Appendix Report

Appendix A: Procedure for cutting process

```
clc
clear all

% Defining 3 points that define a plane in 3D-space
lm1 = [-106.2,122.5,1075];
lm2 = [88.6,127.7,1080];
lm3 = [-5.6,147.4,999.8];

% I want to rotate pont lm3 around the axis between lm1 and lm2
rotation = -96;  %Degrees
theta = degtorad(rotation); %radians

%Defining the rotation axis between lm1 and lm2
rot_axis = [lm2(1)-lm1(1), lm2(2) - lm1(2), lm2(3) - lm1(3)];

% Make a unit vector of the rotation axis
urot = rot_axis/norm(rot_axis);

%split up direction vector
u = urot(1);
v = urot(2);
w = urot(3);

%Point that the axis is going through
a = lm2(1);
b = lm2(2);
c = lm2(3);

% Splitting up the point of rotation
x =lm3(1);
y =lm3(2);
z =lm3(3);

% Defining the rotation matrix and define new lm3
lm3_new = [(a*(v^2+w^2)-u*(b*v+c*w-u*x-v*y-w*z))*(1-
cos(theta))+x*cos(theta)+(-c*v+b*w-w*y+v*z)*sin(theta);...
    (b*(u^2+w^2)-v*(a*u+c*w-u*x-v*y-w*z))*(1-
cos(theta))+y*cos(theta)+(c*u-a*w+w*x-u*z)*sin(theta);...
    (c*(u^2+v^2)-w*(a*u+b*v-u*x-v*y-w*z))*(1-
cos(theta))+z*cos(theta)+(-b*u+a*v-v*x+u*y)*sin(theta)];
```

Plotting to check if correct

```
% Aquire normal vector for initial plane
vec1_old = [lm3(1)-lm2(1), lm3(2) - lm2(2), lm3(3) - lm2(3)];
vec2_old = [lm3(1)-lm1(1), lm3(2) - lm1(2), lm3(3) - lm1(3)];
norm_old = cross(vec1_old,vec2_old);
x_norm_old = [lm3(1),norm_old(1)];
y_norm_old = [lm3(2),norm_old(2)];
z_norm_old = [lm3(3),norm_old(3)];

% Aquire normal vector for new plane
```

```
vec1_new = [lm3_new(1)-lm2(1), lm3_new(2) - lm2(2), lm3_new(3) -
 lm2(3)];
vec2_new = [lm3_new(1)-lm1(1), lm3_new(2) - lm1(2), lm3_new(3) -
 lm1(3)];
norm_new = cross(vec1_new,vec2_new);
x_norm_new = [lm3_new(1),norm_new(1)];
y_norm_new = [lm3_new(2),norm_new(2)];
z_norm_new = [lm3_new(3),norm_new(3)];

plane_initial = [lm1', lm2', lm3']; % Inital plane
plane_rotated = [lm1', lm2', lm3_new]; % New rotated plane
%
% figure()
% fill3(plane_initial(1,:),plane_initial(2,:),plane_initial(3,:),'r')
% hold on
% fill3(plane_rotated(1,:),plane_rotated(2,:),plane_rotated(3,:),'c')
% hold on
% plot3(x_norm_old,y_norm_old,z_norm_old)
% hold on
% plot3(x_norm_new,y_norm_new,z_norm_new)
% grid on
% xlabel('X')
% ylabel('Y')
% zlabel('Z')

% For blender, a point is needed where the plane goes through and the
% normal, output these here
norm_new = norm_new/norm(norm_new)
```

Published with MATLAB® R2018b

Appendix B: Bone remodelling model A

```
C Subroutine for changing the bone density based on the loading history
C Stress and E-modulus is taken every step, based on this the E-modulus is altered

C October 2018
C Nick Wassenberg

C Algorithm-equations are based on model from Shitong Luo,1 Xingquan Shen,1 Xin Bai,1
Jing Bai,2 Jianning Han,1 and Yu Shang1
C as presented in their paper: 'Validation of material algorithms for femur remodelling
using medical imaging data

C First need to define the initial conditions
C We can only update the E-modulus after we obtain strain from the first step
C Do this with subroutine SDVINI



      SUBROUTINE SDVINI(STATEV,COORDS,NSTATV,NCRDS,NOEL,NPT,
     1 LAYER,KSPT)


C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION STATEV(NSTATV),COORDS(NCRDS)

C Define the initial E-modulus
      STATEV(1) = 18070


C     Define the initial Stress Stimulus signals for each loading scenario (x3)
      STATEV(2) = 0
      STATEV(3) = 0
      STATEV(4) = 0
C     Define initial stimulus signal

      STATEV(5) = 0

C     Testing
      statev(6) = 0
      statev(7) = 0
      statev(8) = 0
      statev(9) = 0
      statev(10) = 0
```

```
      statev(11) = 0
      statev(12) = 0
      statev(13) = 0
      statev(14) = 0
      statev(15) = 0
      statev(16) = 0
      statev(17) = 0
      statev(18) = 0




      RETURN
      END


C Now that we have the initial variables, UMAT subroutine is used to define the bone
  material parameters
C The E-moudulus is updated after every 3rd loading step (1 day)


      SUBROUTINE UMAT(STRESS,STATEV,DDSDDE,SSE,SPD,SCD,
     1 RPL,DDSDDT,DRPLDE,DRPLDT,
     2 STRAN,DSTRAN,TIME,DTIME,TEMP,DTEMP,PREDEF,DPRED,bone,
     3 NDI,NSHR,NTENS,NSTATV,PROPS,NPROPS,COORDS,DROT,PNEWDT,
     4 CELENT,DFGRD0,DFGRD1,NOEL,NPT,LAYER,KSPT,JSTEP,KINC)
C

      INCLUDE 'ABA_PARAM.INC'

C Creating dimension arrays of properties

      CHARACTER*80 bone
      DIMENSION STRESS(NTENS),STATEV(NSTATV),
     1 DDSDDE(NTENS,NTENS),
     2 STRAN(NTENS),DSTRAN(NTENS),
     3 PROPS(NPROPS),s(3),ps(3)

C Initialising variables

      INTEGER trif,tr,ti,tf,lstr

      REAL modulus, nu, bulkmod, shearmod, lambda, rl1, rl2,rl3,m,B,
     1 wzero, dE, maxmag, tstep, maxprin, minprin

C nu is the poissons ratio, entered as mechanical constant in Abaqus
C From E and u, calculate bulkmodulus, shearmodulus and lambda

      modulus = statev(1)

      nu = props(1)
      bulkmod = modulus/(3*(1-2*nu))
      shearmod = modulus/(2*(1+nu))
      lambda = bulkmod-2*shearmod/3

C Defining the C tensor, a 6x6 matrix (ni by ni)

C First put lambda on all locations in top left 3x3 matrix, then replace diagonals
C with approproiate terms
```

```
      do k1 =1,ndi
       do k2=1,ndi
          ddsdde(k2,k1) = lambda
       enddo

       ddsdde(k1,k1) = 2*shearmod+lambda
      enddo

C The bottom left and top right 3x3 submatrix are filled with zeros
C This is default, not needed to define this
C All that is left is the bottom right 3x3 submatrix.
C This is filled with values for u on the diagonal

      do k1=ndi+1,ntens
          ddsdde(k1,k1) = shearmod
      enddo

C Initalize zero stress at start
      do k1=1,ntents
          stress(k1)=0.0
      enddo

C Now that C matrix is known, define stress
C This is a 6x1 matrix, defined as current stress + increase in stress
C increase in stress is calculated by multypling Jacobian with increase in strain
      do k1=1,ntens
          do k2=1,ntens
          stress(k2) = stress(k2) + ddsdde(k2,k1)*(stran(k1)+dstran(k1))
          enddo
      enddo

C Writing the C matrix  for checking in debugging

C Top left 3x3 matrix
C      WRITE(7,*) 'IND1-1',ddsdde(1,1)
C      WRITE(7,*) 'IND1-2',ddsdde(1,2)
C      WRITE(7,*) 'IND1-3',ddsdde(1,3)

C      WRITE(7,*) 'IND2-1',ddsdde(2,1)
C      WRITE(7,*) 'IND2-2',ddsdde(2,2)
C      WRITE(7,*) 'IND2-3',ddsdde(2,3)

C      WRITE(7,*) 'IND3-1',ddsdde(3,1)
C      WRITE(7,*) 'IND3-2',ddsdde(3,2)
C      WRITE(7,*) 'IND3-3',ddsdde(3,3)

C Bottom right 3x3 matrix
C      WRITE(7,*) 'IND4,4',ddsdde(4,4)
C      WRITE(7,*) 'IND4,5',ddsdde(4,5)
C      WRITE(7,*) 'IND4,6',ddsdde(4,6)

C      WRITE(7,*) 'IND5,4',ddsdde(5,4)
C      WRITE(7,*) 'IND5,5',ddsdde(5,5)
C      WRITE(7,*) 'IND5,6',ddsdde(5,6)

C      WRITE(7,*) 'IND6,4',ddsdde(6,4)
C      WRITE(7,*) 'IND6,5',ddsdde(6,5)
C      WRITE(7,*) 'IND6,6',ddsdde(6,6)
```

```
C Writing material values for checking in debugging
C      WRITE(7,*) 'nu',nu
C      WRITE(7,*) 'modulus', modulus
C      WRITE(7,*) 'bulkodulus', bulkmod
C      WRITE(7,*) 'shearmodulus', shearmod
C      WRITE(7,*) 'lambda', lambda
C      WRITE(7,*) 'porosity', statev(1)
C      WRITE(7,*) 'NSTATV', NSTATV

C Bone remodeling algorithm
C sprinc obtains maximal principal values
C Apply max strain according to step 1,2 or 3.

          rl1 = 300
          rl2 = 5
          rl3 = 3000
          m = 4
          B = 400000
          wzero = 0.0025


C boundhigh = 0.0198
C boundlow =  0.0162

          if (noel.lt.9999999999) then
              do k1=1,ntens
                  s(k1)=stress(k1)
                  statev(15) = s(1)
                  statev(16) = s(2)
                  statev(17) = s(3)
                  statev(18) = s(4)
                  statev(19) = s(5)
                  statev(20) = s(6)
              enddo

              lstr = 1
              call sprinc(s,ps,lstr,ndi,nshr)

              if (abs(ps(1)).ge.abs(ps(2))) then
                  maxmag = ps(1)
              else
                  maxmag = ps(2)
              endif


              tstep = abs((real(jstep))/3-int(jstep/3))
              if ((tstep.gt.0.2).and.(tstep.lt.0.5)) then
                  statev(2) = rl1*(abs(maxmag)**m)
                  statev(6) = abs(maxmag)
                  statev(7) = maxmag
              else if ((tstep.gt.0.5).and.(tstep.lt.0.8)) then
                  statev(3) = rl2*(abs(maxmag)**m)
                  statev(9) = abs(maxmag)
                  statev(10) = maxmag
              else if (tstep.lt.0.2) then
                  statev(4) = rl3*(abs(maxmag)**m)
                  statev(12) = abs(maxmag)
                  statev(13) = maxmag
```

```
            endif

            strtot  = statev(2) + statev(3) + statev(4)

            w = ((strtot)**(1/m))/modulus
            statev(5) = w

            if (w.ge.0.00275) then
                dE = B*(statev(5)-0.00275)
            else if ((w.lt.0.00275).and.(w.gt.0.00225))
     1          then
                dE = 0
            else if (w.le.0.00225) then
                    dE = B*(w-0.00225)

            endif

            if ((jstep.ne.1).and.(tstep.lt.0.2)) then
                modulus = modulus + dE
                statev(1) = modulus
            endif

            if (statev(1).gt.18070) then
                statev(1) = 18070
            endif

            if (statev(1).lt.50) then
                statev(1) = 50
            endif

            if (modulus.gt.13028) then
                por = 1-(modulus**(1/5.74)/(23440**(1/5.74)))
            else
                por = 1-(modulus**(1/1.33)/(14927**(1/1.33)))
            endif

            statev(18) = por

        endif




        RETURN
        END
```

Appendix C: Bone remodelling model B

```
C Subroutine for changing the bone density based on the loading history
C Strain is taken after each step and porisity is calculated based on equations.
C Based on the E-modulus, the BMD is altered, which can be used to identify stress shielding regions.

C October 2018
C Nick Wassenberg

C Algorithm-equations are based on mechanistic model from J. Hazelwood, R. bruce Martin, Mark M Rashid and Juan J. Rodrigo
C As presented in their paper 'A mechanistic model for internal bone remodeling exhibits different dynamic responses in disuse and overload'



C Sharevars to define global array value to average porosity and stimulus signal

      module shareVars
          Dimension por_save_array(10000)
          Dimension dp_save_array(10000)
      end module shareVars

C First need to define the initial conditions
C We can only update the E-modulus after we obtain strain from the first step
C Do this with subroutine SDVINI

      SUBROUTINE SDVINI(STATEV,COORDS,NSTATV,NCRDS,NOEL,NPT,
     1 LAYER,KSPT)

      use shareVars
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION STATEV(NSTATV),COORDS(NCRDS)

C Define the initial porosity
      STATEV(1) = 0.044321

C Define minimal strain for step 1,2 en 3 (3 loads are used)
      STATEV(2) = 0
      STATEV(3) = 0
      STATEV(4) = 0


C Mechanical stimulus: zero in beginning
       STATEV(5) = 0

C Damage, initial value in beginning
        STATEV(6) = 0.03662944

C Amount of bone building is zero in beginning: Amount of BMU for building
        STATEV(7) = 0

C Amount of bone removal is zero in beginning: Amount of BMU for removing
       STATEV(8) = 0


C The activation frequency of the BMU: How many BMU's are crteaed per time unit
```

```
C We want to make sure we define this value for whole timeduration (55) in the remodelling cycle

      DO k=9,NSTATV
          statev(k) = 0.00670
      ENDDO




      RETURN
      END


C Now that we have the initial variables, UMAT subroutine is used to define the bone material parameters
C The E-moudulus is updated after every loading step


      SUBROUTINE UMAT(STRESS,STATEV,DDSDDE,SSE,SPD,SCD,
     1 RPL,DDSDDT,DRPLDE,DRPLDT,
     2 STRAN,DSTRAN,TIME,DTIME,TEMP,DTEMP,PREDEF,DPRED,bone,
     3 NDI,NSHR,NTENS,NSTATV,PROPS,NPROPS,COORDS,DROT,PNEWDT,
     4 CELENT,DFGRD0,DFGRD1,NOEL,NPT,LAYER,KSPT,JSTEP,KINC)
C
      use shareVars




      INCLUDE 'ABA_PARAM.INC'

      integer R,Nel
      real dec



C Creating dimension arrays of properties

      CHARACTER*80 bone
      DIMENSION STRESS(NTENS),STATEV(NSTATV),
     1 DDSDDE(NTENS,NTENS),
     2 STRAN(NTENS),DSTRAN(NTENS),
     3 PROPS(NPROPS),s(3),ps(3),
     4 por_save_matrix(200,200), por_summed_matrix(200,200),
     5 por_avg_matrix(200,200), stress_save(10,10,10),
     6 por_avg_array(40000),
     7 por_summed_array(40000),
     8 dp_save_matrix(200,200), dp_summed_matrix(200,200),
     5 dp_avg_matrix(200,200),
     6 dp_avg_array(40000),
     7 dp_summed_array(40000)

C Initialising variables

      INTEGER trif,tr,ti,tf,lstr,index

      REAL kb,kc,kd,kr,nf,nr,nfexist,nrexist,doo,sa,modulus,nu,
     1 tstep,minprin,rl1,rl2,rl3,phi,shearmod,bulkmod,lambda,samax,
     2 q,pi,fs,dt,facur,famax1,famax2,fao,rc,rh,dexist,area,ddotform,
```

```
     3 ddotrep,dcurrent,fadivsa,fa,ab,ac,qb,qc,phc,qnet,dp,maxprin,
     4 maxmag, weightA, weightB, wd_x, wdt_y, avg_factor,dist, infl,
     5 dp_avg

C R is the parameter defining the k-values
C Nel determines the amount of elements (In this case only 100, since we apply it to the cubic shape model)
C (100x100 elements = 40.000 elements)
C Dec is the value for D (Determines Spatical Decay)
      R = 5
      Nel = 200
      dec = 0.06

C Lt : less then
C Defining equations for cortical and cancellous bone (relation with porosity)


      IF (statev(1).lt.0.097267787) then
          modulus = 23440*(1-statev(1))**5.74

      ELSE
          modulus = 14927*(1-statev(1))**1.33

      ENDIF



C nu is the poissons ratio, entered as mechanical constant in Abaqus
C From E and u, calculate bulkmodulus, shearmodulus and lambda
      nu = props(1)
      bulkmod = modulus/(3*(1-2*nu))
      shearmod = modulus/(2*(1+nu))
      lambda = bulkmod-2*shearmod/3


C Defining the C tensor, a 6x6 matrix (ni by ni)

C First put lambda on all locations in top left 3x3 matrix, then replace diagonals
C with approproiate terms

      do k1 =1,2
       do k2=1,2
          ddsdde(k2,k1) = lambda
       enddo

       ddsdde(k1,k1) = 2*shearmod+lambda
      enddo

C The bottom left and top right 3x3 submatrix are filled with zeros
C This is default, not needed to define this
C All that is left is the bottom right 3x3 submatrix.
C This is filled with values for u on the diagonal

      do k1=3,4
          ddsdde(k1,k1) = shearmod
      enddo

C Initalize zero stress at start
      do k1=1,4
```

```
172           stress(k1)=0.0
173       enddo
174
175
176
177
178 C Now that C matrix is known, define stress
179 C This is a 6x1 matrix, defined as current stress + increase in stress
180 C increase in stress is calculated by multypling Jacobian with increase in strain
181       do k1=1,4
182           do k2=1,4
183           stress(k2) = stress(k2) + ddsdde(k2,k1)*(stran(k1)+dstran(k1))
184
185           enddo
186       enddo
187
188 C Writing the C matrix  for checking in debugging
189
190 C Top left 3x3 matrix
191 C      WRITE(7,*) 'IND1-1',ddsdde(1,1)
192 C      WRITE(7,*) 'IND1-2',ddsdde(1,2)
193 C      WRITE(7,*) 'IND1-3',ddsdde(1,3)
194
195 C      WRITE(7,*) 'IND2-1',ddsdde(2,1)
196 C      WRITE(7,*) 'IND2-2',ddsdde(2,2)
197 C      WRITE(7,*) 'IND2-3',ddsdde(2,3)
198
199 C      WRITE(7,*) 'IND3-1',ddsdde(3,1)
200 C      WRITE(7,*) 'IND3-2',ddsdde(3,2)
201 C      WRITE(7,*) 'IND3-3',ddsdde(3,3)
202
203 C Bottom right 3x3 matrix
204 C      WRITE(7,*) 'IND4,4',ddsdde(4,4)
205 C      WRITE(7,*) 'IND4,5',ddsdde(4,5)
206 C      WRITE(7,*) 'IND4,6',ddsdde(4,6)
207
208 C      WRITE(7,*) 'IND5,4',ddsdde(5,4)
209 C      WRITE(7,*) 'IND5,5',ddsdde(5,5)
210 C      WRITE(7,*) 'IND5,6',ddsdde(5,6)
211
212 C      WRITE(7,*) 'IND6,4',ddsdde(6,4)
213 C      WRITE(7,*) 'IND6,5',ddsdde(6,5)
214 C      WRITE(7,*) 'IND6,6',ddsdde(6,6)
215
216 C Writing material values for checking in debugging
217 C      WRITE(7,*) 'nu',nu
218 C      WRITE(7,*) 'modulus', modulus
219 C      WRITE(7,*) 'bulkodulus', bulkmod
220 C      WRITE(7,*) 'shearmodulus', shearmod
221 C      WRITE(7,*) 'lambda', lambda
222 C      WRITE(7,*) 'porosity', statev(1)
223 C      WRITE(7,*) 'NSTATV', NSTATV
224
225 C Bone remodeling algorithm
226 C sprinc obtains maximal principal values
227 C Apply max strain according to step 1,2 or 3.
228
229
230
```

```
231
232           if (noel.lt.9999999999) then
233               do k1=1,ntens
234                   s(k1)=stran(k1)+dstran(k1)
235               enddo
236               lstr =2
237
238               call sprinc(s,ps,lstr,ndi,nshr)
239               if (abs(ps(1)).ge.abs(ps(2))) then
240                   maxmag = ps(1)
241               else
242                   maxmag = ps(2)
243               endif
244
245  C Loading rates for 3 load cases:
246               rl1 = 3000
247               rl2 = 112
248               rl3 = 1
249
250  C Determine maxmag based on the step
251           tstep = abs((real(jstep))/2-int(jstep/2))
252               if ((tstep.gt.0.2).and.(tstep.lt.0.7)) then
253                   statev(2) = 2*maxmag
254               else if (tstep.lt.0.1) then
255                   statev(3) = 2*maxmag
256               endif
257
258
259  ! Relative weighting between the 2 loading scenarios (only applied for validation step)
260   !              weightA = 0.5
261   !              weightB = 1-weightA
262
263
264
265
266
267  C Exponent for damage formation equation
268               q=4
269
270  C Specify pi, repair speciality factor and time step
271               pi = 3.1415926535897932385
272               fs = 5.0
273               dt = 8
274
275
276  C Current day activation frequency (the population amount of refilling and resorbing bmu)
277               facur = statev(9)
278  C The maximum activation frequency (not more bmu can be present)
279               famax1 = 0.50
280               famax2 = 1.15
281  C The maximum specific area
282               samax = 4.1905
283  C The activation frequency at equilibirum (very small bone modelling)
284               fao = 0.00670
285  C Area constants
286               rc = 0.095
287               rh = 0.020
```

```
289 C Damage in equilibirum is low (doo). existing damage at beginning is the same as
      equilibirum
290               doo=0.03662944
291               dexist = statev(6)
292
293 C Coefficients for damage formation equation
294 C              kd= 1088
295               kd= 100000
296
297               kr= -1.6
298               kb= 65000000000
299               kc = 0.000000016000
300
301
302
303
304
305
306 C Tr = Time needed for resoptiv phase
307 C Ti = time for reversal phase of modelling cycle
308 C tf = Time for refilling phase of modelling cycle
309 C trif = Total time for remodelling
310
311
312               tr = 3
313               ti = 1
314               tf = 8
315               trif = tr+ti+tf
316
317 C Define the amount of BMU at the start
318 C This is equal to the activation frequency (at the start) x time of modelling cycle
319
320               if(jstep.eq.1) then
321                   statev(7) = fao*tf
322                   statev(8) = fao*tr
323               endif
324
325 C Put the amount of BMU (refilling and resobring) into a variable
326               nfexist = statev(7)
327               nrexist = statev(8)
328
329
330 C Update porosity after 1 loading cycle (so after step 3 everytime)
331 C Calculate damage potential
332 C Calculate rate of damage formation
333 C Calculate rate of damage repair
334 C Calculate current day damage
335 C Store current day damage as state variable
336
337
338               if ((jstep.ne.1).and.(tstep.lt.0.2)) then
339
340                   phi = weightA*((abs(statev(2)))**q)*rl1+weightB
341      1                *((abs(statev(3)))**q)*rl2
342
343                   statev(5) = phi
344
345                   if (statev(1).le.0.20) then
346                       area = pi*rc**2
```

```
347                   else
348                       area=0.5*pi*rc**2
349                   endif
350
351                   ddotform = kd*phi
352                   ddotrep = dexist*facur*area*fs
353
354                   dcurrent = dexist+(ddotform-ddotrep)*dt
355
356                   if (dcurrent.ge.1000000) then
357                       dcurrent = 1000000
358                   endif
359
360
361                   statev(6) = dcurrent
362
363
364   C Update the current activation frequency
365   C Calculate the specific surface area
366
367                    fadivsa=(fao*(famax1/samax))/(fao+(famax1-fao)
368        1              *exp(kr*famax1*(dcurrent-doo)/doo))
369
370                    if (dcurrent.lt.doo) then
371                       fadivsa=(fao/samax)*(dcurrent/doo)
372                    endif
373
374                    sa=((((28.8*statev(1)-101)*statev(1)+134)*statev(1)
375        1                   -93.9)*statev(1)+32.3)*statev(1)
376
377                    if (phi.lt.0.000000000005) then
378                     fadivsa=fadivsa+(famax2/samax)/(1+exp(kb*(phi-kc)))
379                    endif
380
381                    fa=fadivsa*sa
382                    statev(100) = fa
383   C Update the activation frequency of previous time steps. Than the activation
        frequency of the new time step can then be compared
384   C to the activation frequency of the previous time step, to determine the increase.
385   C Day 1 means 1 day ago, day 6 means 6 days ago etc.
386   C One we obtain a value for a new step, this is added to day 1 (statev(9)). Then
        every other value needs to move to one day furthery away
387
388                   do k1 = trif+9,10,-1
389                       statev(k1) = statev(k1-1);
390                   enddo
391
392                   statev(9) = fa
393
394   C If a low porosity is present, other area is specified. This changes bone filling
395   C properties
396                   ac = area
397
398                   if (statev(1).le.0.20) then
399                       area = pi*(rc**2-rh**2)
400                   endif
401
402                   ab = area
403
```

```
404 C Amount of bone added per day per BMU
405                   qb = ab/tf
406                   statev(90) = qb
407
408 C Amount of bone removed per day per BMU
409                   qc = ac/tr
410                   statev(91) = qc
411
412 C Less bone remodelling on trabecular surfaces which are not enough loaded
413 C                   if (statev(1).gt.0.20) then
414                       if (phi.lt.0.000000000005) then
415                       qb = (0.5+0.5*phi/0.000000000005)*ab/tf
416                     endif
417 C               endif
418
419 C Calculating the amount of refilling BMU for current day
420               nf=nfexist+(statev(tr+ti+9)-statev(trif+9))*dt
421
422
423 C Calculating the amount of resorbning BMU for current day
424               nr=nrexist+(statev(9)-statev(tr+9))*dt
425
426 C Make sure these amounts can not be equal to zero=/0
427               if (nf.lt.0.0) then
428                   nf=0.0
429               endif
430               if (nr.lt.0.0) then
431                   nr=0.0
432       endif
433
434                statev(92) = nf
435                statev(92) = nr
436 C Store the amount of BMU's as statevector
437
438               statev(7)=nf
439               statev(8)=nr
440
441
442 C Calculate netto ammount of bone added for the day
443
444               phc=0.04432132964
445               qnet=(qb*nf)-(qc*nr)
446               if (statev(1).le.0.20) then
447                   qnet=(qb*nf)-(1-phc)*(qc*nr)
448               endif
449
450               statev(94)  = (qb*nf)
451               statev(95)  = (qc*nr)
452               statev(96) = qnet
453
454 C Update the porosity
455
456               dp=-qnet*dt
457 !              statev(1)=statev(1)+dp
458
459 C The following code is to average the stimulus signal for every element
460 C Based on the surrouding elements
461 C We can identify 8 regions in this case: middle, top-left, top-middle, top-right,
      left-middle,
```

```
462  C middle,middle, middle right, bottom left, bottom middle, bottom right.
463
464                    ! Save in array based on location of element number
465                    dp_save_array(noel) = dp
466
467                    ! Convert from matrix to array (for individual stimulus signal)
468                    do k1 =1,Nel
469                        do k2=1,Nel
470                            index = k2+((k1-1)*Nel)
471                            dp_save_matrix(k2,k1) = dp_save_array(index)
472                        enddo
473                    enddo
474
475                  !! Convert from matrix to array (for summed stimulus signal)
476                   do k1=1,Nel
477                       do k2=1,Nel
478                           dp_summed_matrix(k2,k1) = 0
479                       enddo
480                   enddo
481                   ! Convert from matrix to array (for avg stimulus signal)
482                   do k1=1,Nel
483                       do k2=1,Nel
484                           dp_avg_matrix(k2,k1) = 0
485                       enddo
486                   enddo
487
488
489  C              Creating summed matrix of dp
490                   do p1=1,Nel
491                       do p2 = 1,Nel
492                          IF (p1.gt.R .AND. p2.gt.R. AND.
493       1                      p1.le.(Nel-R). and. p2.le.(Nel-R)) then ! inner side of the
          cube
494
495                           avg_factor = ((R*2)+1)*((R*2)+1)
496                           do k1 =-R,R
497                               do k2=-R,R
498                                       dist = sqrt((k1**2) +(k2**2))
499                                       infl = exp(-dist/dec)
500                                       dp_summed_matrix(p2,p1) =
501       1                               dp_summed_matrix(p2,p1) +
502       2                               dec*dp_save_matrix(p2+k2,p1+k1) ! Sum every
          element
503                                       ! with the values of the surrounding elements
504                               enddo
505                           enddo
506                                   dp_avg_matrix(p2,p1) =
507       1                           dp_summed_matrix(p2,p1)/
508       2                           avg_factor
509
510
511                          else if(p2.gt.(Nel-R)) then ! Here we define the left side of
                            the cube
512                           if(p1.lt.R+1) then ! top-left corner
513                               wd_x = Nel-p2
514                               wd_y = p1-1
515                               avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
516                                   do k1 = -wd_y,R
517                                       do k2 = -R,wd_x
```

```
519                                     infl = exp(-dist/dec)
520                                        dp_summed_matrix(p2,p1) =
521     1                                  dp_summed_matrix(p2,p1) +
522     2                                  dec*dp_save_matrix(p2+k2,p1+k1)
523                                     enddo
524                                  enddo
525                                  dp_avg_matrix(p2,p1) =
526     1                            dp_summed_matrix(p2,p1)/
527     2                            avg_factor
528
529                              else if(p1.ge.(R+1). AND. p1.le.(Nel-R)) then  ! middle-
                            left side
530                                  wd_x = Nel-p2
531                                  avg_factor = ((R*2)+1)*((wd_x+R)+1)
532                                     do k1 = -R,R
533                                        do k2 = -R,wd_x
534                                        dist = sqrt((k1**2) +(k2**2))
535                                        infl = exp(-dist/dec)
536                                           dp_summed_matrix(p2,p1) =
537     1                                     dp_summed_matrix(p2,p1) +
538     2                                     dec*dp_save_matrix(p2+k2,p1+k1)
539                                        enddo
540                                     enddo
541                                     dp_avg_matrix(p2,p1) =
542     1                               dp_summed_matrix(p2,p1)/
543     2                               avg_factor
544                                 else if(p1.gt.(Nel-R)) then ! bottom-left corner
545                                     wd_x = Nel-p2
546                                     wd_y = Nel-p1
547                                     avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
548                                        do k1 = -R,wd_y
549                                           do k2 = -R,wd_x
550                                           dist = sqrt((k1**2) +(k2**2))
551                                           infl = exp(-dist/dec)
552                                              dp_summed_matrix(p2,p1) =
553     1                                        dp_summed_matrix(p2,p1) +
554     2                                        dec*dp_save_matrix(p2+k2,p1+k1)
555                                           enddo
556                                        enddo
557                                        dp_avg_matrix(p2,p1) =
558     1                                  dp_summed_matrix(p2,p1)/
559     2                                  avg_factor
560
561                                 endif
562                                 else if(p2.lt.(R+1)) then ! Here we define the right side
                            of the cube
563                                 if(p1.lt.(R+1)) then ! top-right corner
564                                     wd_x = p2-1
565                                     wd_y = p1-1
566                                     avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
567                                        do k1 = -wd_y,R
568                                           do k2 = -wd_x,R
569                                           dist = sqrt((k1**2) +(k2**2))
570                                           infl = exp(-dist/dec)
571                                              dp_summed_matrix(p2,p1) =
572     1                                        dp_summed_matrix(p2,p1) +
573     2                                        dec*dp_save_matrix(p2+k2,p1+k1)
574                                           enddo
575                                        enddo
```

```
576                                 dp_avg_matrix(p2,p1) =
577       1                         dp_summed_matrix(p2,p1)/
578       2                         avg_factor
579
580                             else if(p1.ge.(R+1). AND. p1.le.(Nel-R)) then ! middle-
                          right side
581                                 wd_x = p2-1
582                                 avg_factor = ((R*2)+1)*((wd_x+R)+1)
583                                     do k1 = -R,R
584                                         do k2 = -wd_x,R
585                                         dist = sqrt((k1**2) +(k2**2))
586                                         infl = exp(-dist/dec)
587                                            dp_summed_matrix(p2,p1) =
588       1                                    dp_summed_matrix(p2,p1) +
589       2                                    dec*dp_save_matrix(p2+k2,p1+k1)
590                                         enddo
591                                     enddo
592                                     dp_avg_matrix(p2,p1) =
593       1                             dp_summed_matrix(p2,p1)/
594       2                             avg_factor
595                                 else if(p1.gt.(Nel-R)) then ! bottom-right corner
596                                     wd_x = p2-1
597                                     wd_y = Nel-p1
598                                     avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
599                                     do k1 = -R,wd_y
600                                         do k2 = -wd_x,R
601                                         dist = sqrt((k1**2) +(k2**2))
602                                         infl = exp(-dist/dec)
603                                            dp_summed_matrix(p2,p1) =
604       1                                    dp_summed_matrix(p2,p1) +
605       2                                    dec*dp_save_matrix(p2+k2,p1+k1)
606                                         enddo
607                                     enddo
608                                     dp_avg_matrix(p2,p1) =
609       1                             dp_summed_matrix(p2,p1)/
610       2                             avg_factor
611
612                                 endif
613
614                                 else if(p2.ge.(R+1) .and. p2.le.(Nel-R) .and.
615       1                          p1.gt.(Nel-R)) then ! Here we define the middle bottom
616                                     wd_y = Nel-p1
617                                     avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
618                                     do k1 = -R,wd_y
619                                         do k2 = -R,R
620                                         dist = sqrt((k1**2) +(k2**2))
621                                         infl = exp(-dist/dec)
622                                            dp_summed_matrix(p2,p1) =
623       1                                    dp_summed_matrix(p2,p1) +
624       2                                    dec*dp_save_matrix(p2+k2,p1+k1)
625                                         enddo
626                                     enddo
627                                     dp_avg_matrix(p2,p1) =
628       1                             dp_summed_matrix(p2,p1)/
629       2                             avg_factor
630                                 else if(p2.ge.(R+1) .and. p2.le.(Nel-R) .and.
631       1                          p1.lt.(R+1)) then ! Here we define the middle top
632                                     wd_y = p1-1
633                                     avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
```

```
                                    do k1 = -wd_y,R
                                        do k2 = -R,R
                                        dist = sqrt((k1**2) +(k2**2))
                                        infl = exp(-dist/dec)
                                           dp_summed_matrix(p2,p1) =
     1                                     dp_summed_matrix(p2,p1) +
     2                                     dec*dp_save_matrix(p2+k2,p1+k1)
                                        enddo
                                    enddo
                                    dp_avg_matrix(p2,p1) =
     1                              dp_summed_matrix(p2,p1)/
     2                              avg_factor
                        endif









                     enddo
                  enddo

C Going back from matrix to array (for dp averaged)
                    do k1 =1,Nel
                        do k2=1,Nel
                            index_rev = k2+((k1-1)*Nel)
                            dp_summed_array(index_rev) =
     1                      dp_summed_matrix(k2,k1)
                        enddo
                    enddo

                    dp_avg = dp_avg_array(noel)
                    statev(95) = dp_avg


                statev(1)=statev(1)+dp_avg









C Update the porosity with averaged value



                if (statev(1).ge.0.95) then
                    statev(1) = 0.95
                endif

                if (statev(1).le.0.0) then
                    statev(1) = 0.0
                endif
```

```
693
694                 IF (statev(1).lt.0.097267787) then
695                 modulus = 23440*(1-statev(1))**5.74
696
697                 ELSE
698                 modulus = 14927*(1-statev(1))**1.33
699
700                 ENDIF
701
702
703                 statev(100) = modulus
704
705                 if(jstep.gt.58) then
706
707                     por_save_array(noel) = statev(1)
708
709                     do k1 =1,Nel
710                         do k2=1,Nel
711                             index = k2+((k1-1)*Nel)
712                             por_save_matrix(k2,k1) = por_save_array(index)
713                         enddo
714                     enddo
715
716 C                  statev(80) = por_save_matrix(45,88)
717 C                  statev(81) = por_save_array(17445)
718
719 C                 Initialising with zeros
720
721
722
723                   do k1=1,Nel
724                       do k2=1,Nel
725                           por_summed_matrix(k2,k1) = 0
726                       enddo
727                   enddo
728
729                   do k1=1,Nel
730                       do k2=1,Nel
731                           por_avg_matrix(k2,k1) = 0
732                       enddo
733                   enddo
734
735
736 C                 Creating summed matrix
737                    do p1=1,Nel
738                       do p2 = 1,Nel
739                          IF (p1.gt.R .AND. p2.gt.R. AND.
740      1                       p1.le.(Nel-R). and. p2.le.(Nel-R)) then ! inner side of the
       cube
741
742                           avg_factor = ((R*2)+1)*((R*2)+1)
743                           do k1 =-R,R
744                               do k2=-R,R
745                                       dist = sqrt((k1**2) +(k2**2))
746                                       infl = exp(-dist/dec)
747                                       por_summed_matrix(p2,p1) =
748      1                                por_summed_matrix(p2,p1) +
749      2                                dec*por_save_matrix(p2+k2,p1+k1)
750                               enddo
```

```
751                         enddo
752                                 por_avg_matrix(p2,p1) =
753       1                         por_summed_matrix(p2,p1)/
754       2                         avg_factor
755
756
757                      else if(p2.gt.(Nel-R)) then ! Here we define the left side of
                        the cube
758                         if(p1.lt.R+1) then ! top-left corner
759                             wd_x = Nel-p2
760                             wd_y = p1-1
761                             avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
762                                 do k1 = -wd_y,R
763                                     do k2 = -R,wd_x
764                                     dist = sqrt((k1**2) +(k2**2))
765                                     infl = exp(-dist/dec)
766                                        por_summed_matrix(p2,p1) =
767       1                                por_summed_matrix(p2,p1) +
768       2                                dec*por_save_matrix(p2+k2,p1+k1)
769                                     enddo
770                                 enddo
771                                 por_avg_matrix(p2,p1) =
772       1                         por_summed_matrix(p2,p1)/
773       2                         avg_factor
774
775                         else if(p1.ge.(R+1). AND. p1.le.(Nel-R)) then  ! middle-
                        left side
776                             wd_x = Nel-p2
777                             avg_factor = ((R*2)+1)*((wd_x+R)+1)
778                                 do k1 = -R,R
779                                     do k2 = -R,wd_x
780                                     dist = sqrt((k1**2) +(k2**2))
781                                     infl = exp(-dist/dec)
782                                        por_summed_matrix(p2,p1) =
783       1                                por_summed_matrix(p2,p1) +
784       2                                dec*por_save_matrix(p2+k2,p1+k1)
785                                     enddo
786                                 enddo
787                                 por_avg_matrix(p2,p1) =
788       1                         por_summed_matrix(p2,p1)/
789       2                         avg_factor
790                         else if(p1.gt.(Nel-R)) then ! bottom-left corner
791                             wd_x = Nel-p2
792                             wd_y = Nel-p1
793                             avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
794                                 do k1 = -R,wd_y
795                                     do k2 = -R,wd_x
796                                     dist = sqrt((k1**2) +(k2**2))
797                                     infl = exp(-dist/dec)
798                                        por_summed_matrix(p2,p1) =
799       1                                por_summed_matrix(p2,p1) +
800       2                                dec*por_save_matrix(p2+k2,p1+k1)
801                                     enddo
802                                 enddo
803                                 por_avg_matrix(p2,p1) =
804       1                         por_summed_matrix(p2,p1)/
805       2                         avg_factor
806
807                         endif
```

```
                        else if(p2.lt.(R+1)) then ! Here we define the right side of the cube
                            if(p1.lt.(R+1)) then ! top-right corner
                                wd_x = p2-1
                                wd_y = p1-1
                                avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
                                    do k1 = -wd_y,R
                                        do k2 = -wd_x,R
                                        dist = sqrt((k1**2) +(k2**2))
                                        infl = exp(-dist/dec)
                                           por_summed_matrix(p2,p1) =
     1                                     por_summed_matrix(p2,p1) +
     2                                     dec*por_save_matrix(p2+k2,p1+k1)
                                        enddo
                                    enddo
                                    por_avg_matrix(p2,p1) =
     1                              por_summed_matrix(p2,p1)/
     2                              avg_factor

                            else if(p1.ge.(R+1). AND. p1.le.(Nel-R)) then ! middle-right side
                                wd_x = p2-1
                                avg_factor = ((R*2)+1)*((wd_x+R)+1)
                                    do k1 = -R,R
                                        do k2 = -wd_x,R
                                        dist = sqrt((k1**2) +(k2**2))
                                        infl = exp(-dist/dec)
                                           por_summed_matrix(p2,p1) =
     1                                     por_summed_matrix(p2,p1) +
     2                                     dec*por_save_matrix(p2+k2,p1+k1)
                                        enddo
                                    enddo
                                    por_avg_matrix(p2,p1) =
     1                              por_summed_matrix(p2,p1)/
     2                              avg_factor
                            else if(p1.gt.(Nel-R)) then ! bottom-right corner
                                wd_x = p2-1
                                wd_y = Nel-p1
                                avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
                                    do k1 = -R,wd_y
                                        do k2 = -wd_x,R
                                        dist = sqrt((k1**2) +(k2**2))
                                        infl = exp(-dist/dec)
                                           por_summed_matrix(p2,p1) =
     1                                     por_summed_matrix(p2,p1) +
     2                                     dec*por_save_matrix(p2+k2,p1+k1)
                                        enddo
                                    enddo
                                    por_avg_matrix(p2,p1) =
     1                              por_summed_matrix(p2,p1)/
     2                              avg_factor

                            endif

                            else if(p2.ge.(R+1) .and. p2.le.(Nel-R) .and.
     1                       p1.gt.(Nel-R)) then ! Here we define the middle bottom
                                wd_y = Nel-p1
                                avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
                                    do k1 = -R,wd_y
```

```
                                      do k2 = -R,R
                                      dist = sqrt((k1**2) +(k2**2))
                                      infl = exp(-dist/dec)
                                         por_summed_matrix(p2,p1) =
     1                                   por_summed_matrix(p2,p1) +
     2                                   dec*por_save_matrix(p2+k2,p1+k1)
                                      enddo
                                  enddo
                                  por_avg_matrix(p2,p1) =
     1                            por_summed_matrix(p2,p1)/
     2                            avg_factor
                          else if(p2.ge.(R+1) .and. p2.le.(Nel-R) .and.
     1                     p1.lt.(R+1)) then ! Here we define the middle top
                              wd_y = p1-1
                              avg_factor = ((wd_y+R)+1)*((wd_x+R)+1)
                                  do k1 = -wd_y,R
                                      do k2 = -R,R
                                      dist = sqrt((k1**2) +(k2**2))
                                      infl = exp(-dist/dec)
                                         por_summed_matrix(p2,p1) =
     1                                   por_summed_matrix(p2,p1) +
     2                                   dec*por_save_matrix(p2+k2,p1+k1)
                                      enddo
                                  enddo
                                  por_avg_matrix(p2,p1) =
     1                            por_summed_matrix(p2,p1)/
     2                            avg_factor
                      endif










                   enddo
               enddo

C Going back from matrix to array (for averaged)
                  do k1 =1,Nel
                      do k2=1,Nel
                          index_rev = k2+((k1-1)*Nel)
                          por_avg_array(index_rev) =
     1                    por_avg_matrix(k2,k1)
                      enddo
                  enddo


C Going back from matrix to array (for summed)
                  do k1 =1,Nel
                      do k2=1,Nel
                          index_rev = k2+((k1-1)*Nel)
                          por_summed_array(index_rev) =
     1                    por_summed_matrix(k2,k1)
                      enddo
                  enddo
```

```
924                   por_final_averaged = por_avg_array(noel)
925                   statev(98) = por_final_averaged
926
927                   por_final_summed = por_summed_array(noel)
928                   statev(97) = por_final_summed
929
930
931
932              endif
933           endif
934
935           endif
936
937
938
939
940
941           RETURN
942           END
943
944
945
```

Appendix D: Test data results