPC – CIMNE

Masters Internship at institute of bioengineering of Cataluña – IBEC

Mohammad Mohsen Zadehkamand

Supervised by:

Prof. Marino Arroyo [at UPC]

Prof. Xavier Trepat [at IBEC]

Eng. Ernest Latorre Ibars [at UPC – IBEC]

30 August 2017

Table of Contents

1. Introduction

A cell traction force is defined as a tangential tension exerted by cells to the extracellular matrix or the underlying layer. It is crucial to many biological processes like inflammation, wound healing, angiogenesis and metastasis. It can direct many cellular functions such as cell migration, extracellular matrix organization and mechanical signal generation. Traction Force Microscopy (TFM) is an experimental method for determining the tractions on the surface of a biological cell by obtaining measurements of the surrounding displacement field. These methods are widely used to quantify cellular forces in mechanobiological studies. These methods are inverse, in the sense that forces must be determined such that they comply with a measured displacement field. [1]

At present, Cell Traction Force Microscopy (CTFM) is among the most efficient and reliable method for determining CTF field of an entire cell spreading on a two-dimensional (2D) substrate surface. Recent researches improve CTFM methods such that they can automatically track dynamic CTFs, thereby providing new insights into cell motility in response to altered biological conditions. In addition, research effort should be devoted to developing novel experimental and theoretical methods for determining CTFs in three-dimensional (3D) matrix, which better reflects physiological conditions than 2D substrate used in current CTFM methods [2].

The tractions can be measured by observing the displacements of beads embedded on a flexible gel substrate on which the cells are cultured. This is shown in Figure1 as colorful beads. Exact solutions are presented widely to the problem of computing the traction field from the observed displacement field. The solution rests on recasting the relationship between displacements and tractions into Fourier space, where the recovery of the traction field is especially simple [3].

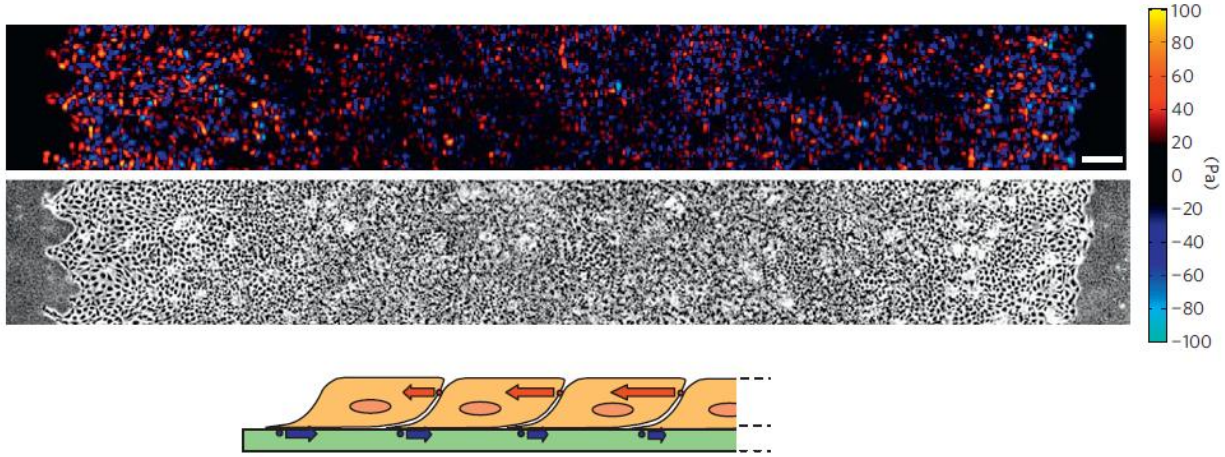


Figure 1. Cells use a tug–of–war mechanism to integrate local tractions (red) into long-range gradients of intra- and inter-cellular tension (blue). A small portion of the traction that each cell generates is transmitted to the cell behind.

However, these exact (analytical) methods are not so flexible for non-regular domains or domains containing holes or irregularities. In this study, a new approach is conducted in order to use approximated finite element methods which is so flexible for any domain or material property. As we may witness in the last chapter, the FEM results has a good correlation with exact methods but help us so much in studying 3D free-shape domains.

At IBEC, new technologies are developed to map and perturb the main physical properties that determine how cells and tissues grow, move, invade and remodel. This physical information is combined with systematic molecular perturbations and computational models and the governing principles on the interplay between chemical and physical cues in living tissues are explored. To study cell and tissue dynamics new technologies are developed to measure physical forces at the cell-cell and cell-matrix interface. Using these technologies beside computational analysis of cells we obtain a full experimental characterization of epithelial dynamics during tissue growth, wound healing and cancer cell invasion [4].

[1]. Zündel M, Ehret AE, Mazza E (2017) Factors influencing the determination of cell traction forces. PLoS ONE 12(2): e0172927. https://doi.org/10.1371/journal.pone.0172927

[2]. Cell traction force and measurement methods. Wang JH1, Lin JS. Biomech Model Mechanobiol. 2007 Nov;6(6):361-71. Epub 2007 Jan 3.

[3]. Traction fields, moments, and strain energy that cells exert on their surroundings. Am J Physiol Cell Physiol 282: C595–C605, 2002. First published October 31, 2001; 10.1152/ajpcell.00270.2001.

[4]. http://ibecbarcelona.eu/integrative

2. Open Source FEM Codes for Hyperelastic Models

The initial step for this study is to find an appropriate open source FEM software package containing hyperelastic models. Moreover, since at IBEC laboratory all the procedure of mechanical modeling of cells should be done in a fully automatic manner, the chosen software should have the possibility of operating by an external code like MATLAB automatically. Following, selected FEM softwares are introduced, briefly.

2.1. Vega

Vega is a computationally efficient and stable C/C++ physics library which is designed to model large deformations of 3D solid deformable objects, including geometric and material nonlinearities, and can also efficiently simulate linear systems.

For any 3D tetrahedral or cubic mesh, Vega can compute the elastic energy, the internal elastic forces and their gradients (tangent stiffness matrix), in any deformed configuration. Different parts of the mesh can be assigned arbitrary material properties. It conducts the time-stepping under any user-specified forces, using several provided integrators: implicit backward Euler, implicit Newmark and explicit central differences and it is important to note that all models include support for multi-core computing.

About the Material models it includes linear materials, neo-Hookean and Mooney-Rivlin nonlinear material models. Arbitrary nonlinear material models can be added to Vega. For isotropic hyperelastic materials, this is as easy as defining an energy function, and its first and second derivatives.

Most of Vega was written by Jernej Barbič, during his doctoral studies at CMU, postdoctoral research at MIT, and faculty position at USC. More details can be found on the reference website:

http://run.usc.edu/vega/index.html

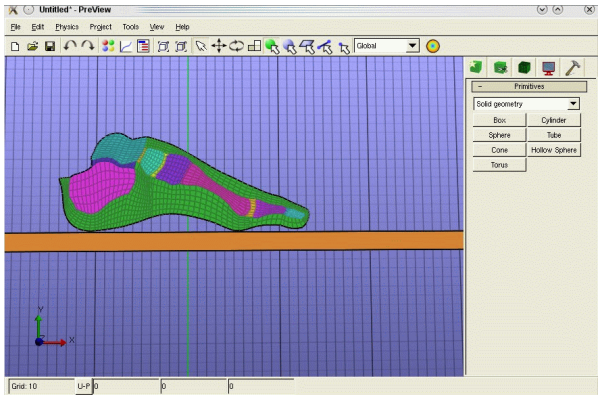
2.2. FEBio

In the field of computational biomechanics, investigators have primarily used commercial software that is neither geared toward biological applications nor sufficiently flexible to follow the latest developments in the field. FEBio suite is a nonlinear implicit finite element framework, designed specifically for analysis in computational solid biomechanics. This open-source software is written in C++, with particular attention to scalar and parallel performance on modern computer architectures. It is consisted of three parts:

- **PreView**: A finite element preprocessing package designed for mesh generation; mesh editing, specification of materials, boundary conditions and analysis options. User interaction in PreView occurs via a user-friendly GUI.

- **FEBio**: As a nonlinear implicit FE solver, offers modeling scenarios, constitutive models and boundary conditions that are relevant to many research areas in biomechanics. All features can be used together seamlessly, giving the user a powerful tool for solving 3D problems in computational biomechanics.
- **Postview**: A finite element post-processor that is designed to visualize and analyze results from an FE analysis. It can import the FEBio extendible plot file format (XPLT), as well as several other data formats (e.g. LSDYNA Keyword, LSDYNA binary database, VTK). It also offers a way to add additional data to an already loaded model.

It can include prescribed displacements, nodal forces and pressure forces, fluid pressure and flux. FEBio provides the ability to represent frictionless contact for elastic, viscoelastic, rigid, and multiphasic materials, as well as frictional contact for elastic and viscoelastic materials. Finally, the user may specify a body force to model the effects of, for instance, gravity or base acceleration.



It was originally developed in the Musculoskeletal Research Laboratories at the University of Utah. More details can be found on the reference website:

https://febio.org/

2.3. FEniCS Project

The FEniCS Project is a collection of free, open source, software components with the common goal to enable automated solution of differential equations providing scientific computing tools for working with computational meshes, finite element variational formulations of ODEs and PDEs, and numerical linear algebra. With the high-level Python and C++ interfaces it is designed as an umbrella project for a collection of interoperable components. The core components are:

- UFL (Unified Form Language), a domain-specific language embedded in Python for specifying finite element discretizations of differential equations in terms of finite element variational forms;
- FIAT (Finite element Automatic Tabulator), a Python module for generation of arbitrary order finite element basis functions on simplices;
- FFC (FEniCS Form Compiler), a compiler for finite element variational forms taking UFL code as input and generating UFC output;

- UFC (Unified Form-assembly Code), a C++ interface consisting of low-level functions for evaluating and assembling finite element variational forms;
- Instant, a Python module for inlining C and C++ code in Python;
- DOLFIN, a C++/Python library providing data structures and algorithms for finite element meshes, automated finite element assembly, and numerical linear algebra. DOLFIN functions as the main problem solving environment and user interface. Its functionality integrates the other FEniCS components and handles communication with external libraries such as PETSc, Trilinos and Eigen for numerical linear algebra, ParMETIS and SCOTCH for mesh partitioning, and MPI and OpenMP for distributed computing.

More details can be found on the reference website:

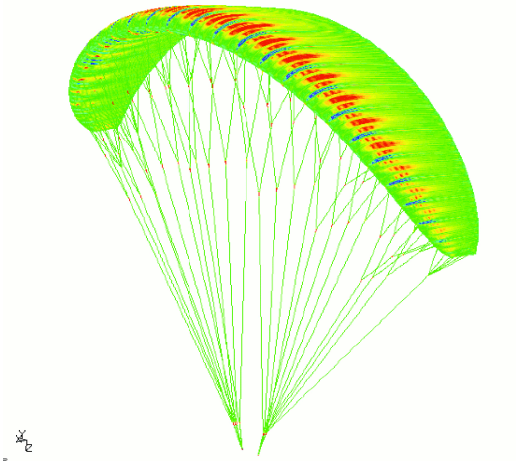
https://fenics.readthedocs.io/en/latest/

https://fenicsproject.org/

2.4. CalculiX

CalculiX is a package designed to solve field problems using finite element method. The pre- and post-processor is an interactive 3D-tool using the openGL API. The solver is able to do linear and non-linear calculations for Static, dynamic and thermal loadings.

Because the solver makes use of the abaqus input format it is possible to use commercial pre-processors as well. In turn the pre-processor is able to write mesh related data for nastran, abaqus, ansys, code-aster and for the free-cfd codes dolfyn, duns, ISAAC and OpenFOAM. A simple step reader is included. In addition external CAD interfaces like vda_to_fbd are available. The program is designed to run on Unix platforms like Linux and Irix computers but also on MS-Windows.



Linear elastic, isotropic hyperelastic, deformation plasticity, large deformation incremental isotropic plasticity with isotropic and kinematic hardening models are included.

The CalculiX package was developed at MTU Aero Engines in Munich, Germany which granted the publication. More details can be found on the reference website:

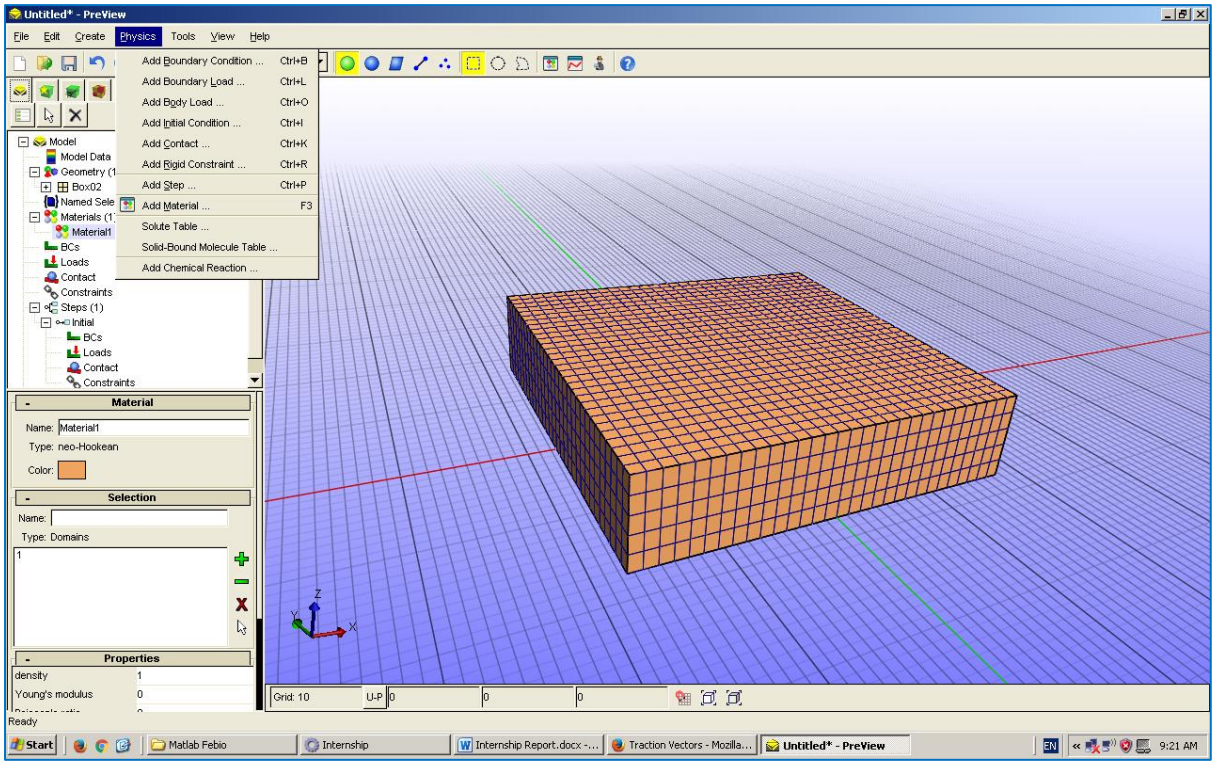
http://www.dhondt.de/

http://www.calculix.de/

3. FEBio Steps – MATLAB automatic procedure

In this study the FEBio suite is chosen between different open source FEM packages because of its simplicity in installation, application and also the fact that it can be operated on both 32 and 64 bit computer architectures either on Windows or Linux. On the other hand, this package can be linked to famous FEM software like ABAQUS and ANSYS for the input files which one may like to use in next studies.

In this part a brief review is introduced on the interactive atmosphere of the program which would help the reader to have a sense of what the program is doing in different modules.



For this purpose the following steps shall be done:

- building the **geometry** in Pre-View Program
- Assigning **material** properties
- **Meshing**
- Assigning fixed **zero displacement** to bottom layer
- Assigning fixed **Dirichlet** boundary condition to top layer
- Defining **steps**
- Saving the Pre-View file and exporting it as the input for FEBio program
- Running the FEBio file and obtaining the result file
- Opening the results in Post-View program and dealing with the different stress and displacement components for element and nodes.

Here we describe each related part in MATLAB code which does the whole procedure in a fully automated manner:

- **Input Data.**
  - Code will ask for the size of 3D sample and the user-desired number of elements in each direction.

- **Extracting top layer's Displacement, Traction & Stress field from reference.**
  - Code will reduce the size of reference displacement and stress field by an interpolation scheme in MATLAB [***Imresize***] to the user-desired number of elements if it is lower than the reference one.

- **Filling Data into 'FEBio_input.feb' file.**
  - Code will Input various material properties for the ***New_Hookean*** model. On the other hand many other material models like ***Mooney-Rivlin***, ***isotropic elastic*** and ***orthotropic elastic*** are also applicable in FEBio for this part beside some specific material models for muscle, Tendon, Ogden and lots of other variants.

- **Filling Coordinate & Connectivity Data into ''FEBio_input.feb' file.**
  - Code will automatically build the appropriate mesh which is readable by FEBio, using the data given by user.

- **Filling top layer's Displacement field Data into ''FEBio_input.feb' file.**
  - These imposed displacements would act as the ***Dirichlet boundary conditions*** on the top layer of medium. The bottom layer of medium is considered as fully fixed.

- **Run FEBio & Continue to Post-Process.**
  - In this part the MATLAB code will ask for the permissions of running FEBio after building its complete input file; then it would open the Post-View program if user wants to play around with the graphic interface of the post processor program and see different results from different angles. Finally it would call for the last part of the code which is summarizing the results.

- **Averaging element stresses in joint nodes.**
  - The stress results in FEBio are reported for each element (Gauss Point). On the other hand, the reference results are provided on the nodes; so as we are going to compare the stress results of reference and FEBio together, we would need to transfer the elemental stresses to nodal ones by doing an averaging mechanism.

- **Transferring node Stresses, from deformed coordinate axis to original one.**
  - As we know in large deformation problems, when we are dealing with ***Cauchy stresses*** they are actually reported on the deformed mesh. But one may be interested to have the results on the original coordinate axis. For this reason we are using the ability of MATLAB called [***ScatteredInterpolant***]. Using this ability, one can

pass a set of (x,y) points and values, v, to scatteredInterpolant, and it returns a surface of the form v = F(x, y). This surface always passes through the sample values at the point locations. So we can evaluate this surface at any query point (xq,yq), to produce an interpolated value, vq.

- **Traction force calculation using stress matrix and surface unit normal vector.**
  - The final fruit of this study which is so interesting for the biologists are the traction forces which are transferred between the growing cells and the substrate. For this purpose we only need to know the unit vector in the normal direction of top layer elements. This can be easily achieved by a MATLAB ability called [***Surfnorm***]. Then by multiplying the 3 by 3 stress matrix of each node to its unit normal vector, we will have 3 values for the traction force for each node. The relationship between the traction vector and stress state at a point are as following:

$$\boldsymbol{T} = \boldsymbol{\sigma}.\boldsymbol{n}$$

Each component of T vector is calculated by:

$$\begin{bmatrix} \sigma_{xx}n_x + \sigma_{xy}n_y + \sigma_{xz}n_z = T_x \\ \sigma_{yx}n_x + \sigma_{yy}n_y + \sigma_{yz}n_z = T_y \\ \sigma_{zx}n_x + \sigma_{zy}n_y + \sigma_{zz}n_z = T_z \end{bmatrix}$$

Finally the normal and shear stresses on the surface are related to the traction vector by:

$$\sigma = \boldsymbol{T}.\boldsymbol{n} = \boldsymbol{n}.\boldsymbol{\sigma}.\boldsymbol{n}$$

$$\tau = \boldsymbol{T}.\boldsymbol{s} = \boldsymbol{s}.\boldsymbol{\sigma}.\boldsymbol{n}$$

Recall that ***n*** is the unit normal vector to the surface, and ***s*** is the unit vector parallel to it.

- **Plot Stress Matrices.**
  - The last part in the MATLAB code is to plot reference and FEBio results side by side to have an estimation of FEBio's accuracy compared the reference results.

4. Results

Here we are comparing final FEBio results by reference study results which is conducted in IBEC on a sample with a mesh of 116x116 on top with different mesh sizes. The size of sample is 350x350x80 micro meter and E=3000 Kpa and nu=0.48 is considered for the material properties.

Different mesh sizes in FEBio are checked due to memory restrains and here only the results for mesh of 60x60x20 and 100x100x12 element are provided. As we can see a good correlation is observed between reference and FEBio results for all stress components.

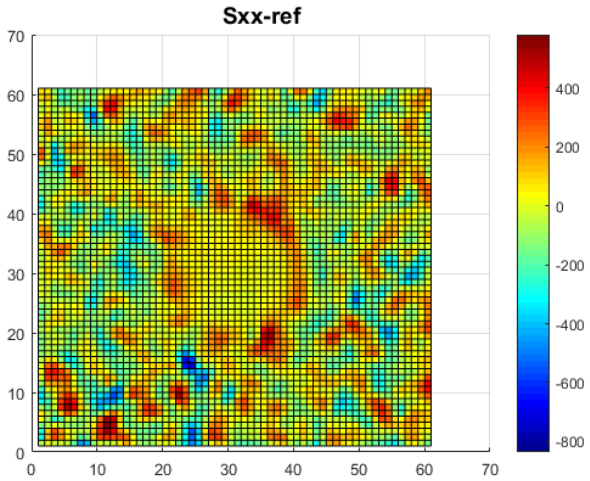


Figure1a. Reference results – mesh 60x60 – Sxx

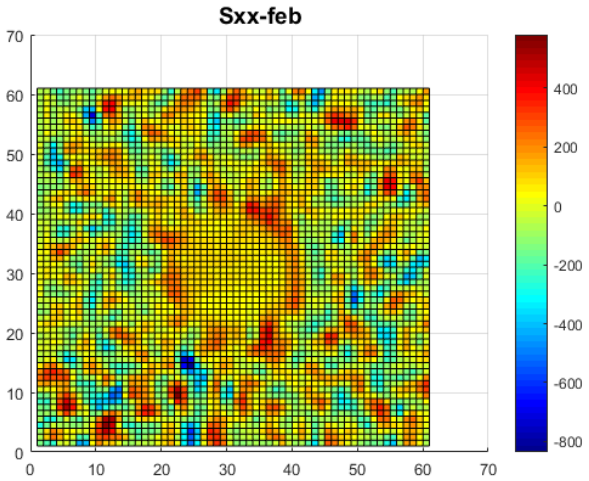


Figure1b. Reference results – mesh 60x60 – Sxx

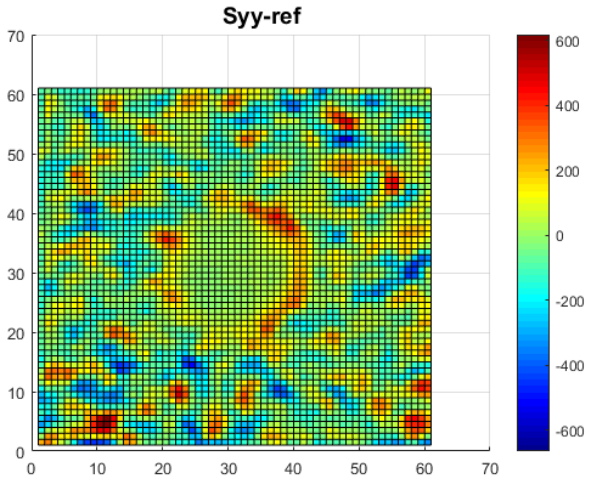


Figure2a. Reference results – mesh 60x60 – Syy

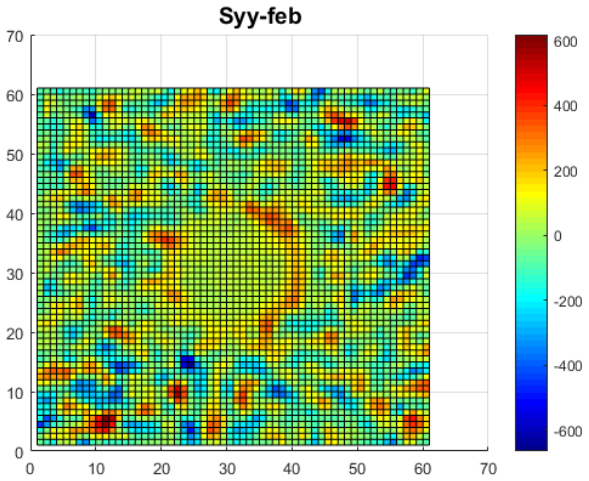


Figure2b. Reference results – mesh 60x60 – Syy

In Figure 3 some oscillations are observed which are mainly due to the way we average the stresses from elements into joint nodes, but the peaks and the general trend is completely matched by the reference results

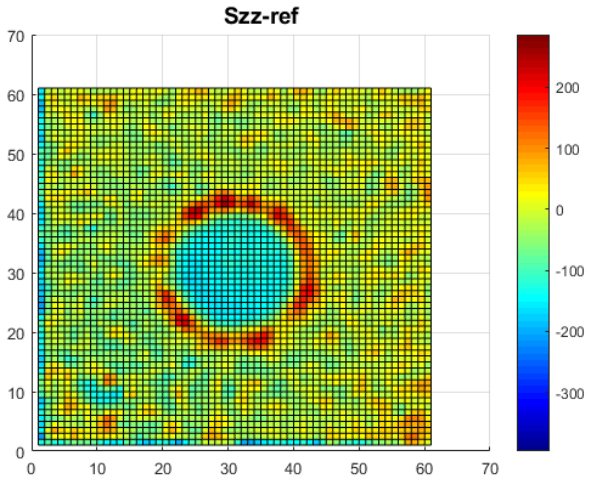


Figure3a. Reference results – mesh 60x60 – Szz

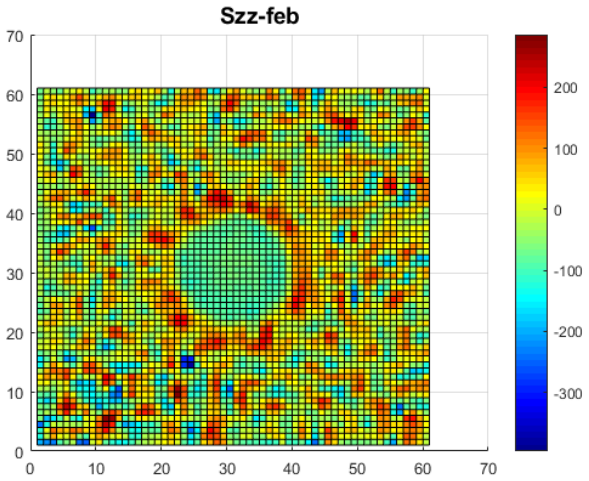


Figure3b. Reference results – mesh 60x60 – Szz

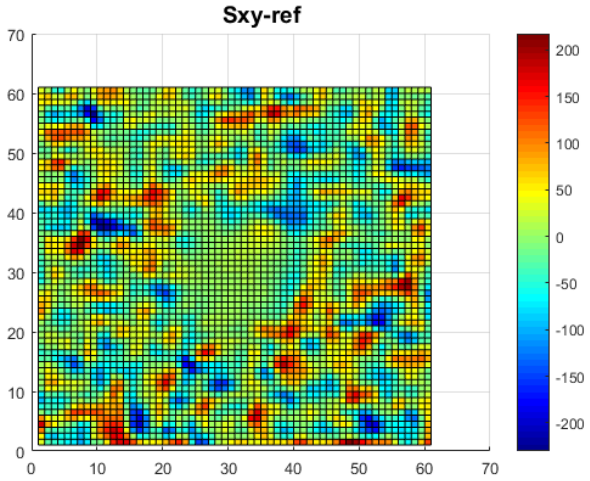


Figure4a. Reference results – mesh 60x60 – Sxy

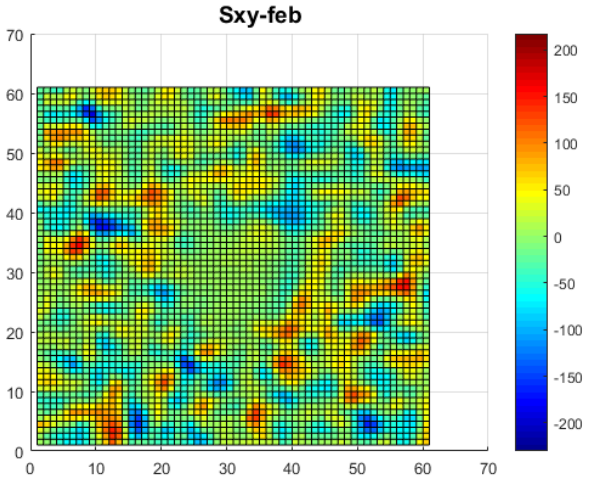


Figure4b. Reference results – mesh 60x60 – Sxy

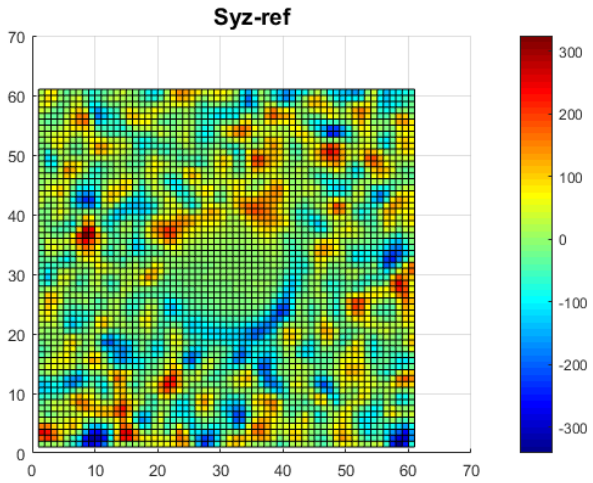


Figure5a. Reference results – mesh 60x60 – Syz

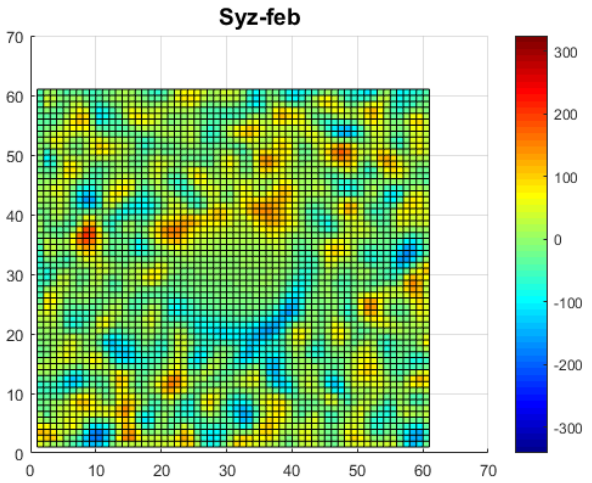


Figure5b. Reference results – mesh 60x60 – Syz

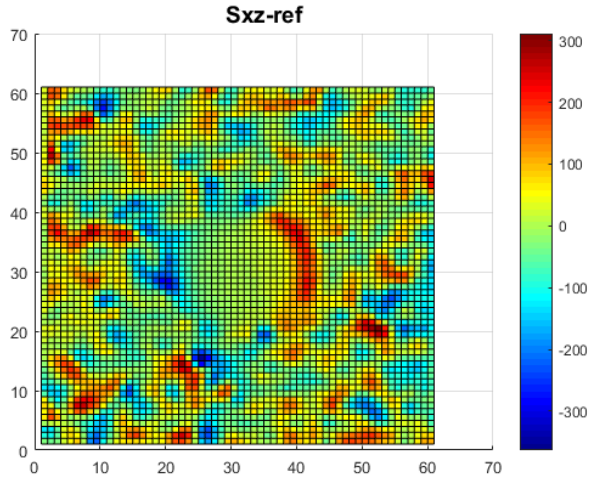


Figure6a. Reference results – mesh 60x60 – Sxz

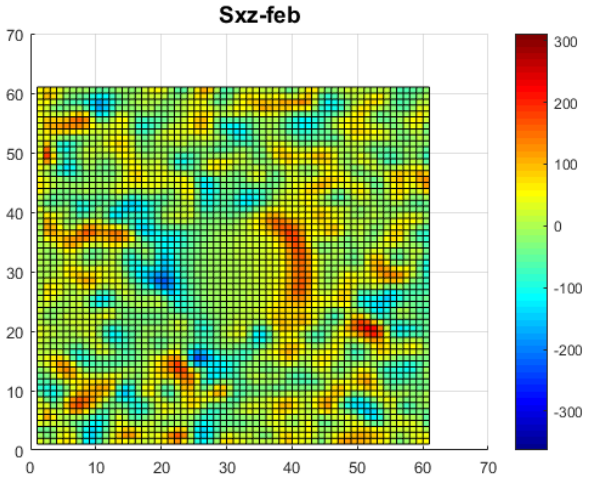


Figure6b. Reference results – mesh 60x60 – Sxz

Here the FEBio results for a finer mesh are provided and as we see the general calculations for stresses are almost identical as the reference results.

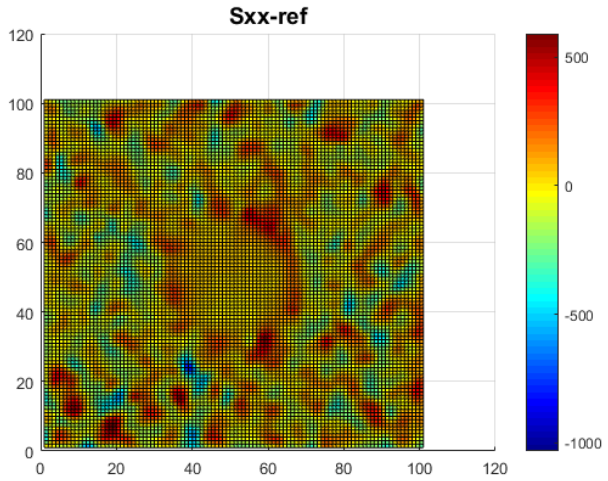


Figure1a. Reference results – mesh 100x100 – Sxx

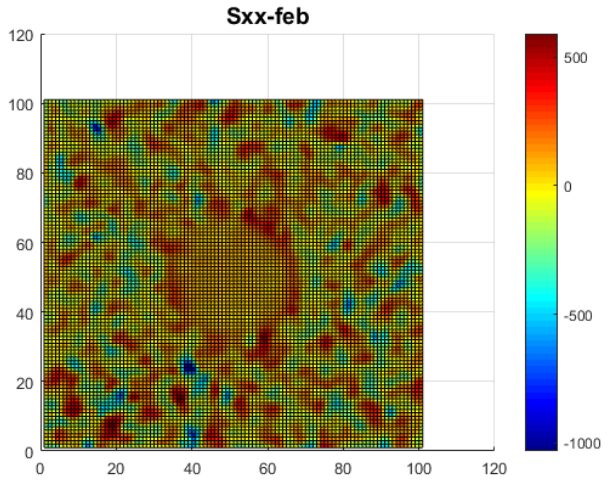


Figure1b. Reference results – mesh 100x100 – Sxx

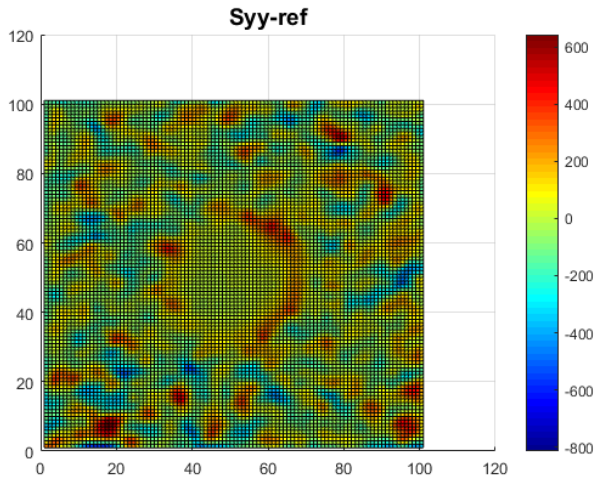


Figure2a. Reference results – mesh 100x100 – Syy

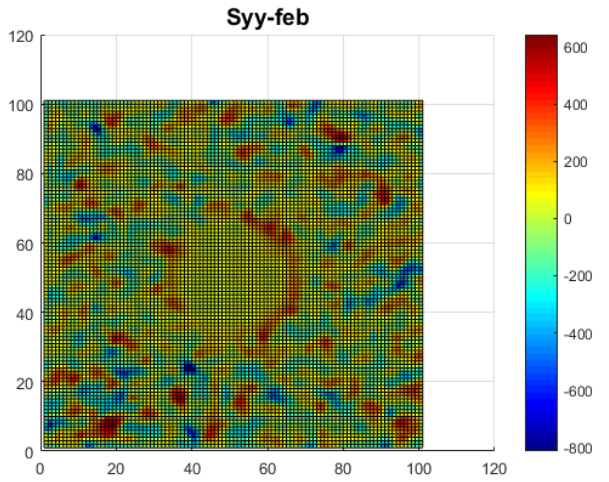


Figure2b. Reference results – mesh 100x100 – Syy

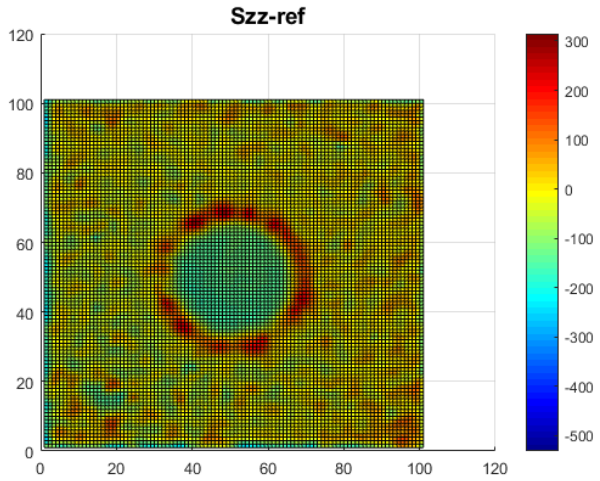


Figure3a. Reference results – mesh 100x100 – Szz

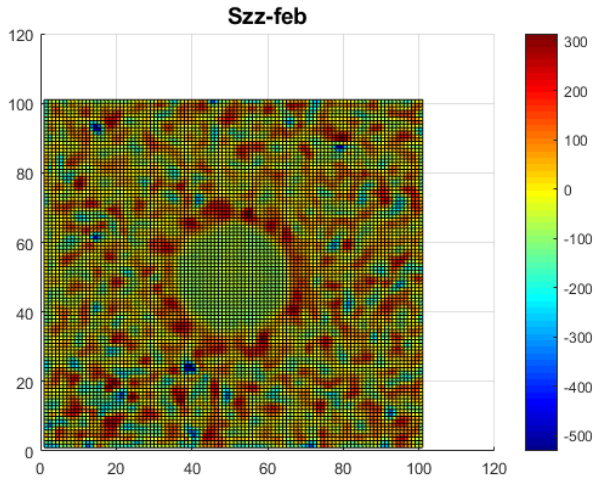


Figure3b. Reference results – mesh 100x100 – Szz

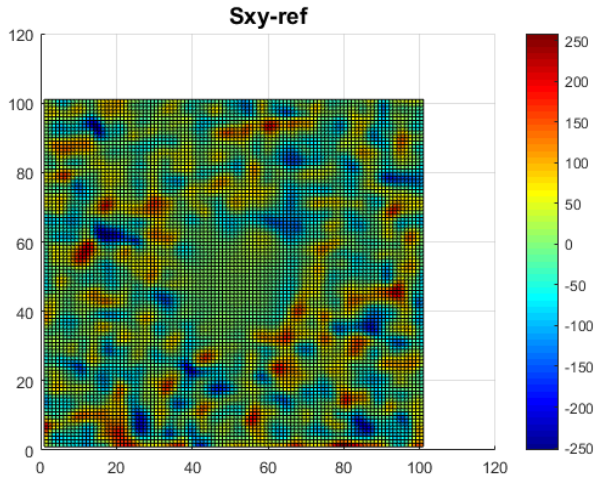


Figure4a. Reference results – mesh 100x100 – Sxy

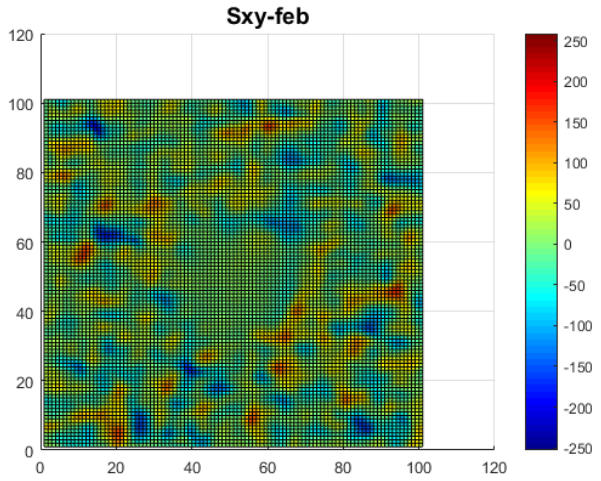


Figure4b. Reference results – mesh 100x100 – Sxy

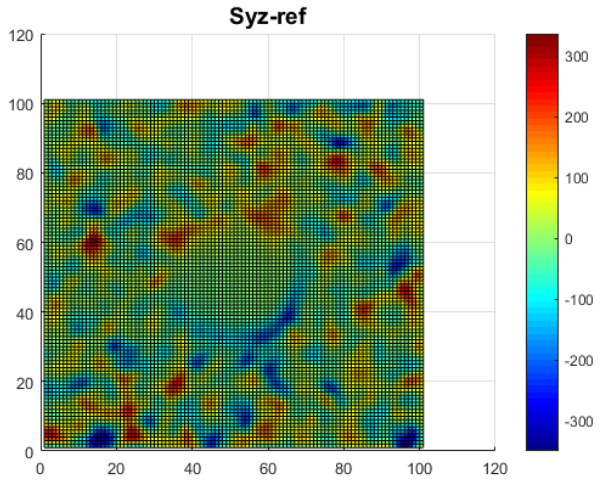


Figure5a. Reference results – mesh 100x100 – Syz

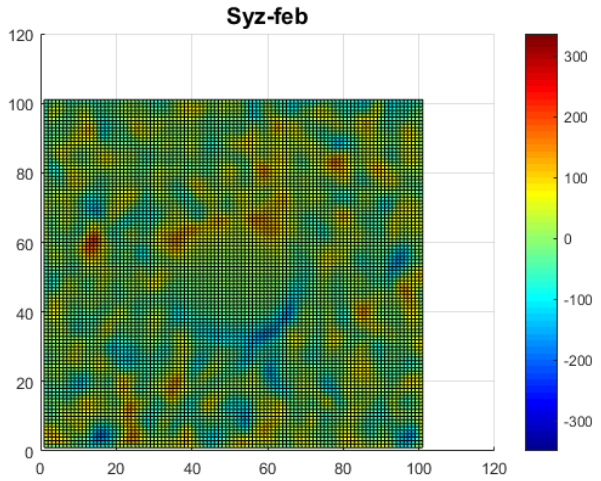


Figure5b. Reference results – mesh 100x100 – Syz

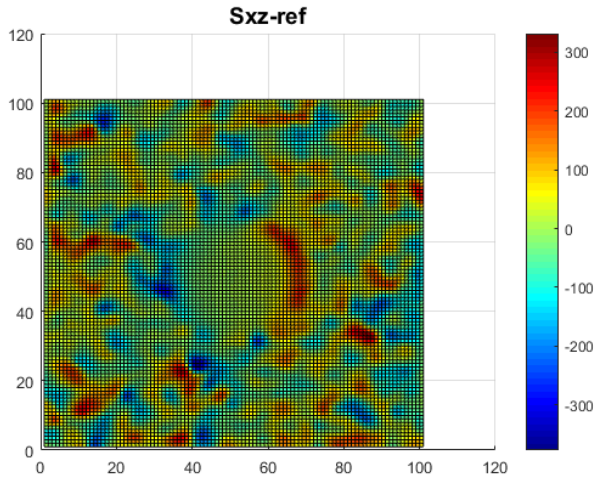


Figure6a. Reference results – mesh 100x100 – Sxz

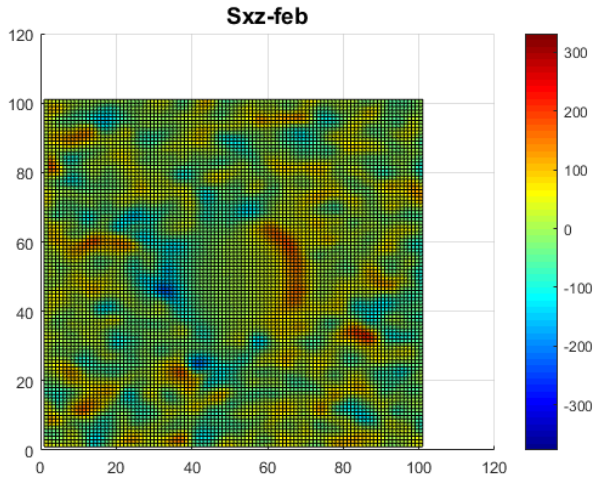


Figure6b. Reference results – mesh 100x100 – Sxz

❖ *Appendix – MATLAB codes*

1. *Appendix1: PreFEBio_Input.m*

```
clear
clc

%% Input Data

fileID  = fopen('FEBio_input.feb','w');
fileNm1 = 'FEBio_input.feb';
fileNm2 = 'FEBio_input.xplt';

a = 350.0 ; b = a ; d = 80.0;
MeshSize = 100;                                % 20 - 60 - 100 - 116

switch MeshSize
    case 20
        nx = 20  ; ny = nx ; nz = 10;
    case 60
        nx = 60  ; ny = nx ; nz = 20;
    case 90
        nx = 90  ; ny = nx ; nz = 15;
    case 116
        nx = 116 ; ny = nx ; nz = 10;
end

Node_N = (nx+1)*(ny+1)*(nz+1);                % Nr. of All Nodes
Elem_N = nx*ny*nz;                            % Nr. of All Elements


%% Extracting top layer's Displacement, Traction & Stress field from ref.

%Plot Extracted Displacement and Stress fields compared to original ones---
Plot_Extract = 0;
%1 = Displacement x,y,z   |   2 = Stress xx,yy,zz   |   3 = Stress xy,yz,xz

[Disp_ref_mat,Trac_ref_mat,Strs_ref_mat,Disp_ref_vec,Trac_ref_vec,Strs_ref_
vec]=Extraction(nx,ny,Plot_Extract);

%Plot Febio Stress fields compared to Reference ones-----------------------
Plot_Stress  = 0;
%1 = plot   |   0 = Don't Plot

%% Filling Data into '###.feb' file

fprintf(fileID,'<?xml version="1.0" encoding="ISO-8859-1"?>\r\n');
fprintf(fileID,'<febio_spec version="2.0">\r\n');
fprintf(fileID,'    <Module type="solid"/>\r\n');
fprintf(fileID,'    <Control>\r\n');
% fprintf(fileID,'        <time_steps>10</time_steps>\r\n');
fprintf(fileID,'        <time_steps>1</time_steps>\r\n');
% fprintf(fileID,'        <step_size>0.1</step_size>\r\n');
fprintf(fileID,'        <step_size>1.0</step_size>\r\n');
fprintf(fileID,'        <max_refs>15</max_refs>\r\n');
fprintf(fileID,'        <max_ups>10</max_ups>\r\n');
```

```
fprintf(fileID,'        <dtol>0.001</dtol>\r\n');
fprintf(fileID,'        <etol>0.01</etol>\r\n');
fprintf(fileID,'        <rtol>0</rtol>\r\n');
fprintf(fileID,'        <lstol>0.9</lstol>\r\n');
fprintf(fileID,'        <time_stepper>\r\n');
% fprintf(fileID,'            <dtmin>0.01</dtmin>\r\n');
% fprintf(fileID,'            <dtmax>0.1</dtmax>\r\n');
fprintf(fileID,'            <dtmin>1.0</dtmin>\r\n');
fprintf(fileID,'            <dtmax>1.0</dtmax>\r\n');
fprintf(fileID,'            <max_retries>5</max_retries>\r\n');
fprintf(fileID,'            <opt_iter>10</opt_iter>\r\n');
fprintf(fileID,'        </time_stepper>\r\n');
fprintf(fileID,'        <analysis type="static"/>\r\n');
fprintf(fileID,'    </Control>\r\n');
fprintf(fileID,'    <Globals>\r\n');
fprintf(fileID,'        <Constants>\r\n');
fprintf(fileID,'            <T>0</T>\r\n');
fprintf(fileID,'            <R>0</R>\r\n');
fprintf(fileID,'            <Fc>0</Fc>\r\n');
fprintf(fileID,'        </Constants>\r\n');
fprintf(fileID,'    </Globals>\r\n');
fprintf(fileID,'    <Material>\r\n');
fprintf(fileID,'        <material id="1" name="Material1" type="neo-
Hookean">\r\n');
fprintf(fileID,'            <density>1</density>\r\n');
fprintf(fileID,'            <E>3000.0</E>\r\n');
fprintf(fileID,'            <v>0.48</v>\r\n');
fprintf(fileID,'        </material>\r\n');
fprintf(fileID,'    </Material>\r\n');

%% Filling Coordinate & Connectivity Data into '###.feb' file

%--------------------------------------------------------------------------
[N,T] = Connectivity(a,b,d,nx,ny,nz);
%--------------------------------------------------------------------------

fprintf(fileID,'    <Geometry>\r\n');
fprintf(fileID,'        <Nodes>\r\n');

formatSpec1 = '%10s%5i%2s%10.5f%1s%10.5f%1s%10.5f%10s' ;
for i=1:Node_N
    fprintf(fileID,formatSpec1,'            <node
id="',i,'">',N(i,1),',',N(i,2),',',N(i,3),'</node>');
    fprintf(fileID,'\r\n');
end
fprintf(fileID,'        </Nodes>\r\n');

formatSpec2 = '%10s%5i%2s%6i%1s%6i%1s%6i%1s%6i%1s%6i%1s%6i%1s%6i%1s%6i%10s'
;
fprintf(fileID,'        <Elements type="hex8" mat="1" elset="Part1">\r\n');
for i=1:Elem_N
    fprintf(fileID,formatSpec2,'            <elem
id="',i,'">',T(i,1),',',T(i,2),',',T(i,3),',',T(i,4),',',T(i,5),',',T(i,6),
',',T(i,7),',',T(i,8),'</elem>');
    fprintf(fileID,'\r\n');
end
fprintf(fileID,'        </Elements>\r\n');
fprintf(fileID,'    </Geometry>\r\n');
```

```
%% Filling top layer's Displacement field Data into '###.feb' file

%--------------------------------------------------------------------------
NodeTop_N   = (nx+1)*(ny+1);
NodeBot_ID  = (1:nz+1:Node_N);
NodeTop_ID  = (nz+1:nz+1:Node_N);
%--------------------------------------------------------------------------

fprintf(fileID,'    <Boundary>\r\n');

formatSpec3 = '%10s%1i%2s' ;
fprintf(fileID,'        <fix bc="xyz">\r\n');
for i=1:NodeTop_N
    fprintf(fileID,formatSpec3,'            <node
id="',NodeBot_ID(i),'"/>');
    fprintf(fileID,'\r\n');
end
fprintf(fileID,'        </fix>\r\n');

formatSpec4 = '%10s%1i%2s%8.5f%7s' ;
fprintf(fileID,'        <prescribe bc="x" lc="1">\r\n');
for i=1:NodeTop_N
    fprintf(fileID,formatSpec4,'            <node
id="',NodeTop_ID(i),'">',Disp_ref_vec(i,1),'</node>');
    fprintf(fileID,'\r\n');
end
fprintf(fileID,'        </prescribe>\r\n');

fprintf(fileID,'        <prescribe bc="y" lc="2">\r\n');
for i=1:NodeTop_N
    fprintf(fileID,formatSpec4,'            <node
id="',NodeTop_ID(i),'">',Disp_ref_vec(i,2),'</node>');
    fprintf(fileID,'\r\n');
end
fprintf(fileID,'        </prescribe>\r\n');

fprintf(fileID,'        <prescribe bc="z" lc="3">\r\n');
for i=1:NodeTop_N
    fprintf(fileID,formatSpec4,'            <node
id="',NodeTop_ID(i),'">',Disp_ref_vec(i,3),'</node>');
    fprintf(fileID,'\r\n');
end
fprintf(fileID,'        </prescribe>\r\n');

fprintf(fileID,'    </Boundary>\r\n');


%% Filling Data into '###.feb' file

fprintf(fileID,'    <LoadData>\r\n');
fprintf(fileID,'        <loadcurve id="1" type="smooth">\r\n');
fprintf(fileID,'            <point>0,0</point>\r\n');
fprintf(fileID,'            <point>1,1</point>\r\n');
fprintf(fileID,'        </loadcurve>\r\n');

fprintf(fileID,'        <loadcurve id="2" type="smooth">\r\n');
fprintf(fileID,'            <point>0,0</point>\r\n');
fprintf(fileID,'            <point>1,1</point>\r\n');
fprintf(fileID,'        </loadcurve>\r\n');
```

```
fprintf(fileID,'        <loadcurve id="3" type="smooth">\r\n');
fprintf(fileID,'            <point>0,0</point>\r\n');
fprintf(fileID,'            <point>1,1</point>\r\n');
fprintf(fileID,'        </loadcurve>\r\n');
fprintf(fileID,'    </LoadData>\r\n');

fprintf(fileID,'    <Output>\r\n');
fprintf(fileID,'        <plotfile type="febio">\r\n');
fprintf(fileID,'            <var type="displacement"/>\r\n');
fprintf(fileID,'            <var type="stress"/>\r\n');
fprintf(fileID,'        </plotfile>\r\n');

fprintf(fileID,'        <logfile>\r\n');
fprintf(fileID,'        <node_data data="ux;uy;uz" delim="," 
file="output_displac.txt"/>\r\n');
fprintf(fileID,'        <element_data data="sx;sy;sz;sxy;syz;sxz" delim="," 
file="output_stress.txt"/>\r\n');
fprintf(fileID,'        </logfile>\r\n');

fprintf(fileID,'    </Output>\r\n');

fprintf(fileID,'</febio_spec>\r\n');

fclose(fileID);

%% Run Febio & Continue to Post-Process

disp('-----------------------------------');
disp('Do You Want To Run The Febio File?');
flag_1=input('Yes:1 | No:0 ');
if flag_1==1
    system(fileNm1);
end

disp('-------------------------------------------------------');
disp('Have You Prepared Appropriate "output_stress.txt" File?');
flag_2=input('Yes:1 | No:0 ');
if flag_2==1
    run PostFEBio_Output.m
end

disp('-------------------------------------');
disp('Do You Want To Run The PostView File?');
flag_3=input('Yes:1 | No:0 ');
if flag_3==1
    system(fileNm2);
end
```

2. Appendix2: PostFEBio_Output.m

```
%% Averaging element stress in joint nodes

%--------------------------------------------
PostFebio = importdata('output_stress.txt', ',');
%--------------------------------------------

NodeTop_N = (nx+1)*(ny+1);
NodeTop_ID      = (nz+1:nz+1:Node_N);
NodeTop_Ele     = zeros(NodeTop_N,4);
NodeTop_Ele_N   = zeros(NodeTop_N,1);

EleTop_N        = nx*ny;
EleTop_ID       = (nz:nz:Elem_N);

for i=1:NodeTop_N
    s=1;
    for j=1:EleTop_N
        for k=5:8
            if NodeTop_ID(i) == T(EleTop_ID(j),k)
                NodeTop_Ele(i,s) = EleTop_ID(j);
                s = s + 1;
            end
        end
    end
    NodeTop_Ele_N(i) = s-1;
end

Strs_feb_vec = zeros(NodeTop_N,6);
for i=1:NodeTop_N
    for j=1:NodeTop_Ele_N(i)
        Strs_feb_vec(i,1)= Strs_feb_vec(i,1)+
PostFebio.data(NodeTop_Ele(i,j),2)/ NodeTop_Ele_N(i);
        Strs_feb_vec(i,2)= Strs_feb_vec(i,2)+
PostFebio.data(NodeTop_Ele(i,j),3)/ NodeTop_Ele_N(i);
        Strs_feb_vec(i,3)= Strs_feb_vec(i,3)+
PostFebio.data(NodeTop_Ele(i,j),4)/ NodeTop_Ele_N(i);
        Strs_feb_vec(i,4)= Strs_feb_vec(i,4)+
PostFebio.data(NodeTop_Ele(i,j),5)/ NodeTop_Ele_N(i);
        Strs_feb_vec(i,5)= Strs_feb_vec(i,5)+
PostFebio.data(NodeTop_Ele(i,j),6)/ NodeTop_Ele_N(i);
        Strs_feb_vec(i,6)= Strs_feb_vec(i,6)+
PostFebio.data(NodeTop_Ele(i,j),7)/ NodeTop_Ele_N(i);
    end
end

nnx = nx + 1;
nny = ny + 1;
Strs_feb_mat = zeros(nny,nnx,6);
s = 1;
for j=1:nnx
    for i=1:nny
        Strs_feb_mat(i,j,1) = Strs_feb_vec(s,1);     % Sxx
        Strs_feb_mat(i,j,2) = Strs_feb_vec(s,2);     % Syy
        Strs_feb_mat(i,j,3) = Strs_feb_vec(s,3);     % Szz
        Strs_feb_mat(i,j,4) = Strs_feb_vec(s,4);     % Sxy
        Strs_feb_mat(i,j,5) = Strs_feb_vec(s,5);     % Syz
```

```
        Strs_feb_mat(i,j,6) = Strs_feb_vec(s,6);     % Sxz
        s = s + 1;
    end
end


%% Transferring node Stress, from deformed coordinate axis to original one

F1 =
scatteredInterpolant(N(NodeTop_ID(:),1)+Disp_ref_vec(:,1),N(NodeTop_ID(:),2
)+Disp_ref_vec(:,2),Strs_feb_vec(:,1));
F2 =
scatteredInterpolant(N(NodeTop_ID(:),1)+Disp_ref_vec(:,1),N(NodeTop_ID(:),2
)+Disp_ref_vec(:,2),Strs_feb_vec(:,2));
F3 =
scatteredInterpolant(N(NodeTop_ID(:),1)+Disp_ref_vec(:,1),N(NodeTop_ID(:),2
)+Disp_ref_vec(:,2),Strs_feb_vec(:,3));
F4 =
scatteredInterpolant(N(NodeTop_ID(:),1)+Disp_ref_vec(:,1),N(NodeTop_ID(:),2
)+Disp_ref_vec(:,2),Strs_feb_vec(:,4));
F5 =
scatteredInterpolant(N(NodeTop_ID(:),1)+Disp_ref_vec(:,1),N(NodeTop_ID(:),2
)+Disp_ref_vec(:,2),Strs_feb_vec(:,5));
F6 =
scatteredInterpolant(N(NodeTop_ID(:),1)+Disp_ref_vec(:,1),N(NodeTop_ID(:),2
)+Disp_ref_vec(:,2),Strs_feb_vec(:,6));

Strs_feb_vec_Int = zeros(NodeTop_N,6);
Strs_feb_vec_Int(:,1) = F1( N(NodeTop_ID(:),1) , N(NodeTop_ID(:),2) );
Strs_feb_vec_Int(:,2) = F2( N(NodeTop_ID(:),1) , N(NodeTop_ID(:),2) );
Strs_feb_vec_Int(:,3) = F3( N(NodeTop_ID(:),1) , N(NodeTop_ID(:),2) );
Strs_feb_vec_Int(:,4) = F4( N(NodeTop_ID(:),1) , N(NodeTop_ID(:),2) );
Strs_feb_vec_Int(:,5) = F5( N(NodeTop_ID(:),1) , N(NodeTop_ID(:),2) );
Strs_feb_vec_Int(:,6) = F6( N(NodeTop_ID(:),1) , N(NodeTop_ID(:),2) );

Strs_feb_mat_Int = zeros(nny,nnx,6);
s = 1;
for j=1:nnx
    for i=1:nny
        Strs_feb_mat_Int(i,j,1) = Strs_feb_vec_Int(s,1);
        Strs_feb_mat_Int(i,j,2) = Strs_feb_vec_Int(s,2);
        Strs_feb_mat_Int(i,j,3) = Strs_feb_vec_Int(s,3);
        Strs_feb_mat_Int(i,j,4) = Strs_feb_vec_Int(s,4);
        Strs_feb_mat_Int(i,j,5) = Strs_feb_vec_Int(s,5);
        Strs_feb_mat_Int(i,j,6) = Strs_feb_vec_Int(s,6);
        s = s + 1;
    end
end


%% Traction force calculation using stress matrix and surface normal vector

mat_X = zeros(nx+1,ny+1);
mat_Y = zeros(nx+1,ny+1);
mat_Z = zeros(nx+1,ny+1);

s=1;
for i=1 : nx+1
    for j=1 : ny+1
```

```
        mat_X(i,j) = N(NodeTop_ID(s),1) + Disp_ref_vec(s,1);
        mat_Y(i,j) = N(NodeTop_ID(s),2) + Disp_ref_vec(s,2);
        mat_Z(i,j) = N(NodeTop_ID(s),3) + Disp_ref_vec(s,3);
        s=s+1;
    end
end

surfnorm( mat_X' , mat_Y' , mat_Z' );
[unitn1,unitn2,unitn3]=surfnorm( mat_X' , mat_Y' , mat_Z' );

Tx=zeros(nx+1,ny+1);
Ty=zeros(nx+1,ny+1);
Tz=zeros(nx+1,ny+1);

for j=1:nnx
    for i=1:nny

Tx(i,j)=Strs_feb_mat_Int(i,j,1)*unitn1(i,j)+Strs_feb_mat_Int(i,j,4)*unitn2(
i,j)+Strs_feb_mat_Int(i,j,6)*unitn3(i,j);

Ty(i,j)=Strs_feb_mat_Int(i,j,4)*unitn1(i,j)+Strs_feb_mat_Int(i,j,2)*unitn2(
i,j)+Strs_feb_mat_Int(i,j,5)*unitn3(i,j);

Tz(i,j)=Strs_feb_mat_Int(i,j,6)*unitn1(i,j)+Strs_feb_mat_Int(i,j,5)*unitn2(
i,j)+Strs_feb_mat_Int(i,j,3)*unitn3(i,j);
    end
end


%% Plot Stress Matrices

plotTool_2(Plot_Stress,Strs_feb_mat,Strs_feb_mat_Int);
% plotTool_2(Plot_Strs,Strs_ref_mat,Strs_feb_mat_Int);
```

3. Appendix3: Connectivity.m

```
function [N,T] = Connectivity(a,b,d,nx,ny,nz)

Nnod = (nx+1)*(ny+1)*(nz+1);
Nele = nx*ny*nz;

N = zeros(Nnod,3);
T = zeros(Nele,8);

dlx = a/nx ; dly = b/ny ; dlz = d/nz;

s = 0;
for i = 1 : nx+1
    for j = 1 : ny+1
        for k = 1 : nz+1
            s = s + 1;
            N(s,1) = (i-1) * dlx;
            N(s,2) = (j-1) * dly;
            N(s,3) = (k-1) * dlz;
        end
    end
end

s = 1;
p = 0;
for i = 1 : nx
    for j = 1 : ny
        for k = 1 : nz
            T(s,1) = p * (nz+1) + (i-1)*(nz+1) + k;
            s = s + 1;
        end
        p = p + 1;
    end
end

T(:,2) = T(:,1) + (ny+1)*(nz+1);
T(:,3) = T(:,2) + (nz+1);
T(:,4) = T(:,1) + (nz+1);
T(:,5) = T(:,1) + 1;
T(:,6) = T(:,2) + 1;
T(:,7) = T(:,3) + 1;
T(:,8) = T(:,4) + 1;

end
```

4. Appendix4: Extraction.m

```
function
[Disp_ref_mat,Trac_ref_mat,Strs_ref_mat,Disp_ref_vec,Trac_ref_vec,Strs_ref_
vec]=Extraction(nx,ny,Plot_Extract)

load tractions_and_displacements.mat
load stress_tensor.mat

NodeTop_N   = (nx+1)*(ny+1);
nnx         = nx+1;
nny         = ny+1;

Disp_ref_mat(:,:,1) = imresize( DX, [ nny nnx ] );
Disp_ref_mat(:,:,2) = imresize( DY, [ nny nnx ] );
Disp_ref_mat(:,:,3) = imresize( DZ, [ nny nnx ] );
Trac_ref_mat(:,:,1) = imresize( TX, [ nny nnx ] );
Trac_ref_mat(:,:,2) = imresize( TY, [ nny nnx ] );
Trac_ref_mat(:,:,3) = imresize( TZ, [ nny nnx ] );
Strs_ref_mat(:,:,1) = imresize( Sxx, [ nny nnx ] );
Strs_ref_mat(:,:,2) = imresize( Syy, [ nny nnx ] );
Strs_ref_mat(:,:,3) = imresize( Szz, [ nny nnx ] );
Strs_ref_mat(:,:,4) = imresize( Sxy, [ nny nnx ] );
Strs_ref_mat(:,:,5) = imresize( Syz, [ nny nnx ] );
Strs_ref_mat(:,:,6) = imresize( Sxz, [ nny nnx ] );

Disp_ref_vec = zeros(NodeTop_N,3);
Trac_ref_vec = zeros(NodeTop_N,3);
Strs_ref_vec = zeros(NodeTop_N,6);
s = 1;
for j=1:nnx
    for i=1:nny
        Disp_ref_vec(s,1) = Disp_ref_mat(i,j,1);
        Disp_ref_vec(s,2) = Disp_ref_mat(i,j,2);
        Disp_ref_vec(s,3) = Disp_ref_mat(i,j,3);
        Trac_ref_vec(s,1) = Trac_ref_mat(i,j,1);
        Trac_ref_vec(s,2) = Trac_ref_mat(i,j,2);
        Trac_ref_vec(s,3) = Trac_ref_mat(i,j,3);
        Strs_ref_vec(s,1) = Strs_ref_mat(i,j,1);
        Strs_ref_vec(s,2) = Strs_ref_mat(i,j,2);
        Strs_ref_vec(s,3) = Strs_ref_mat(i,j,3);
        Strs_ref_vec(s,4) = Strs_ref_mat(i,j,4);
        Strs_ref_vec(s,5) = Strs_ref_mat(i,j,5);
        Strs_ref_vec(s,6) = Strs_ref_mat(i,j,6);
        s = s + 1;
    end
end

plotTool_1(Plot_Extract,Disp_ref_mat,Strs_ref_mat);

end
```