Shepard's Method

```
clear all
close all
clc

tic

stepno = 10; % this is the number of time-steps which are going to be used in
calculation.



% xl = -30; %x min
% xr = 30; %x max
% yb = -25; %y min
% yt = 25;%y max
% zb = -20;  % z min
% zf = 20;   % z max

xl =-33.5;  %These values determine the size of our grid.
xr = 36.0;
yb =-27.2;
yt = 30.4;
zb =-21.6;
zf = 22.6;

nx = 25;  % nx gives the number of elements from end to end in the x
direction.
ny = nx;
nz = nx; % The grid will have nx*ny*nz elements

delx = (xr-xl)/nx; dely = (yt-yb)/ny; delz = (zf-zb)/nz; %get element height,
width and depth



%%%%%%%%%%%%%%%%%%%%%%%%%%

load('C:\Users\Rodrigo\Documents\MATLAB\MATH thesis\C_coord\Ccoords1');
x0 = X;
y0 = Y;
z0 = Z;

% we get the positions at time zero (t0).
x2 = x0; y2 = y0; z2 = z0; %set positions at time zero to be current position
for first time.




%initialize quantitites
FsU = zeros(3,3,nx*ny*nz);
FsUp = zeros(3,3,length(x2));
Fp = zeros(3,3,length(x2));
```

```
%%%stepno = 2; % this is the number of time-steps which are going to be used
in calculation.

% %initialize quantitites
% quotma = zeros(1,stepno);
% Gma = zeros(1,stepno);
% LSma = zeros(1,stepno);
% quotmi = zeros(1,stepno);
% Gmi = zeros(1,stepno);
% LSmi = zeros(1,stepno);

for K = 1:stepno

    x1=x2; y1=y2; z1=z2; % Set the position of the last time step as the
"old" position for the current time step.

    string1 = num2str(K); string2 = num2str(K+1);

    combo1 = strcat('C:\Users\Rodrigo\Documents\MATLAB\MATH
thesis\C_coord\Ccoords',string2,'.mat');

    load(combo1)
    x2 = X;
    y2 = Y;
    z2 = Z;


    xys = (nx+2)*(nx+2); %number of elements on a plane parallel to the xy
plane (on outer enclosing grid).

    [xgro,ygro,zgro] = biggrid(xl,xr,yb,yt,zb,zf,nx,ny,nz); %get the points
for the outer grid. (see function biggrid).

    u = x2 - x1; v = y2 - y1; w = z2 - z1; %velocities in 3 directions, given
as change in position over a uniform time step.
    umag = sqrt((u.^2)+(v.^2)+(w.^2));     %magnitude of the velocity.

    U = x2 - x0; V = y2 - y0; W = z2 - z0; %Displacement  = current position
- initial position
    Umag = sqrt((U.^2)+(V.^2)+(W.^2));


    % The function below interpolates the values of the data points to the
    % grid points for the "enclosing grid". % this will give the sum of the
contributions and of the weights to the
    % grid points of the "enclosing grid".

    [fu,fv,fw,fD,fU,fV,fW,fDU,N] =
my_lin_interp(x2,y2,z2,u,v,w,umag,U,V,W,Umag,nx,ny,nz,xys,delx,dely,delz,xgro
,ygro,zgro,xl,yb,zb);

    % This function gives the interpolated values at the grid points by
```

```
    % dividing the sum of the contributions from the data points by the sum
of
    % the weights of those contributions.
    [ugrid,vgrid,wgrid,Dgrid,Ugrid,Vgrid,Wgrid,DUgrid] =
grid_vals(N,x2,fu,fv,fw,fD,fU,fV,fW,fDU);


    %%%%%% Now we get deformation gradient (and other quenatities) at the
    %%%%%% location of the atoms.

    F = zeros(3,3,nx*ny*nz);  % initialize F, which will contain the
deformation gradient at the center of the elements.
    for i = 1:nx*ny*nz
        F(:,:,i) = eye(3);
    end

    % initialize quantities.
    normLSp     = zeros(1,length(x2));    %
    nabladelup  = zeros(1,length(x2)); %
    normGp      = zeros(1,length(x2));     %
    normNLp     = zeros(1,length(x2));
    normnoli    = zeros(1,length(x2));
    normnoli2   = zeros(1,length(x2));

    for p = 1:length(x2)
        xp = x2(p);
        yp = y2(p);
        zp = z2(p);
        i = 1 + fix((xp-xl)/delx);
        j = 1 + fix((yp-yb)/dely);
        k = 1 + fix((zp-zb)/delz);
        e = i + 1 + j*(nx+2)+k*xys;
        xi = (xp - xgro(e))/delx;
        eta  = (yp - ygro(e))/dely;
        zeta = (zp-zgro(e))/delz;


        kl(1) = e;
        kl(2) = e+1;
        kl(3) = e+nx+2;
        kl(4) = e+nx+3;
        kl(5) = e+xys;
        kl(6) = e+1+xys;
        kl(7) = e+nx+2+xys;
        kl(8) = e+nx+3+xys;

        udef = zeros(1,8); % initialize velocities at points
        vdef = zeros(1,8);
        wdef = zeros(1,8);

        Udef = zeros(1,8); % initialize displacements at points
        Vdef = zeros(1,8);
        Wdef = zeros(1,8);

        for i = 1:8
```

```
            udef(i) = ugrid(kl(i)); % velocities at element's corners
            vdef(i) = vgrid(kl(i));
            wdef(i) = wgrid(kl(i));

            Udef(i) = Ugrid(kl(i)); % displacements at element's corners
            Vdef(i) = Vgrid(kl(i));
            Wdef(i) = Wgrid(kl(i));
        end

        ds = derivshape(xi,eta,zeta,delx,dely,delz);

        dudxp = zeros(3);
        dUdxp = zeros(3);
        for j =1:3
            dudxp(1,j) = sum(ds(:,j).*udef(:));
            dudxp(2,j) = sum(ds(:,j).*vdef(:));
            dudxp(3,j) = sum(ds(:,j).*wdef(:));
            dUdxp(1,j) = sum(ds(:,j).*Udef(:));
            dUdxp(2,j) = sum(ds(:,j).*Vdef(:));
            dUdxp(3,j) = sum(ds(:,j).*Wdef(:));
        end

        Fmp = eye(3)+dudxp;
        Fp(:,:,p) = Fmp*Fp(:,:,p);

        LSm = (1/2)*(dUdxp+dUdxp');
        normLSp(p) = norm(LSm,'fro');

        nabladelup(p) = trace(dudxp);

        FmU = eye(3)+dUdxp;

        Gm = (1/2)*((FmU')*FmU-eye(3));
        normGp(p) = norm(Gm,'fro'); %Difference between almansi and linear
strains

        normnoli(p)  = norm(Gm-LSm);

        normnoli2(p) = norm(.5*(dUdxp')*dUdxp);

        FsUp(:,:,p) = FmU;
    end

    detFp = ones(1,length(x2));
    for i = 1:length(x2)
        detFp(i) = det(Fp(:,:,i));
    end

    detFsUp = ones(1,length(x2));
    for i = 1:length(x2)
        detFsUp(i) = det(FsUp(:,:,i));
    end
```

```
    %% we add deformation grad, etc to files qith quantities at the points.
    %         normnoli    = zeros(1,length(x2));
    %     normnoli2   = zeros(1,length(x2));
    A = [detFp',detFsUp',nabladelup',normGp',normLSp',normnoli',normnoli2'];

head=['x','y','z','u','v','w','D','U','V','W','M','f','F','d','G','L','N','Q'
];
    Mexact = [x2,y2,z2,u,v,w,umag,U,V,W,Umag,A];

    combo4 = strcat('C:\Users\Rodrigo\Documents\UNM\MATHS\MATH
thesis\csv_interpol_data\nx25\datavels\data_step',string1,'.csv');
    dlmwrite(combo4,head)
    dlmwrite(combo4,Mexact,'-append')



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



    %get values for the points in the inner grid only.
    [xgr,ygr,zgr,Ngr,ugr,vgr,wgr,Dgr,Ugr,Vgr,Wgr,DUgr] =
outer_to_innergrid(nx,ny,nz,xgro,ygro,zgro,N,ugrid,vgrid,wgrid,Dgrid,Ugrid,Vg
rid,Wgrid,DUgrid);



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



    xdfg = zeros(1,(nx^3)); %initialize vectors of location of F, Finv
    ydfg = zeros(1,(nx^3));
    zdfg = zeros(1,(nx^3));



    nabladelu = zeros(1,nx^3);
    normG = zeros(1,nx^3);
    normLS = zeros(1,nx^3);
    normNonlin = zeros(1,nx^3);
    normNonlin2 = zeros(1,nx^3);

    indice = zeros(1,nx^3);
    for j = 1:nx
        for i = 1:nx
            indice(1+(i-1)*nx+(j-1)*nx^2:1+(i-1)*nx+(j-1)*nx^2+nx-1) = (1+(i-
1)*(nx+1)+(j-1)*(nx+1)^2:1+(i-1)*(nx+1)+(j-1)*(nx+1)^2+nx-1);
        end
    end

    for m = 1:nx^3

        k = indice(m);
```

```
        kl(1) = k;          % set indices of points for element we are
working with.
        kl(2) = k+1;
        kl(3) = k+nx+1;
        kl(4) = kl(3)+1;
        kl(5) = k+(nx+1)^2;
        kl(6) = kl(5)+1;
        kl(7) = kl(5)+nx+1;
        kl(8) = kl(7)+1;

        xdfg(m) = xgr(k)+delx/2; % set coordinates of point for which we are
calculating deformation gradient.
        ydfg(m) = ygr(k)+delx/2;
        zdfg(m) = zgr(k)+delx/2;

        udef = zeros(1,8); % initialize velocities at points
        vdef = zeros(1,8);
        wdef = zeros(1,8);

        Udef = zeros(1,8); % initialize displacements at points
        Vdef = zeros(1,8);
        Wdef = zeros(1,8);

        for i = 1:8
            udef(i) = ugr(kl(i)); % velocities at element's corners
            vdef(i) = vgr(kl(i));
            wdef(i) = wgr(kl(i));

            Udef(i) = Ugr(kl(i)); % displacements at element's corners
            Vdef(i) = Vgr(kl(i));
            Wdef(i) = Wgr(kl(i));
        end


        dudx(1,1) = (1/4)*(((udef(2)-udef(1))/delx)+((udef(4)-
udef(3))/delx)+((udef(6)-udef(5))/delx)+((udef(8)-udef(7))/delx));
        dudx(2,1) = (1/4)*(((vdef(2)-vdef(1))/delx)+((vdef(4)-
vdef(3))/delx)+((vdef(6)-vdef(5))/delx)+((vdef(8)-vdef(7))/delx));
        dudx(3,1) = (1/4)*(((wdef(2)-wdef(1))/delx)+((wdef(4)-
wdef(3))/delx)+((wdef(6)-wdef(5))/delx)+((wdef(8)-wdef(7))/delx));

        dudx(1,2) = (1/4)*(((udef(3)-udef(1))/delx)+((udef(4)-
udef(2))/delx)+((udef(7)-udef(5))/delx)+((udef(8)-udef(6))/delx));
        dudx(2,2) = (1/4)*(((vdef(3)-vdef(1))/delx)+((vdef(4)-
vdef(2))/delx)+((vdef(7)-vdef(5))/delx)+((vdef(8)-vdef(6))/delx));
        dudx(3,2) = (1/4)*(((wdef(3)-wdef(1))/delx)+((wdef(4)-
wdef(2))/delx)+((wdef(7)-wdef(5))/delx)+((wdef(8)-wdef(6))/delx));

        dudx(1,3) = (1/4)*(((udef(5)-udef(1))/delx)+((udef(6)-
udef(2))/delx)+((udef(7)-udef(3))/delx)+((udef(8)-udef(4))/delx));
        dudx(2,3) = (1/4)*(((vdef(5)-vdef(1))/delx)+((vdef(6)-
vdef(2))/delx)+((vdef(7)-vdef(3))/delx)+((vdef(8)-vdef(4))/delx));
        dudx(3,3) = (1/4)*(((wdef(5)-wdef(1))/delx)+((wdef(6)-
wdef(2))/delx)+((wdef(7)-wdef(3))/delx)+((wdef(8)-wdef(4))/delx));
```

```
        dUdx(1,1) = (1/4)*(((Udef(2)-Udef(1))/delx)+((Udef(4)-
Udef(3))/delx)+((Udef(6)-Udef(5))/delx)+((Udef(8)-Udef(7))/delx));
        dUdx(2,1) = (1/4)*(((Vdef(2)-Vdef(1))/delx)+((Vdef(4)-
Vdef(3))/delx)+((Vdef(6)-Vdef(5))/delx)+((Vdef(8)-Vdef(7))/delx));
        dUdx(3,1) = (1/4)*(((Wdef(2)-Wdef(1))/delx)+((Wdef(4)-
Wdef(3))/delx)+((Wdef(6)-Wdef(5))/delx)+((Wdef(8)-Wdef(7))/delx));

        dUdx(1,2) = (1/4)*(((Udef(3)-Udef(1))/delx)+((Udef(4)-
Udef(2))/delx)+((Udef(7)-Udef(5))/delx)+((Udef(8)-Udef(6))/delx));
        dUdx(2,2) = (1/4)*(((Vdef(3)-Vdef(1))/delx)+((Vdef(4)-
Vdef(2))/delx)+((Vdef(7)-Vdef(5))/delx)+((Vdef(8)-Vdef(6))/delx));
        dUdx(3,2) = (1/4)*(((Wdef(3)-Wdef(1))/delx)+((Wdef(4)-
Wdef(2))/delx)+((Wdef(7)-Wdef(5))/delx)+((Wdef(8)-Wdef(6))/delx));

        dUdx(1,3) = (1/4)*(((Udef(5)-Udef(1))/delx)+((Udef(6)-
Udef(2))/delx)+((Udef(7)-Udef(3))/delx)+((Udef(8)-Udef(4))/delx));
        dUdx(2,3) = (1/4)*(((Vdef(5)-Vdef(1))/delx)+((Vdef(6)-
Vdef(2))/delx)+((Vdef(7)-Vdef(3))/delx)+((Vdef(8)-Vdef(4))/delx));
        dUdx(3,3) = (1/4)*(((Wdef(5)-Wdef(1))/delx)+((Wdef(6)-
Wdef(2))/delx)+((Wdef(7)-Wdef(3))/delx)+((Wdef(8)-Wdef(4))/delx));

        Fm = eye(3)+dudx;

        F(:,:,m) = Fm*F(:,:,m);
        nabladelu(1,m) = trace(dudx);


        FmU = eye(3)+dUdx;

        FsU(:,:,m) = FmU;

        LSm = (1/2)*(dUdx+dUdx');
        normLS(m) = norm(LSm,'fro');

        Gm = (1/2)*(FmU'*FmU-eye(3));
        normG(m) = norm(Gm,'fro');

        normNonlin(m) = norm(Gm-LSm);
        normNonlin2(m) = norm(.5*(dUdx')*dUdx);

    end

    detF = ones(1,nx^3);
    for i = 1:nx^3
        detF(i) = det(F(:,:,i));
    end

    detFsU = ones(1,nx^3);
    for i = 1:nx^3
        detFsU(i) = det(FsU(:,:,i));
    end

    indexF = find(detF~=1);
```

```
    detF = detF(indexF);
    detFsU = detFsU(indexF);
    nabladelu = nabladelu(indexF);
    normG = normG(indexF);
    normLS= normLS(indexF);
    normNonlin=normNonlin(indexF);
    normNonlin2=normNonlin2(indexF);
    xdfg = xdfg(indexF);
    ydfg = ydfg(indexF);
    zdfg = zdfg(indexF);


    %%%%%%% We are getting some funky norms (O(10^79))

    %     quotma(K) = max((normG-normLS)./normG);
    %     Gma(K) = max(normG);
    %     LSma(K) = max(normLS);
    %     quotmi(K) = min((normG-normLS)./normG);
    %     Gmi(K) = min(normG);
    %     LSmi(K) = min(normLS);

    %%%%%%%%%

    A =
[xdfg',ydfg',zdfg',detF',detFsU',nabladelu',normG',normLS',normNonlin',normNo
nlin2'];

    headF=['x','y','z','f','F','d','G','L','N','Q'];


    comboF = strcat('C:\Users\Rodrigo\Documents\UNM\MATHS\MATH
thesis\csv_interpol_data\nx25\interpstrains\interp_strains_',string1,'.csv');
    dlmwrite(comboF,headF)
    dlmwrite(comboF,A,'-append')

    %%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%

    index3 = find(Ngr~=0);
    xgr = xgr(index3);
    ygr = ygr(index3);
    zgr = zgr(index3);

    ugr = ugr(index3);
    vgr = vgr(index3);
    wgr = wgr(index3);
    Dgr = Dgr(index3);

    Ugr = Ugr(index3);
    Vgr = Vgr(index3);
    Wgr = Wgr(index3);
    DUgr = DUgr(index3);
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    head=['x','y','z','u','v','w','D','U','V','W','M'];

    Mapprox = [xgr',ygr',zgr',ugr',vgr',wgr',Dgr',Ugr',Vgr',Wgr',DUgr'];

    combo3 = strcat('C:\Users\Rodrigo\Documents\UNM\MATHS\MATH
thesis\csv_interpol_data\nx25\interpvels\interp_step',string1,'.csv');
    dlmwrite(combo3,head)
    dlmwrite(combo3,Mapprox,'-append')

    %%%%


end



toc


function fu = elem_lin_interp(e,nx,xys,xi,eta,zeta,fu,g)
% fu = elem_lin_interp(e,nx,xys,xi,eta,zeta,fu)
%  This calculates the contribution of atom p to the gridpoints at the
corners of element e for a
%  quantity fu.

    fu(e) = fu(e) + g*(1-zeta)*(1-xi)*(1-eta);
    fu(e+1) = fu(e+1) + g*(1-zeta)*xi*(1-eta);
    fu(e+nx+2) = fu(e+nx+2) + g*(1-zeta)*(1-xi)*eta;
    fu(e+nx+3) = fu(e+nx+3) + g*(1-zeta)*xi*eta;
    fu(e+xys) = fu(e+xys) + g*zeta*(1-xi)*(1-eta);
    fu(e+1+xys) = fu(e+1+xys) + g*zeta*xi*(1-eta);
    fu(e+nx+2+xys) = fu(e+nx+2+xys) + g*zeta*(1-xi)*eta;
    fu(e+nx+3+xys) = fu(e+nx+3+xys) + g*zeta*xi*eta;

end


function [ugrid,vgrid,wgrid,Dgrid,Ugrid,Vgrid,Wgrid,DUgrid] =
grid_vals(N,x2,fu,fv,fw,fD,fU,fV,fW,fDU)
% [ugrid,vgrid,wgrid,Dgrid,Ugrid,Vgrid,Wgrid,DUgrid] =
clean_nans(N,x2,fu,fv,fw,fD,fU,fV,fW,fDU)
% This function gives the interpolated values at the grid points by
% dividing the sum of the contributions from the data points by the sum of
% the weights of those contributions. It also takes care to avoid NaNs by
% setting the value at the gridpoint to zero when N=0 (if N=0 then there is
% no contribution from any data point to that gridpoint).
```

```
index1 = find(N==0); %Whent N=0, we get division by zero so NaN. Find where
N=0 if no weight then no contribution.
index2 = find(N~=0); % We get N=0 for elements with no data points in them.

lx2 = length(x2);

ugrid = zeros(1,lx2);
ugrid(index1) = 0;
ugrid(index2) = fu(index2)./N(index2); %this is the sum of the contributions
over the sum of the weigths.

vgrid = zeros(1,lx2);
vgrid(index1) = 0;
vgrid(index2) = fv(index2)./N(index2);

wgrid = zeros(1,lx2);
wgrid(index1) = 0;
wgrid(index2) = fw(index2)./N(index2);

Dgrid = zeros(1,lx2);
Dgrid(index1) = 0;
Dgrid(index2) = fD(index2)./N(index2);

Ugrid = zeros(1,lx2);
Ugrid(index1) = 0;
Ugrid(index2) = fU(index2)./N(index2);

Vgrid = zeros(1,lx2);
Vgrid(index1) = 0;
Vgrid(index2) = fV(index2)./N(index2);

Wgrid = zeros(1,lx2);
Wgrid(index1) = 0;
Wgrid(index2) = fW(index2)./N(index2);

DUgrid = zeros(1,lx2);
DUgrid(index1) = 0;
DUgrid(index2) = fDU(index2)./N(index2);

end
```

```
function [fu,fv,fw,fD,fU,fV,fW,fDU,N] =
my_lin_interp(x2,y2,z2,u,v,w,umag,U,V,W,Umag,nx,ny,nz,xys,delx,dely,delz,xgro
,ygro,zgro,xl,yb,zb)
% [fu,fv,fw,fD,fU,fV,fW,fDU,N] =
my_lin_interp(x2,y2,z2,u,v,w,umag,U,V,W,Umag,nx,ny,nz,xys,delx,dely,delz,xgro
,ygro,zgro,xl,yb,zb)
% this will give the sum of the contributions and of the weights to the
% grid points of the "enclosing grid".

[fu,fv,fw,fD,fU,fV,fW,fDU,N] = inicia(nx,ny,nz);
```

```
for p = 1:length(x2) %here we have length(x2) = length(y2) = length(z2) =
number of atoms. So we interpolate all atoms.

    xp = x2(p); yp = y2(p); zp = z2(p);
    i = 1 + fix((xp-xl)/delx);  % this indices are to find the elemnt number
were atom p is located.
    j = 1 + fix((yp-yb)/dely);
    k = 1 + fix((zp-zb)/delz);
    e = i + 1 + j*(nx+2)+k*xys;  %this defines the element number
    xi = (xp - xgro(e))/delx;
    eta  = (yp - ygro(e))/dely;
    zeta = (zp-zgro(e))/delz;


    fu = elem_lin_interp(e,nx,xys,xi,eta,zeta,fu,u(p)); %Compute
contributions to velocity.
    fv = elem_lin_interp(e,nx,xys,xi,eta,zeta,fv,v(p));
    fw = elem_lin_interp(e,nx,xys,xi,eta,zeta,fw,w(p));
    fD = elem_lin_interp(e,nx,xys,xi,eta,zeta,fD,umag(p));

    fU = elem_lin_interp(e,nx,xys,xi,eta,zeta,fu,U(p)); %Compute
contributions to Displacement
    fV = elem_lin_interp(e,nx,xys,xi,eta,zeta,fv,V(p));
    fW = elem_lin_interp(e,nx,xys,xi,eta,zeta,fw,W(p));
    fDU = elem_lin_interp(e,nx,xys,xi,eta,zeta,fD,Umag(p));

    N = elem_lin_interp(e,nx,xys,xi,eta,zeta,N,1); %get weights for
contributions.

end


end
```

Moving Least squares

```
clear all
close all
clc

tic

stepno = 50; % this is the number of time-steps which are going to be used in
calculation.

% xl =-33.5;  %These values determine the size of our grid.
% xr = 36.0;
% yb =-27.2;
% yt = 30.4;
% zb =-21.6;
% zf = 22.6;

xl =-29;  %These values determine the size of our grid.
xr = 29;
yb =-23;
yt = 24;
zb =-17;
zf = 17;

nx = 4;  % nx gives the number of elements from end to end in the x
direction.
ny = nx;
nz = nx; % The grid will have nx*ny*nz elements



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
delx = (xr-xl)/nx; dely = (yt-yb)/ny; delz = (zf-zb)/nz; %get element height,
width and depth

xgr = xl:delx:xr;
ygr = yb:dely:yt;
zgr = zb:delz:zf;

[Xgr,Ygr,Zgr] = meshgrid(xgr,ygr,zgr);  %Create the grid.


xgr = reshape(Xgr,(nx+1)*(ny+1)*(nz+1),1);
ygr = reshape(Ygr,(nx+1)*(ny+1)*(nz+1),1);
zgr = reshape(Zgr,(nx+1)*(ny+1)*(nz+1),1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load('C:\Users\Rodrigo\Documents\MATLAB\MATH
thesis\Comparisons\HIV_DATA\C_coord\Ccoords1')
x0 = X; %extract data from mat files.
y0 = Y;
z0 = Z;
```

```
x2 = x0; y2 = y0; z2 = z0; %set positions at time zero to be current position
for time 0

%D = norm([delx,dely,delz]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lngxgr = length(xgr); %so we save aourselves computing this in every
iteration.


for K = 1:stepno
    x1=x2; y1=y2; z1=z2; % Set the position of the last time step as the
"old" position for the current time step.
    string1 = num2str(K); string2 = num2str(K+1);

    combo1 = strcat('C:\Users\Rodrigo\Documents\MATLAB\MATH
thesis\Comparisons\HIV_DATA\C_coord\Ccoords',string2);
    load(combo1)

    x2 = X;
    y2 = Y;
    z2 = Z;


    u = x2 - x1; v = y2 - y1; w = z2 - z1; %velocities in 3 directions, given
as change in position over a uniform time step.

    U = x2 - x0; V = y2 - y0; W = z2 - z0; %Displacement  = current position
- initial position

    [Au,Av,Aw,AU,AV,AW,bu,bv,bw,bU,bV,bW] =
my_MLS_interp_lin(nx,ny,nz,x2,y2,z2,u,v,w,U,V,W,xgr,ygr,zgr,xl,yb,zb,delx,del
y,delz);

    NG = (nx+1)*(ny+1)*(nz+1);

    [ugr,vgr,wgr,Ugr,Vgr,Wgr] =
grid_valsMLS_lin(NG,xgr,ygr,zgr,Au,Av,Aw,AU,AV,AW,bu,bv,bw,bU,bV,bW);

    %Dgr = sqrt(ugr.^2+vgr.^2+wgr.^2); %Magnitude of velocity vector
    %Mgr = sqrt(Ugr.^2+Vgr.^2+Wgr.^2); %Magnitude of displacement vector

%     head=['x','y','z','u','v','w','U','V','W'];
%     Mapprox = [xgr,ygr,zgr,ugr,vgr,wgr,Ugr,Vgr,Wgr];
%     combo3 = strcat('C:\Users\Rodrigo\Documents\UNM\MATHS\MATH
thesis\MLS_interpolation_thinplate\nx_5\vels_disps\interp_step',string1,'.csv
');
%     dlmwrite(combo3,head)
%     dlmwrite(combo3,Mapprox,'-append')


    %%%%% Now we get the Displacement gradient tensor to calculate the
    %%%%% deformation gradient tensor, strain and linear strain.
```

```
    [gradU] = myfinitediffGRAD(xgr,ygr,zgr,delx,dely,delz,Ugr); %the gradient
of each element of the displacement is a vector
    [gradV] = myfinitediffGRAD(xgr,ygr,zgr,delx,dely,delz,Vgr); % we use
these to build nabla(Displacement), a tensor
    [gradW] = myfinitediffGRAD(xgr,ygr,zgr,delx,dely,delz,Wgr);



    % For the dispGrad the ith row is the grad of the ith component of
    % displacement.

    Dispgrad = zeros(3,3,lngxgr);   %initialize stuff
    F = zeros(3,3,lngxgr);
    G = zeros(3,3,lngxgr);
    LS = zeros(3,3,lngxgr);

    Jac = zeros(lngxgr,1);
    G2norm = zeros(lngxgr,1);
    Gfronorm = zeros(lngxgr,1);
    LS2norm = zeros(lngxgr,1);
    LSfronorm = zeros(lngxgr,1);

    for i = 1:lngxgr;
        Dispgrad(1,:,i) = gradU(i,:);       %Now we use this to build the
deformation gradient tensor F, the Strain
        Dispgrad(2,:,i) = gradV(i,:);       %tensor G and the linear strain
Tensor LS.
        Dispgrad(3,:,i) = gradW(i,:);

        Dispgrad_i = Dispgrad(:,:,i);

        F(:,:,i) = Dispgrad(:,:,i)+eye(3);

        Fi = F(:,:,i);

        G(:,:,i) = 0.5*((Fi')*Fi-eye(3));

        LS(:,:,i) = .5*(Dispgrad_i+Dispgrad_i');

        % now just add norms and determinants and stuff...

        Jac(i) = det(Fi); % This is the Jacobian! cool.

        G2norm(i) = norm(G(:,:,i),2);

        Gfronorm(i) = norm(G(:,:,i),'fro');

        LS2norm(i) = norm(LS(:,:,i),2);

        LSfronorm(i) = norm(LS(:,:,i),'fro');
```

```
    end



    %%%% Get stuff in format to write to VTK



    u = reshape(ugr,nx+1,ny+1,nz+1);
    v = reshape(vgr,nx+1,ny+1,nz+1);
    w = reshape(wgr,nx+1,ny+1,nz+1);

    U = reshape(Ugr,nx+1,ny+1,nz+1);
    V = reshape(Vgr,nx+1,ny+1,nz+1);
    W = reshape(Wgr,nx+1,ny+1,nz+1);

    gradU1 = reshape(gradU(:,1),nx+1,ny+1,nz+1);
    gradU2 = reshape(gradU(:,2),nx+1,ny+1,nz+1);
    gradU3 = reshape(gradU(:,3),nx+1,ny+1,nz+1);
    gradV1 = reshape(gradV(:,1),nx+1,ny+1,nz+1);
    gradV2 = reshape(gradV(:,2),nx+1,ny+1,nz+1);
    gradV3 = reshape(gradV(:,3),nx+1,ny+1,nz+1);
    gradW1 = reshape(gradW(:,1),nx+1,ny+1,nz+1);
    gradW2 = reshape(gradW(:,2),nx+1,ny+1,nz+1);
    gradW3 = reshape(gradW(:,3),nx+1,ny+1,nz+1);

    Jac = reshape(Jac,nx+1,ny+1,nz+1);
    G2norm = reshape(G2norm,nx+1,ny+1,nz+1);
    Gfronorm = reshape(Gfronorm,nx+1,ny+1,nz+1);
    LS2norm = reshape(LS2norm,nx+1,ny+1,nz+1);
    LSfronorm = reshape(LSfronorm,nx+1,ny+1,nz+1);



    liston = strcat('C:\Users\Rodrigo\Documents\UNM\MATHS\MATH
thesis\MLS_interpolation_cubsplines_linbase\nx_4\vels_disps\VTK_steps\data_st
ep',string1,'.vtk');

    writer = VtkWriter(liston,'interpolacion de atomos en
VTK',VtkDataset.Structured_Grid);
    writer.grid.setGridSize(size(Xgr));
    writer.grid.setGrid(Xgr,Ygr,Zgr);
    writer.addPointData();
    writer.addVectorToDataset('velocity',u,v,w);
    writer.addVectorToDataset('displacement',U,V,W);

writer.addVectorToDataset('U_Displacement_gradient',gradU1,gradU2,gradU3);

writer.addVectorToDataset('V_Displacement_gradient',gradV1,gradV2,gradV3);

writer.addVectorToDataset('W_Displacement_gradient',gradW1,gradW2,gradW3);
%%%%%%%%%%%%%%%%%%%%%%%%
    writer.addScalarToDataset('Jacobian','default','components',1,Jac)
    writer.addScalarToDataset('G2norm','default','components',1,G2norm)
    writer.addScalarToDataset('Gfronorm','default','components',1,Gfronorm)
```

```
    writer.addScalarToDataset('LS2norm','default','components',1,LS2norm)
    writer.addScalarToDataset('LSfronorm','default','components',1,LSfronorm)
    writer.write()

end


toc


function [Afu,Afv,Afw,AfU,AfV,AfW,bfu,bfv,bfw,bfU,bfV,bfW] =
iniciaMLS(nx,ny,nz)
% [Afu,Afv,Afw,AfU,AfV,AfW,bfu,bfv,bfw,bfU,bfV,bfW,] = iniciaMLS(nx,ny,nz);
% This function initializes the functions for the linear interpolation at
% the corners of a grid that encloses our grid of interest.

nb = 4; % nb = #of basis elements - 1-constant, 4-linear, 10-quadratic, 20-
cubic

Afu =zeros(nb,nb,(nx+1)*(ny+1)*(nz+1));  % interpolating function for
velocity and magnitude of velocity.
Afv =zeros(nb,nb,(nx+1)*(ny+1)*(nz+1));
Afw =zeros(nb,nb,(nx+1)*(ny+1)*(nz+1));
AfU =zeros(nb,nb,(nx+1)*(ny+1)*(nz+1));  % interpolating function for
displacement and magnitude of displacement.
AfV =zeros(nb,nb,(nx+1)*(ny+1)*(nz+1));
AfW =zeros(nb,nb,(nx+1)*(ny+1)*(nz+1));

bfu =zeros(nb,(nx+1)*(ny+1)*(nz+1));  % interpolating function for velocity
and magnitude of velocity.
bfv =zeros(nb,(nx+1)*(ny+1)*(nz+1));
bfw =zeros(nb,(nx+1)*(ny+1)*(nz+1));
bfU =zeros(nb,(nx+1)*(ny+1)*(nz+1));  % interpolating function for
displacement and magnitude of displacement.
bfV =zeros(nb,(nx+1)*(ny+1)*(nz+1));
bfW =zeros(nb,(nx+2)*(ny+2)*(nz+2));
end


function [Au,Av,Aw,AU,AV,AW,bu,bv,bw,bU,bV,bW] =
my_MLS_interp_lin(nx,ny,nz,x2,y2,z2,u,v,w,U,V,W,xgr,ygr,zgr,xl,yb,zb,delx,del
y,delz)
% [fu,fv,fw,fD,fU,fV,fW,fDU,N] =
my_lin_interp(x2,y2,z2,u,v,w,umag,U,V,W,Umag,nx,ny,nz,xys,delx,dely,delz,xgro
,ygro,zgro,xl,yb,zb)
% this will give the sum of the contributions and of the weights to the
% grid points of the "enclosing grid".

[Au,Av,Aw,AU,AV,AW,bu,bv,bw,bU,bV,bW] = iniciaMLS(nx,ny,nz);

for p = 1:length(x2) %here we have length(x2) = length(y2) = length(z2) =
number of atoms. So we interpolate all atoms.

    xp = x2(p); yp = y2(p); zp = z2(p);
```

```
    i = fix((xp-xl)/delx);  % this indices are to find the elemnt number were
atom p is located.
    j = fix((yp-yb)/dely);
    k = fix((zp-zb)/delz);

    indic = find((xgr>=(xl+(i-1)*delx)&xgr<=(xl+(i+2)*delx))&(ygr>=(yb+(j-
1)*dely)&ygr<=(yb+(j+2)*dely))&(zgr>=(zb+(k-1)*delz)&zgr<=(zb+(k+2)*delz)));
    %This creates an index with the grid points that get a contribution
    %from the current data point.

    up = u(p); % This is the data at the current data point.
    vp = v(p);
    wp = w(p);
    Up = U(p);
    Vp = V(p);
    Wp = W(p);

    for i = 1:length(indic)
        j = indic(i);
        xj = xgr(j);
        yj = ygr(j);
        zj = zgr(j);
        [Aj,bj] = MLS_contrib_lin(xp,yp,zp,xj,yj,zj,delx,dely,delz,up);
        Au(:,:,j)           = Au(:,:,j)             +  Aj;
        bu(:,j)           = bu(:,j)             +  bj;

        [Aj,bj] = MLS_contrib_lin(xp,yp,zp,xj,yj,zj,delx,dely,delz,vp);
        Av(:,:,j)           = Av(:,:,j)             +  Aj;
        bv(:,j)           = bv(:,j)             +  bj;

        [Aj,bj] = MLS_contrib_lin(xp,yp,zp,xj,yj,zj,delx,dely,delz,wp);
        Aw(:,:,j)           = Aw(:,:,j)             +  Aj;
        bw(:,j)           = bw(:,j)             +  bj;

        [Aj,bj] = MLS_contrib_lin(xp,yp,zp,xj,yj,zj,delx,dely,delz,Up);
        AU(:,:,j)           = AU(:,:,j)             +  Aj;
        bU(:,j)           = bU(:,j)             +  bj;

        [Aj,bj] = MLS_contrib_lin(xp,yp,zp,xj,yj,zj,delx,dely,delz,Vp);
        AV(:,:,j)           = AV(:,:,j)             +  Aj;
        bV(:,j)           = bV(:,j)             +  bj;

        [Aj,bj] = MLS_contrib_lin(xp,yp,zp,xj,yj,zj,delx,dely,delz,Wp);
        AW(:,:,j)           = AW(:,:,j)             +  Aj;
        bW(:,j)           = bW(:,j)             +  bj;

    end



end
```

```
end


function [A,b] = MLS_contrib_lin(xp,yp,zp,xgr,ygr,zgr,delx,dely,delz,u)
% [A,b] = MLS_contrib(xp,yp,zp,xgr,ygr,zgr,D,u)
%   Detailed explanation goes here

Xp = [1,xp',yp',zp'];

r1 = abs(xp-xgr)/(2*delx);
r2 = abs(yp-ygr)/(2*dely);
r3 = abs(zp-zgr)/(2*delz);

if r1 <= .5
    w1 = (2/3)-4*(r1^2)+4*(r1^3);
elseif r1 > .5 &&  r1 <= 1
    w1 = (4/3)-4*r1+4*(r1^2)-(4/3)*(r1^3);
else
    w1 = 0;
end

if r2 <= .5
    w2 = (2/3)-4*(r2^2)+4*(r2^3);
elseif r2 > .5 &&  r2 <= 1
    w2 = (4/3)-4*r2+4*(r2^2)-(4/3)*(r2^3);
else
    w2 = 0;
end

if r3 <= .5
    w3 = (2/3)-4*(r3^2)+4*(r3^3);
elseif r3 > .5 &&  r3 <= 1
    w3 = (4/3)-4*r3+4*(r3^2)-(4/3)*(r3^3);
else
    w3 = 0;
end

w = w1*w2*w3*(27/8); %%% This factor makes contributions near gridpoint to
have weight of 1.

A = w*(Xp')*Xp;
b = (w*Xp*u)';


end


function [ugrid,vgrid,wgrid,Ugrid,Vgrid,Wgrid] =
grid_valsMLS_lin(NG,xgr,ygr,zgr,Au,Av,Aw,AU,AV,AW,bu,bv,bw,bU,bV,bW)

[rows, cols] = size(Au(:,:,1));
au = zeros(cols,1);
```

```
av = zeros(cols,1);
aw = zeros(cols,1);
aU = zeros(cols,1);
aV = zeros(cols,1);
aW = zeros(cols,1);

for j  = 1:NG
    if cond(Au(:,:,j))<1e10
        au(:,j) = pinv(Au(:,:,j))*bu(:,j);
    else
        au(:,j) =NaN;
    end

    if cond(Av(:,:,j))<1e10
        av(:,j) = pinv(Av(:,:,j))*bv(:,j);
    else
        av(:,j) =NaN;
    end

    if cond(Aw(:,:,j))<1e10
        aw(:,j) = pinv(Aw(:,:,j))*bw(:,j);
    else
        aw(:,j) =NaN;
    end


    if cond(AU(:,:,j))<1e10
        aU(:,j) = pinv(AU(:,:,j))*bU(:,j);
    else
        aU(:,j) =NaN;

    end

    if cond(AV(:,:,j))<1e10
        aV(:,j) = pinv(AV(:,:,j))*bV(:,j);
    else
        aV(:,j) =NaN;

    end

    if cond(AW(:,:,j))<1e10
        aW(:,j) = pinv(AW(:,:,j))*bW(:,j);
    else
        aW(:,j) =NaN;
    end

end


px = [ones(size(xgr)),xgr,ygr,zgr];



ugrid = zeros(length(xgr),1);
```

```
vgrid = zeros(length(xgr),1);
wgrid = zeros(length(xgr),1);
Ugrid = zeros(length(xgr),1);
Vgrid = zeros(length(xgr),1);
Wgrid = zeros(length(xgr),1);

for i = 1:length(xgr)
    ugrid(i) = au(:,i)'*(px(i,:)');
    vgrid(i) = av(:,i)'*(px(i,:)');
    wgrid(i) = aw(:,i)'*(px(i,:)');
    Ugrid(i) = aU(:,i)'*(px(i,:)');
    Vgrid(i) = aV(:,i)'*(px(i,:)');
    Wgrid(i) = aW(:,i)'*(px(i,:)');
end




end


function [nabu] = myfinitediffGRAD(X,Y,Z,delx,dely,delz,u)
% [nabu] = myfinitediffGRAD(X,Y,Z,u)
% Uses the finite difference method to find the gradient of a scalar field
% u on a uniformaly spaced rectangular grid.
% assumes X,Y,Z are vectors containing the coordinates of your gridpoints.
% delx,dely and delz are scalars which give the distances between adjacents
% elements in the x y and z direction.

N = length(X); %get the number of points.
nabu = zeros(N,3);
for i = 1:N

    xi = X(i);
    yi = Y(i);
    zi = Z(i);
    ui = u(i);


    %%%%% First x direction

    indic = find((Y==yi) & (Z==zi) &  ( ((X>(xi-(1.1)*delx) & X<(xi-
0.9*delx))) | ((X>(xi+0.9*delx)&X<(xi+1.1*delx)) ) )  );

    if length(indic)==2
        Xl = X(indic);
        ul = u(indic);
        [~,le] = min(Xl);
        [~,ri] = max(Xl);
        nabu(i,1) = ((ul(ri)-ul(le))/(2*delx));
    elseif length(indic)==1
        if X(indic)>xi
            nabu(i,1) = (u(indic)-ui)/delx;
        elseif X(indic)<xi
```

```
            nabu(i,1) = (ui-u(indic))/delx;
        else
            error('???')
        end
    else
        error('???')
    end


    %%%%% Then y direction

    indic = find((X==xi) & (Z==zi) &  ( ((Y>(yi-(1.1)*dely) & Y<(yi-
0.9*dely))) | ((Y>(yi+0.9*dely)&Y<(yi+1.1*dely)))  )  );


    if length(indic)==2
        Yl = Y(indic);
        ul = u(indic);
        [~,le] = min(Yl);
        [~,ri] = max(Yl);
        nabu(i,2) = ((ul(ri)-ul(le))/(2*dely));
    elseif length(indic)==1
        if Y(indic)>yi
            nabu(i,2) = (u(indic)-ui)/dely;
        elseif Y(indic)<yi
            nabu(i,2) = (ui-u(indic))/dely;
        else
            error('???')
        end
    else
        error('???')
    end

    %%%%% Then z direction

    indic = find((X==xi)&(Y==yi)&  ( ((Z>(zi-(1.1)*delz) & Z<(zi-0.9*delz)))
| ((Z>(zi+0.9*delz)&Z<(zi+1.1*delz)))  )  );

    if length(indic)==2
        Zl = Z(indic);
        ul = u(indic);
        [~,le] = min(Zl);
        [~,ri] = max(Zl);
        nabu(i,3) = ((ul(ri)-ul(le))/(2*delz));
    elseif length(indic)==1
        if Z(indic)>zi
            nabu(i,3) = (u(indic)-ui)/delz;
        elseif Z(indic)<zi
            nabu(i,3) = (ui-u(indic))/delz;
        else
            error('???')
        end
    else
        error('???')
    end
```

```
end
```