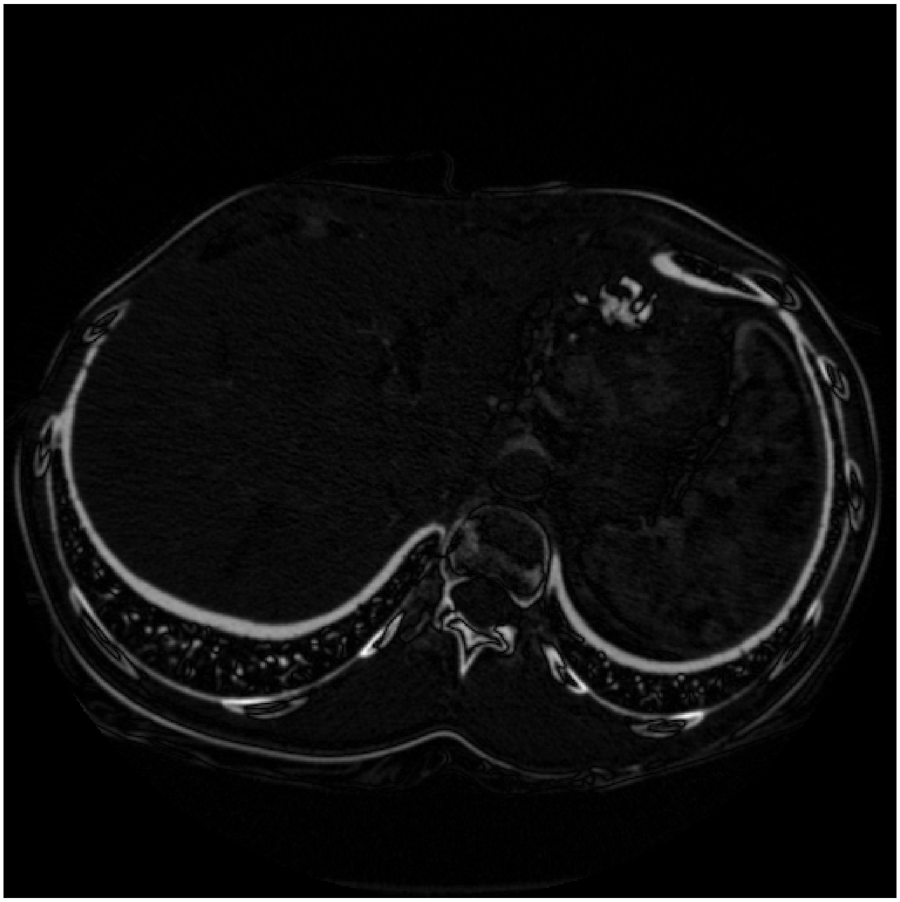
Table of Contents

Clear Up Workspace

```
clc; clear; close all;

% add all paths to current workspace recursively
currentFolder = pwd;
addpath(genpath(currentFolder));

% Read In Images
Template = im2double(dicomread("Data/Template.dcm"));
Source = im2double(dicomread("Data/Source.dcm"));
Diff = Template - Source;
figure; imshowpair(Template,Source,'diff');
```



Defining First Set Of Initial Conditions

```
x0=0;
y0=0;

dt = 10; % initialy define the maximal time step
Umax =  0.05; % define the deformation Limit - Need To come Back
% to this
tInitial = 0;  % define the initial time step
tFinal = 20; % define maximal iterations;
```

Defining Second Set Of Initial Conditions

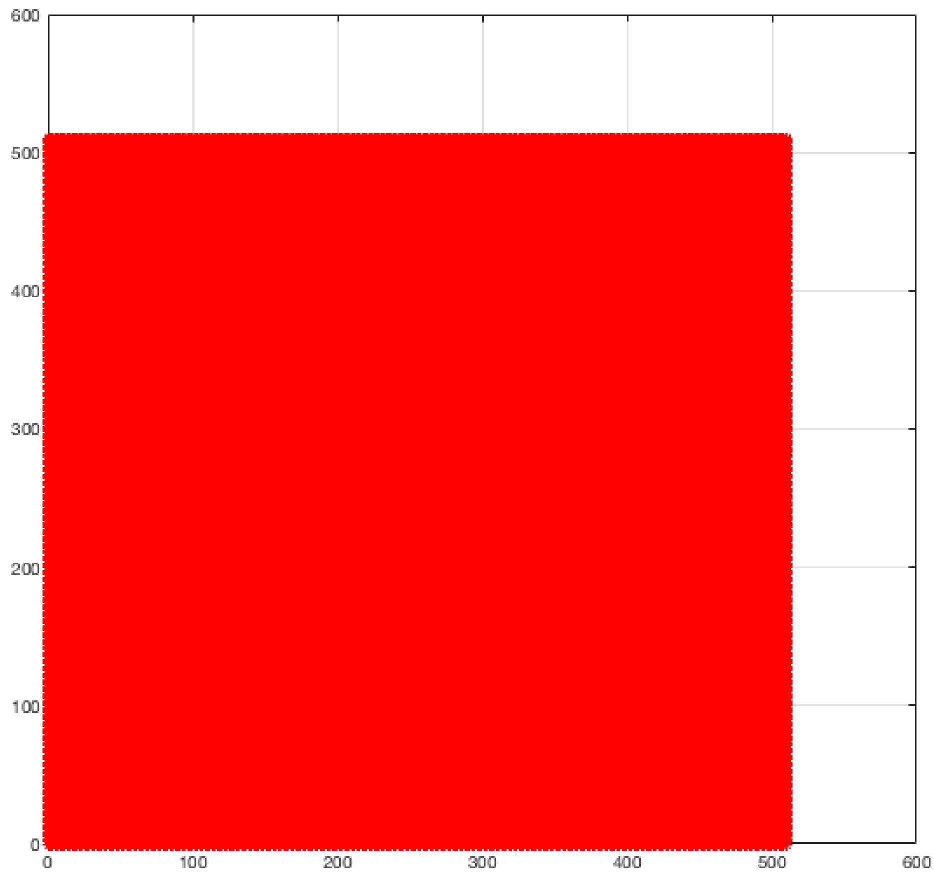
grid/mesh size shoud match that of the template

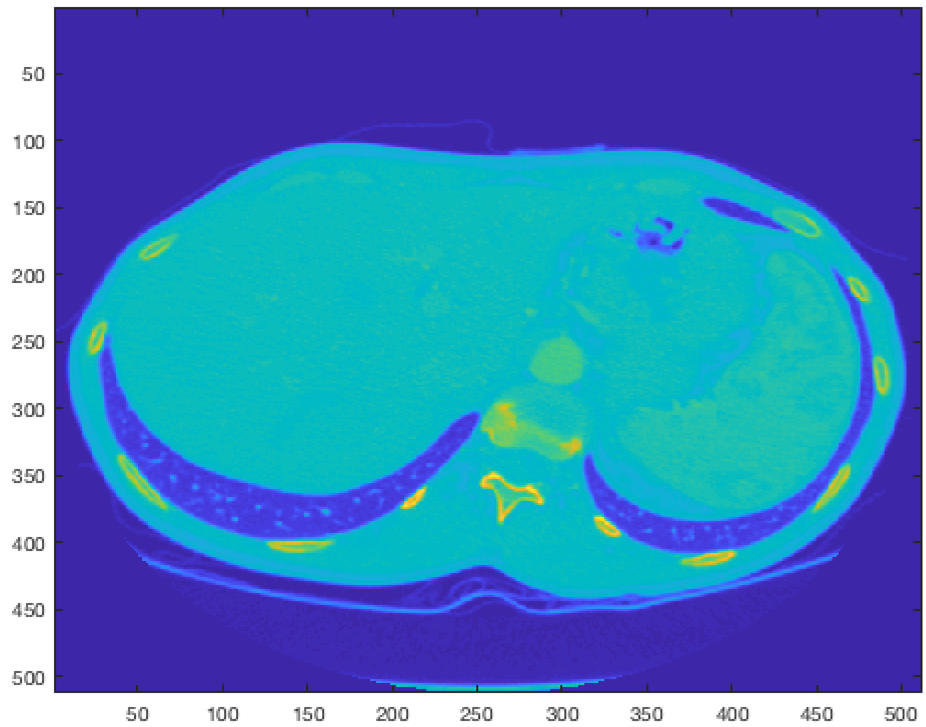
```
[rows, cols] = size(Template);
gridLengthX = rows; % grid width
gridlengthY = cols; % grid height
% define number of control points in each direction
```

```
numPointsX = 300+1;
numPointsY = 300+1;

% define number of time step until we observe current system
% state visualy
numSteps = 20;
```

Defining Second Set Of Experimental Intial Conditions

```
numTimeSteps = ceil(tFinal/dt);
dt = ceil(tFinal/numTimeSteps);

% Note that we must include a few points past the border in order
% in order to efficiently perform linear interpolation at image
 boundaries
% Maked sure to not generate a point for the image edge

% NOTE TODO: Must Fix this and resize image as well as create a bigger
% range of values in order to adequately perform linear interpolation
 at
% image edges
x = linspace(0, gridLengthX-2, numPointsX+1);
y = linspace(0, gridlengthY-2, numPointsY+1);

% compute the spacing between each control point in x and y direction
dx = ceil(gridLengthX/numPointsX);
dy = ceil(gridlengthY/numPointsY);

% create a meshgrid
[X, Y] = meshgrid(x,y);
x = ceil(x); y = ceil(y);
%[x,y]=meshgrid(x0:dx:LX,y0:dy:LY);
plot(X,Y,'*r');hold on;grid on
figure; imagesc(Template);
%[xx,yy]=meshgrid(0.1:0.1:1.1,0.1:0.1:1.1);
```


600
500
400
300
200
100
0
0
100
200
300
400
500
600



Third Initialization Sequence

initialize displacement field

```
Ux = zeros(numPointsX, numPointsY);
Uy = zeros(numPointsX, numPointsY);

% initialize velocity field
Vx = zeros(numPointsX, numPointsY);
Vy = zeros(numPointsX, numPointsY);

% for i=1:numPointsX;
%     for j=1:numPointsY;
%         %Ux(i,j)=randn(1,1);
%         %Uy(i,j)=randn(1,1);
%         Vx(i,j)=randn(1,1);
%         Vy(i,j)=randn(1,1);
%     end
% end
```

Solve for velocity based off of the PDE

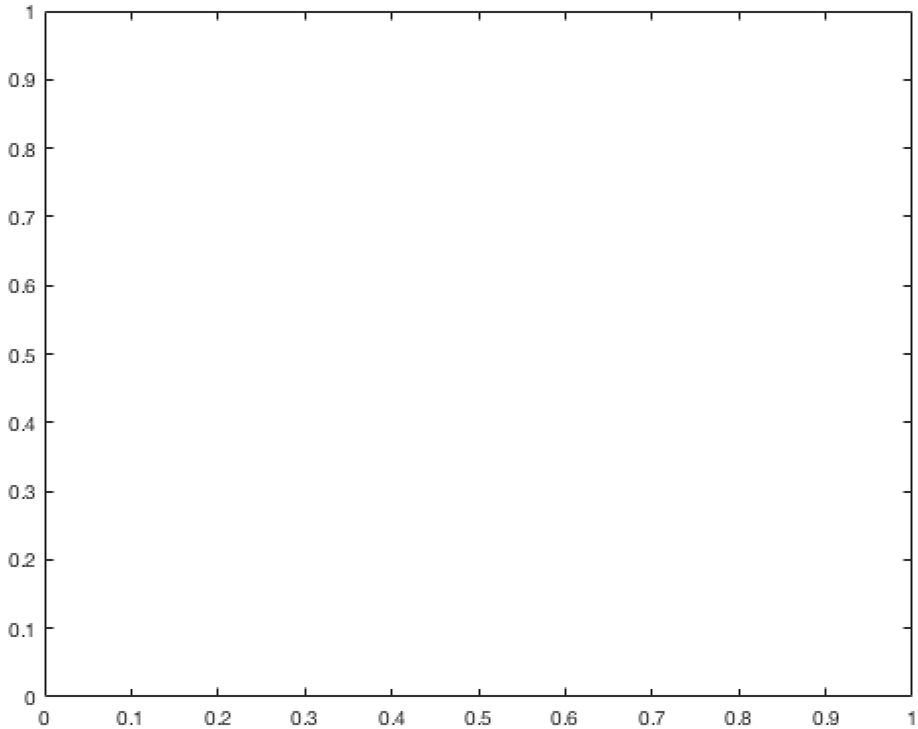
```
mu = 1;
lambda = 1;
```

```
% boundary conditions
uN = x*0+1;    vN = avg(x)*0;
uS = x*0;      vS = avg(x)*0;
uW = avg(y)*0; vW = y*0;
uE = avg(y)*0; vE = y*0;
%-------------------------------------------------------------------
fprintf('initialization')
centralDiffMat = full(gallery('tridiag',numPointsX,-1,2,-1));
% Define Fourier matrix for use later on
fourierMat = dftmtx(numPointsX);
fourierMatInv = inv(fourierMat);

figure;
for i=1:1
    %TEMPx(:,:,i) = Ux;
    %TEMPy(:,:,i) = Uy;
    %Wx = interp2(TEMPx(:,:,i), X(1:end-1, 1:end-1)-Ux);
    %Wy = interp2(TEMPy(:,:,i), Y(1:end-1, 1:end-1)-Uy);
    %Tx = interp2(Template,X(1:end-1, 1:end-1)-Wx-Ux);
    %Ty = interp2(Template,Y(1:end-1, 1:end-1)-Wy-Uy);

    % Force Field Computation
    force = forceField(Template, Source, ceil(x) ,ceil(y), Ux, Uy,dx,
 dy);

    drawnow
    % visualize the force field on the image
    visualize(force(:,:,1), force(:,:,2), X, Y, Diff);
    disp("Displaying force fields");
    pause(3);

    % Compute the first order partial differential equation of the
 vector
    % field in x and y direction as well as the 2nd order version
    [dVx_x, dVx_y] = gradient(Vx, dx, dy);
    [dVy_x, dVy_y] = gradient(Vy, dx, dy);
    [d2Vx_xx, d2Vx_xy] = gradient(dVx_x, dx, dy);       [d2Vx_yx,
 d2Vx_yy] = gradient(dVx_y, dx, dy);
    [d2Vy_xx, d2Vy_xy] = gradient(dVy_x, dx, dy);       [d2Vy_yx,
 d2Vy_yy] = gradient(dVy_y, dx, dy);



%
%
%     % compute 2nd order DIFFQ of V wrt xy
%     % Cast U as a vector
%     Vx_vec = Vx(:);    Vy_vec = Vy(:);
%     % Mixed derivative operator
%     Ax = kron(d2Vx_xx,d2Vx_yy);     Ay = kron(d2Vy_xx,d2Vy_yy);
%
%     Vx_xy_num = Ax*Vx_vec;    Vy_xy_num = Ay*Vy_vec;
%
%     d2Vx_xy = reshape(Vx_xy_num,numPointsX,numPointsY);
```

```
%     d2Vy_xy = reshape(Vy_xy_num,numPointsX,numPointsY);


    % Compute A which is a step in solving the PDE
    % Important to note that A = [A11, A12; A21, A22] is a circular
 matrix
    % since we utilize periodic boundary conditions for the expression
    % and has a dimension of (2N)X(2N) where N = n1*n2
    % Now, we want to solve the linear system of equation Av = F but
 due to
    % the high dimensionality of A we have to employ the FFT method to
    % diagonalize A in O(nlogn)
    A11 = ((mu + 2*lambda) .* d2Vx_xx) + (mu .* d2Vx_yy);
    A12 = (mu + lambda) .* d2Vy_xy;
    A21 = (mu + lambda) .* d2Vx_xy;
    A22 = (mu) .* d2Vy_xx + (mu + 2*lambda) .* d2Vy_yy;

    % Apply fourier transform to A(Circular Matrix) to diagonalize it
    A11 = fourierMat .* A11 .* fourierMatInv;
    A12 = fourierMat .* A12 .* fourierMatInv;
    A21 = fourierMat .* A21 .* fourierMatInv;
    A22 = fourierMat .* A22 .* fourierMatInv;

    % Apply moore penrose pseudoinverse to handle cases of singularity
 of A in a
    % special manner
    D11 = pinv(A11);
    D12 = pinv(A12);
    D21 = pinv(A21);
    D22 = pinv(A22);

    % update and solve for v
    Vx = force(1:end-1,1:end-1,1) .* D11 + force(1:end-1,1:end-1,2) .*
 D12;
    Vy = force(1:end-1,1:end-1,1) .* D21 +
 force(1:end-1,1:end-1,2) .*D22;
    drawnow
    visualize(Vx, Vy,X(2:end, 2:end), Y(2:end, 2:end), Diff);
    disp("Displaying velocity vector fields \n");
    pause(3);

    J_Ux11 = (1/2)*(centralDiffMat .* Ux); detJ_Ux11 = det(J_Ux11);
    J_Ux12 = (1/2)*(centralDiffMat .* Uy); detJ_Ux12 = det(J_Ux12);

    J_Uy21 = (1/2)*(centralDiffMat .* Uy); detJ_Uy21 = det(J_Uy21);
    J_Uy22 = (1/2)*(centralDiffMat .* Uy); detJ_Uy22 = det(J_Uy22);

    detSet = [detJ_Ux11,detJ_Ux12,detJ_Uy21,detJ_Uy22];

    % compute min of determinant of jacobian
    minVal = min(detSet);
    if(minVal < 0.5)
      % set U to 0
      %TEMPx(:,:,i) = Wx + Ux;
```

```
        %TEMPy(:,:,i) = Wy + Uy;

        % Must Implement Regridding Here In Order To Reconfigure V at
 the
        % next iteration
        Ux = 0 .* Ux;
        Uy = 0 .* Uy;
      else
         deltaUx = J_Ux11 .* Vx + J_Ux12 .* Vx;
         deltaUy = J_Uy21 .* Vy + J_Uy22 .* Vy;
         Px = norm(deltaUx);
         Py = norm(deltaUy);
         deltaX = min(1, 0.05/max(max(Px)));
         deltaY = min(1, 0.05/max(max(Py)));
         delta = min(deltaX,deltaY);
         Ux = Ux + delta .* deltaUx;
         Uy = Uy + delta .* deltaUy
      end

      drawnow
      visualize(Ux, Uy,X(2:end, 2:end), Y(2:end, 2:end), Diff);
      pause(3);
      disp("Displaying displacement vector fields \n");

  end
  %Ux = Wx + Ux ;
  %Uy = Wy + Uy
  %Toutx = interp2(Template, X(1:end-1, 1:end-1) - Ux);
  %Touty = interp2(Template, Y(1:end-1, 1:end-1) - Uy)
  figure; quiver(X(1:end-2, 1:end-1),Y(1:end-2,1:end-1),Vx,Vy');
```

initializationDisplaying force fields
Displaying velocity vector fields \n
Displaying displacement vector fields \n




```
%clear all; close all;
%mit18086_navierstokes()
```

Published with MATLAB® R2017b