EBME-461 Di Fan(dxf244); Phoebe Liu(xxl1024); Alexis Wang(xxw962)

Q1

```
function R=myNCC(image1,image2)
s1=size(image1);
s2=size(image2);
% n parameter to control similarity metrics
% n=1 normalized cross-correlation 2sum of absolute difference  3 sum of square
% error
n=1;

if s1~=s2
    disp('size of image not match');
    return
end
%normalized cross-correlation
if n==1
   bar_fl=mean(image1,"all");
   bar_drr=mean(image2,'all');
   Dfl=image1-bar_fl;
   Ddrr=image2-bar_drr;
   corr=Dfl.*Ddrr;
   a=sum(corr,"all");
   c=sum(Dfl.^2,"all");
   d=sum(Ddrr.^2,'all');

   R=a/sqrt(c*d);
end

end
```

```
function error=SSE(fixed, moving, tx, ty, param)
   if isfield(param, 'scaling')
      scaleFactor= param.scaling;
   else
      scaleFactor=1;
   end
   Tx= tx*scaleFactor;
   Ty= ty*scaleFactor;
   translated = imtranslate(moving,[Tx,Ty], "OutputView","same");

   differenceImage = abs(fixed, translated);
   error = sum(differenceImage(4:end,3:end).^2,"all");
end
```

Q2

Import image and convert to gray scale,double. Record size of image.

```
clc;clear;
load('Contrast2.mat');
load('Contrast1.mat');
[m,n]=size(Contrast2);
origin=Contrast1;
translated=Contrast2;
```

Calculate myNCC

```
R_beforeregis=myNCC(origin(4:end,3:end),translated(4:end,3:end));
diff1=origin-translated;
```

In order to translate the contrast2 back. We should translate -1.8 in x and -2.1 in y.

```
dx=-1.8;
dy=-2.1;
```

Now translate in x and y direction and calculate pixel value using intepolation.(backward transformation)

```
% Create a grid of coordinates
[X, Y] = meshgrid(1:n, 1:m);

% Compute the translated coordinates
X_regis = X + dx;
Y_regis = Y + dy;

% Initialize the output image with NaNs
regis_image = NaN(m, n);

% Find the integer coordinates surrounding the translated coordinates
x1 = floor(X_regis);
x2 = ceil(X_regis);
y1 = floor(Y_regis);
y2 = ceil(Y_regis);

% Identify valid interpolation points (within image bounds)
validMask = x1 >= 1 & x2 <= n & y1 >= 1 & y2 <= m;

% Interpolation weights
wx = X_regis - x1;
wy = Y_regis - y1;

% Retrieve pixel values at the four surrounding points for valid points
Q11 = NaN(m, n); Q21 = NaN(m, n);
Q12 = NaN(m, n); Q22 = NaN(m, n);

Q11(validMask) = double(translated(sub2ind([m, n], y1(validMask), x1(validMask))));
```

```
Q21(validMask) = double(translated(sub2ind([m, n], y1(validMask), x2(validMask))));
Q12(validMask) = double(translated(sub2ind([m, n], y2(validMask), x1(validMask))));
Q22(validMask) = double(translated(sub2ind([m, n], y2(validMask), x2(validMask))));

% Perform bilinear interpolation for valid points
regis_image(validMask) = (1 - wx(validMask)) .* (1 - wy(validMask)) .*
Q11(validMask) + ...
    wx(validMask) .* (1 - wy(validMask)) .* Q21(validMask) + ...
    (1 - wx(validMask)) .* wy(validMask) .* Q12(validMask) + ...
    wx(validMask) .* wy(validMask) .* Q22(validMask);
regis_image(isnan(regis_image)) = 0;
```

Calculate myNCC and subtract image with Contrast1

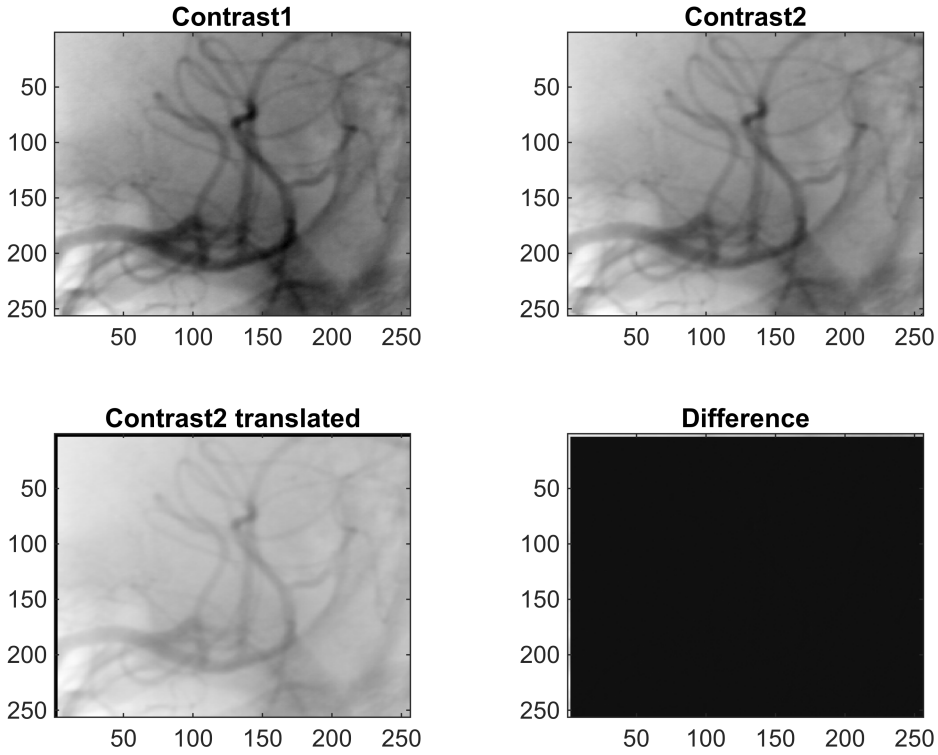
```
Difference=origin-regis_image;
% regis_image=uint8(regis_image);
% origin=uint8(origin);
% translated=uint8(translated);
R_afterregistration=myNCC(origin(4:end,3:end),regis_image(4:end,3:end));
```

Print the result value and show the difference image with scale bar

```
fprintf('The result before registration is %d and after registration is
%d.\n',R_beforeregis,R_afterregistration);
```

```
The result before registration is 9.778922e-01 and after registration is 9.998184e-01.
```

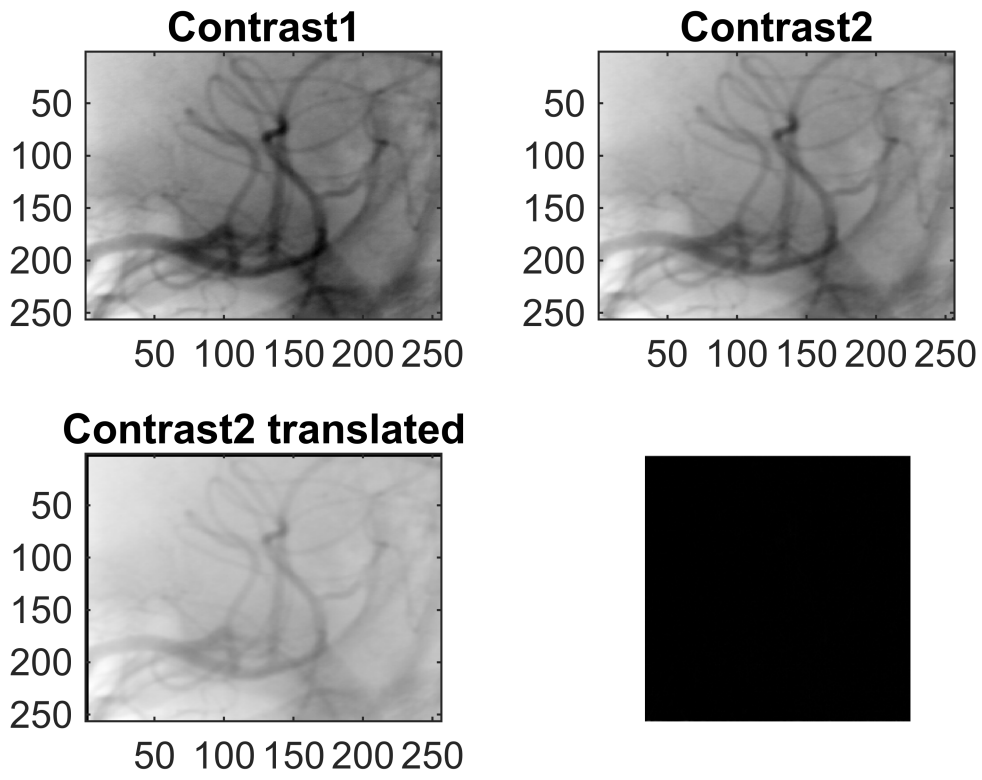
```


tiledlayout(2,2)
nexttile
imagesc(origin)
colormap('gray')
title('Contrast1')

nexttile
imagesc(translated)
title('Contrast2')
colormap('gray')
nexttile
imagesc(regis_image)
title('Contrast2 translated')
colormap('gray')
nexttile
imagesc(Difference)
title('Difference')
colormap('gray')
```



Q3

NEW

```
%initialize first guess for translation, [tx, ty]
t_i= [0,0];
frames = struct('cdata', {}, 'colormap', {});
%initialize array
param=struct();
param.scaling=1;

%calling the cost function SSE
costF= @(t) registrationcost(t, origin, translated);
%costF= @(t) SSE(origin, translated, t(1),t(2), param );
% cost_test= SSE(origin, translated, -1.8, -2.1);
% display(cost_test)

%run fminsearch on optimal translation parameters
% and custom options to show iterations
options = optimset( 'TolFun',1e-3, 'TolX',1e-3);
%     'OutputFcn', @(x, optimValues, state) ...
%     updateFrames(x, optimValues, state, frames)
                    %'OutputFcn',@regOutFun);
[t_optimal, fval]= fminsearch(costF, t_i, options);
```

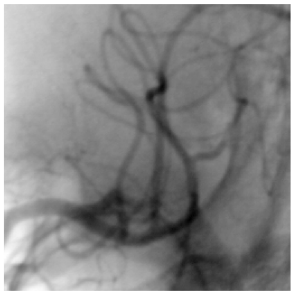



```
fprintf(['The optimal translation parameters t_x is %.4f, t_y is %4f, ' ...
    'and minimal cost function is %d\n'], ...
    t_optimal(1), t_optimal(2), fval);
```

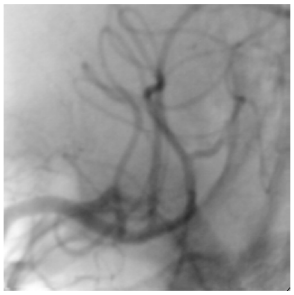
```
The optimal translation parameters t_x is 1.8398, t_y is 2.064162, and minimal cost function is 7.100666e+04
```

```
%subtracted image without registration
diffImg_noReg= abs(origin-translated);

figure;
subplot(2,2,1), imshow(origin,[]), title('Contrast 1');
subplot(2,2,2), imshow(translated,[]), title('Contrast 2');
subplot(2,2,3), imshow(uint8(diffImg_noReg)), title('Subtracted Image without
Registration');
```

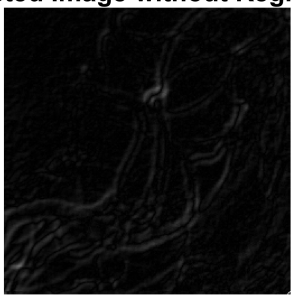
Contrast 1



Contrast 2



Subtracted Image without Registration



```
figure;
axis off;

movie(frames);
```

```
function cost = registrationcost(params, fixedImage, movingImage)
    persistent frames;
    if isempty(frames)
        frames = struct('cdata', {}, 'colormap', {});
    end

    % Extract translation parameters
    tx = params(1);
    ty = params(2);

    % Translate the moving image
    translatedImage = imtranslate(movingImage, [tx, ty]);

    % Compute the difference image
    differenceImage = abs(fixedImage - translatedImage);

    % Compute the sum of squared errors
    cost = sum(differenceImage(4:end,3:end).^2,"all");

    % Capture the difference image as a frame
    figure(100);
    imshow(uint8(differenceImage));
    drawnow;
    frame = getframe(gcf);
```

Q4

```
t_i= [0,0];
%initialize array
param=struct();
param.scaling=1;

tolFun = [1e-1, 1e-3, 1e-5]; % Different TolFun values
tolX = [1e-1, 1e-3, 1e-5];   % Different TolX values
scaling= [1,10,100] % scaling factor
```

```
scaling = 1×3
     1    10   100
```

```
costF= @(t) SSE(origin, translated, t(1),t(2),param );

%initialize arrays to store result
results_fun=[];
results_x=[];
results_scaling=[];
%chaing tolerance for function value
for f=1:length(tolFun)
    options=optimset('TolFun', tolFun(f),'Display','off');
    [t_op, fvalue,exitflag, output]=fminsearch(costF, t_i, options)
    r1.TolFun= tolFun(f);
    r1.Cost= fvalue;
    r1.Iteration= output.iterations;
    if exitflag==1;
        r1.Outcomes='Sucess';
    else
        r1.Outcomes='Failure';
    end
    results_fun= [results_fun, r1];
end
```

```
t_op = 1×2
    1.8398    2.0642
fvalue = 7.1007e+04
exitflag = 1
output = struct with fields:
    iterations: 83
     funcCount: 160
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
t_op = 1×2
    1.8398    2.0642
fvalue = 7.1007e+04
exitflag = 1
output = struct with fields:
    iterations: 86
     funcCount: 166
```

```
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
t_op = 1×2
    1.8398    2.0642
fvalue = 7.1007e+04
exitflag = 1
output = struct with fields:
    iterations: 94
     funcCount: 182
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
```

```
table1= struct2table(results_fun);
disp(table1);
```

TolFun	Cost	Iteration	Outcomes
0.1	71007	83	{'Sucess'}
0.001	71007	86	{'Sucess'}
1e-05	71007	94	{'Sucess'}

```
%changing tolerance for x value
for x= 1:length(tolX)
    options= optimset('TolX', tolX(x),'Display','off');
    [t_op, fvalue,exitflag, output]=fminsearch(costF, t_i, options)
    r2.TolX= tolX(x);
    r2.Cost=fvalue;
    r2.Iteration= output.iterations;
    if exitflag==1;
        r2.Outcomes='Sucess';
    else
        r2.Outcomes='Failure';
    end
    results_x=[results_x, r2];
end
```

```
t_op = 1×2
    1.8399    2.0642
fvalue = 7.1007e+04
exitflag = 1
output = struct with fields:
    iterations: 89
     funcCount: 172
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
t_op = 1×2
    1.8399    2.0642
fvalue = 7.1007e+04
exitflag = 1
output = struct with fields:
    iterations: 89
     funcCount: 172
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
t_op = 1×2
    1.8399    2.0642
fvalue = 7.1007e+04
exitflag = 1
```

```
output = struct with fields:
    iterations: 90
     funcCount: 174
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
```

```
table2= struct2table(results_x);
disp(table2);
```

TolX	Cost	Iteration	Outcomes
0.1	71007	89	{'Sucess'}
0.001	71007	89	{'Sucess'}
1e-05	71007	90	{'Sucess'}

```
%changing scaling factor
for s=1:length(scaling)
    param.scaling= scaling(s);
    options=optimset('Display','off');
    [t_op, fvalue,exitflag, output]=fminsearch(costF, t_i, options)
    r3.Scale= scaling(s);
    r3.Cost=fvalue;
    r3.Iteration= output.iterations;
    if exitflag==1;
        r3.Outcomes='Sucess';
    else
        r3.Outcomes='Failure';
    end
    results_scaling=[results_scaling, r3];
end
```

```
t_op = 1×2
    1.8399    2.0642
fvalue = 7.1007e+04
exitflag = 1
output = struct with fields:
    iterations: 89
     funcCount: 172
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
t_op = 1×2
    1.8399    2.0642
fvalue = 7.1007e+04
exitflag = 1
output = struct with fields:
    iterations: 89
     funcCount: 172
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
t_op = 1×2
    1.8399    2.0642
fvalue = 7.1007e+04
exitflag = 1
output = struct with fields:
    iterations: 89
     funcCount: 172
```

```
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x satisfies the termination criteria using OPTIONS.TolX of 1
```

```
table3= struct2table(results_scaling);
disp(table3);
```

```
    Scale    Cost     Iteration     Outcomes
    _____    _____    _________    _________

      1      71007       89        {'Sucess'}
     10      71007       89        {'Sucess'}
    100      71007       89        {'Sucess'}
```

Q5

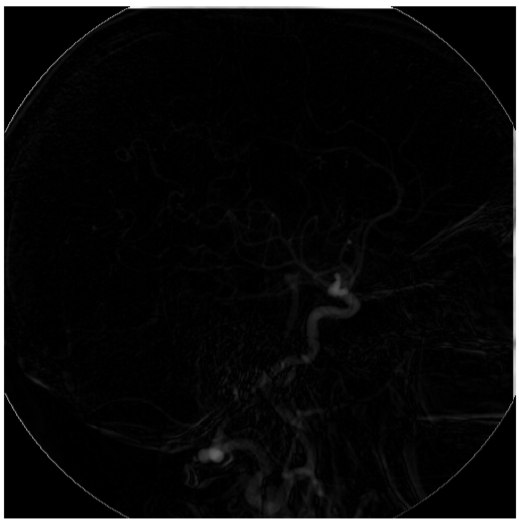
```
% 1. Initial Setup
clc; clear;
live = imread('live_new.tif');    % Reference image (with contrast)
mask = imread('mask_new.tif');    % Floating image (no contrast)
live=im2gray(live);
mask=im2gray(mask);
[m, n] = size(live);

% Convert to double for numerical operations
live = double(live);
mask = double(mask);

% Calculate initial subtrPaction (before registration)D:\drive\OneDrive - Case
Western Reserve University\FILE\2025spring\EBME461 Image\GroupProject\HW04
diff_before = live - mask;

% Setup optimization parameters
param = struct();
param.scaling = 1;  % Start with no scaling
```

cost function

```
function cost = registrationcost(params, fixedImage, movingImage)
    persistent frames;
    if isempty(frames)
        frames = struct('cdata', {}, 'colormap', {});
    end

    % Extract translation parameters
    tx = params(1);
    ty = params(2);

    % Translate the moving image
    translatedImage = imtranslate(movingImage, [tx, ty]);
    row_i= ceil(abs(ty)+1);
    col_i= ceil(abs(tx)+1);

    % Compute the difference image
    differenceImage = abs(fixedImage - translatedImage);

    % Compute the sum of squared errors
    cost = sum(differenceImage(row_i:end,col_i:end).^2,"all");

    % Capture the difference image as a frame
    figure(100);
    imshow(uint8(differenceImage));
    drawnow;
```

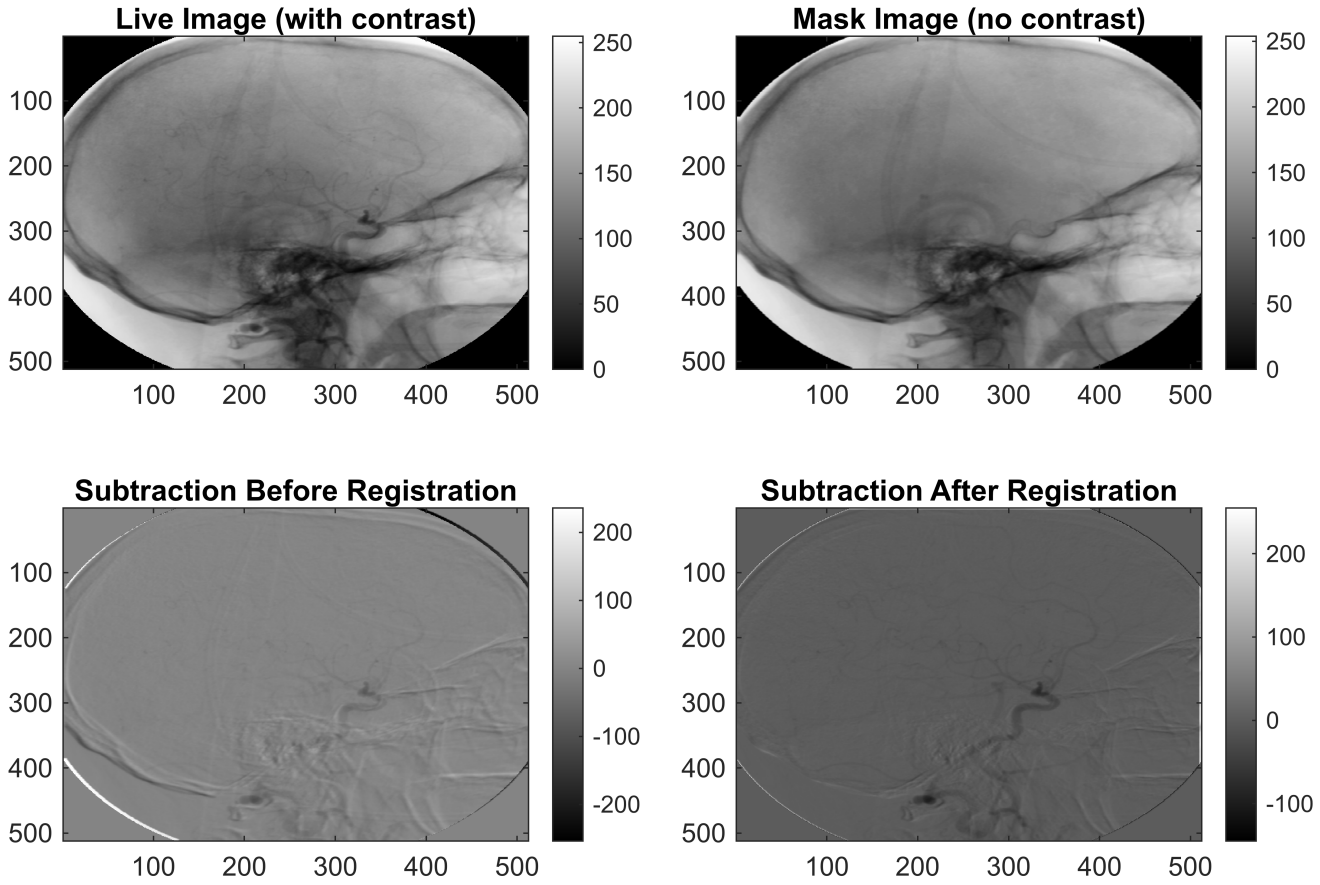
```
    frame = getframe(gcf);

    % Store the frame
    frames(end+1) = frame;

    % Assign the updated frames to the base workspace
    assignin('base', 'frames', frames);
end
```

```
% Setup cost function
costF= @(t) registrationcost(t, live, mask);

% Create the output function with additional parameters
outfun = @(x, optimVals, state) regOutFun(x, optimVals, state, live, mask, param);

% Set optimization options
options = optimset( 'TolFun',1e-3, 'TolX',1e-3);


% Run optimization
t_i= [0,0];
[t_optimal, fval]= fminsearch(costF, t_i, options);
```



```
% Apply optimal translation to create registered image
translated = imtranslate(mask, [t_optimal(1), t_optimal(2)], 'OutputView', 'same');

% Calculate final subtraction
diff_after = live - translated;
```

```
% Display results
figure('Position', [100 100 1200 800]);
tiledlayout(2,2)

% Original live image
nexttile
imagesc(uint8(live))
colormap('gray')
title('Live Image (with contrast)')
colorbar

% Original mask image
nexttile
imagesc(uint8(mask))
title('Mask Image (no contrast)')
colormap('gray')
colorbar

% Subtraction before registration
nexttile
imagesc(diff_before)
title('Subtraction Before Registration')
colormap('gray')
colorbar

% Subtraction after registration
nexttile
imagesc(diff_after)
title('Subtraction After Registration')
colormap('gray')
colorbar
```




```
% Print optimization results
fprintf('\nOptimization Results:\n');
```

```
Optimization Results:
```

```
fprintf('Inital cost value: %.4f\n', (sum(diff_before.^2, "all")));
```

```
Inital cost value: 94956400.0000
```

```
fprintf('Optimal translation parameters:\n');
```

```
Optimal translation parameters:
```

```
fprintf('X translation: %.4f pixels\n', t_optimal(1));
```

```
X translation: -2.7436 pixels
```

```
fprintf('Y translation: %.4f pixels\n', t_optimal(2));
```

```
Y translation: 2.3570 pixels
```

```
fprintf('Final cost value: %.4f\n', fval);
```

```
Final cost value: 39798779.2723
```

```
% Save results
% save('dsa_registration_results.mat', 't_optimal', 'fval', 'output');
```

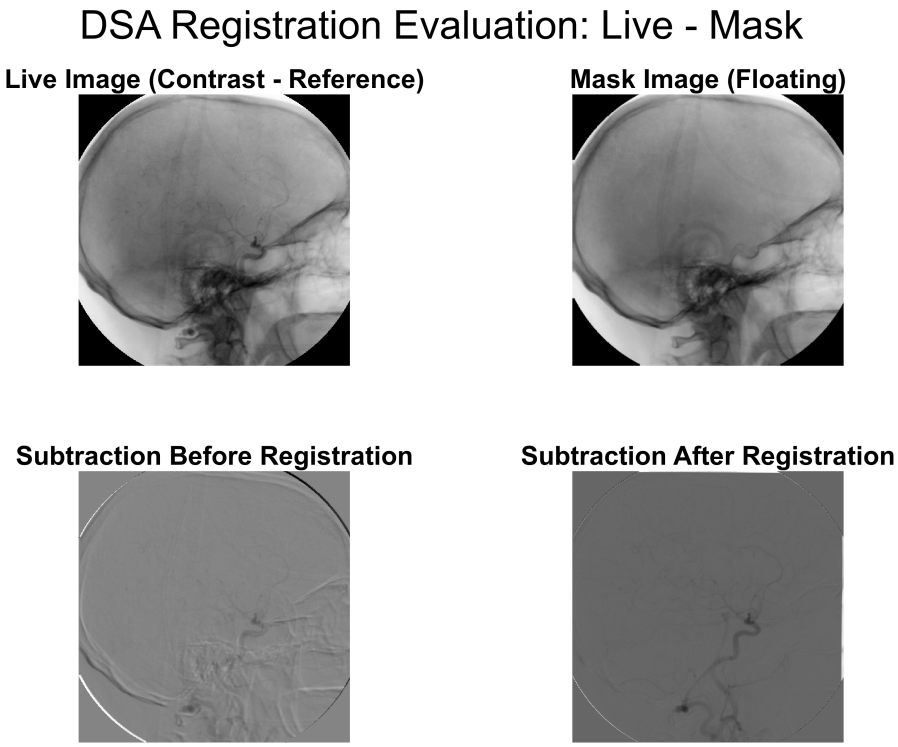
Q6

```
%% Clear workspace and close figures
clear; close all; clc;

%% Read the images
% Contrast image (live_new) is the reference image.
live = imread('live_new.tif');
% Mask image (mask_new) is the floating image.
mask = imread('mask_new.tif');

% Convert to grayscale if necessary
if size(live,3) == 3
    live = rgb2gray(live);
end
if size(mask,3) == 3
    mask = rgb2gray(mask);
end

% Convert images to double precision for processing
live = im2double(live);
mask = im2double(mask);

%% Compute the Subtracted Image WITHOUT Registration
% Always subtract the mask from the live image (live - mask) so that the vessels
appear dark.
diffBefore = live - mask;
```

1- Iteration array=[5000 400 200], Accumulated Field Smoothing (AFS)= 30

```
%% Perform Non-Rigid Registration
numIterations = [5000, 400, 200];
% numIterations = [50, 20, 5];
AFS = 30;

[~, maskReg] = imregdemons(mask, live, numIterations, 'AccumulatedFieldSmoothing',
AFS);

%% Compute the Subtracted Image WITH Registration
diffAfter = live - maskReg;

%% Display the Evaluation Figures
% The evaluation includes:
%   1. The original live (contrast) image (reference)
%   2. The original mask image (floating)
%   3. The subtracted image BEFORE registration (live - mask)
%   4. The subtracted image AFTER registration (live - registered mask)
figure('Name','DSA Registration Evaluation','NumberTitle','off');

subplot(2,2,1);
imshow(live, []);
```

```
title('Live Image (Contrast - Reference)');

subplot(2,2,2);
imshow(mask, []);
title('Mask Image (Floating)');

subplot(2,2,3);
imshow(diffBefore, []);
title('Subtraction Before Registration');

subplot(2,2,4);
imshow(diffAfter, []);
title('Subtraction After Registration');

sgtitle('DSA Registration Evaluation: Live - Mask');
```

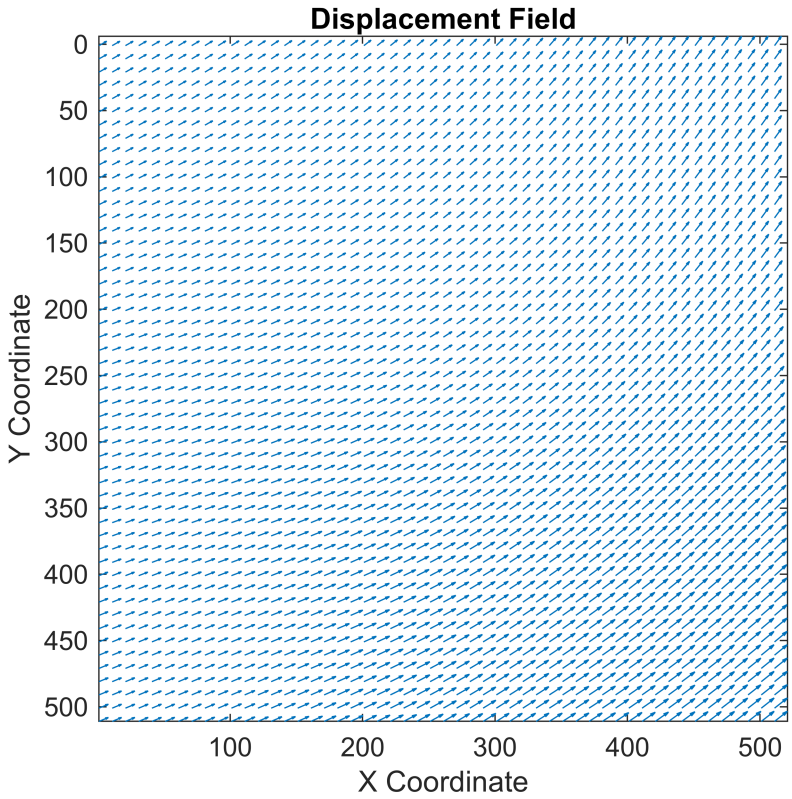



```
% Save the Output Images
imwrite(live, 'live_new.png');
imwrite(mask, 'mask_new.png');
imwrite(mat2gray(diffBefore), 'diff_before_registration.png');
imwrite(mat2gray(diffAfter), 'diff_after_registration.png');
imwrite(mat2gray(maskReg), 'mask_registered.png');
```

Displacement matrix

```
% Now extract the displacement field as the first output
```

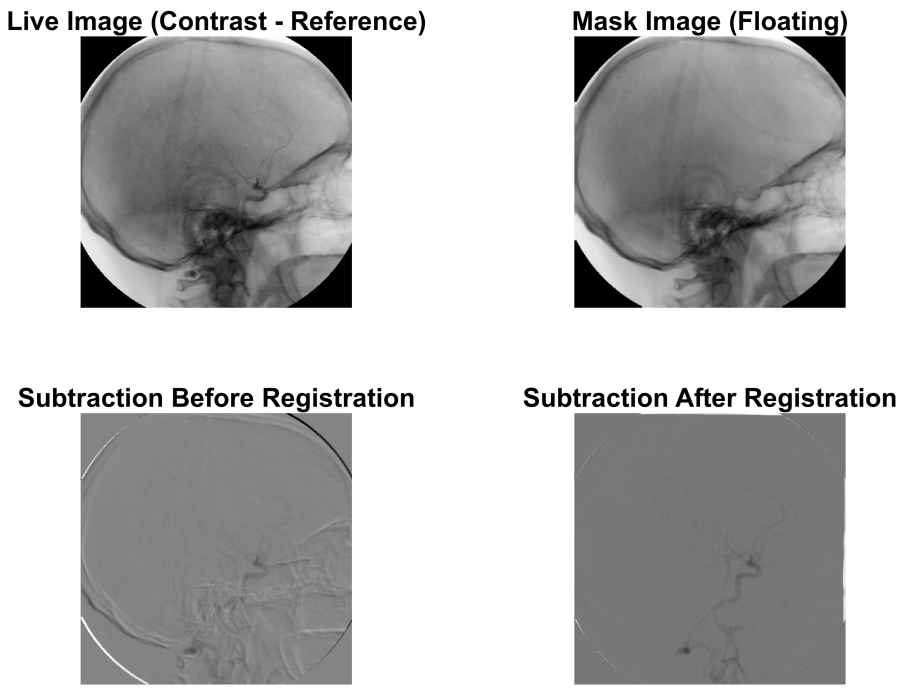
```
[dispField, maskReg] = imregdemons(mask, live, numIterations,
'AccumulatedFieldSmoothing', AFS);

[rows, cols, ~] = size(dispField);

% Create a grid of coordinates corresponding to the displacement field
[X, Y] = meshgrid(1:cols, 1:rows);

subsample = 10;
Xsub = X(1:subsample:end, 1:subsample:end);
Ysub = Y(1:subsample:end, 1:subsample:end);
u = dispField(1:subsample:end, 1:subsample:end, 1);  % x-component
v = dispField(1:subsample:end, 1:subsample:end, 2);  % y-component

% Create the quiver plot
figure;
quiver(Xsub, Ysub, u, v, 'AutoScale', 'on');
title('Displacement Field');
xlabel('X Coordinate');
ylabel('Y Coordinate');
axis image;   % Keep aspect ratio
set(gca, 'YDir','reverse');  % Optional: reverse y-axis if image coordinates are
used
```



2- Iteration array=[5000 400 200], Accumulated Field Smoothing (AFS)= 3

```
%% Perform Non-Rigid Registration
```

```
numIterations = [5000, 400, 200];
% numIterations = [50, 20, 5];
AFS = 3;

[~, maskReg] = imregdemons(mask, live, numIterations, 'AccumulatedFieldSmoothing',
AFS);

%% Compute the Subtracted Image WITH Registration
diffAfter = live - maskReg;


figure('Name','DSA Registration Evaluation','NumberTitle','off');

subplot(2,2,1);
imshow(live, []);
title('Live Image (Contrast - Reference)');

subplot(2,2,2);
imshow(mask, []);
title('Mask Image (Floating)');

subplot(2,2,3);
imshow(diffBefore, []);
title('Subtraction Before Registration');

subplot(2,2,4);
imshow(diffAfter, []);
title('Subtraction After Registration');

sgtitle('DSA Registration Evaluation: Live - Mask');
```

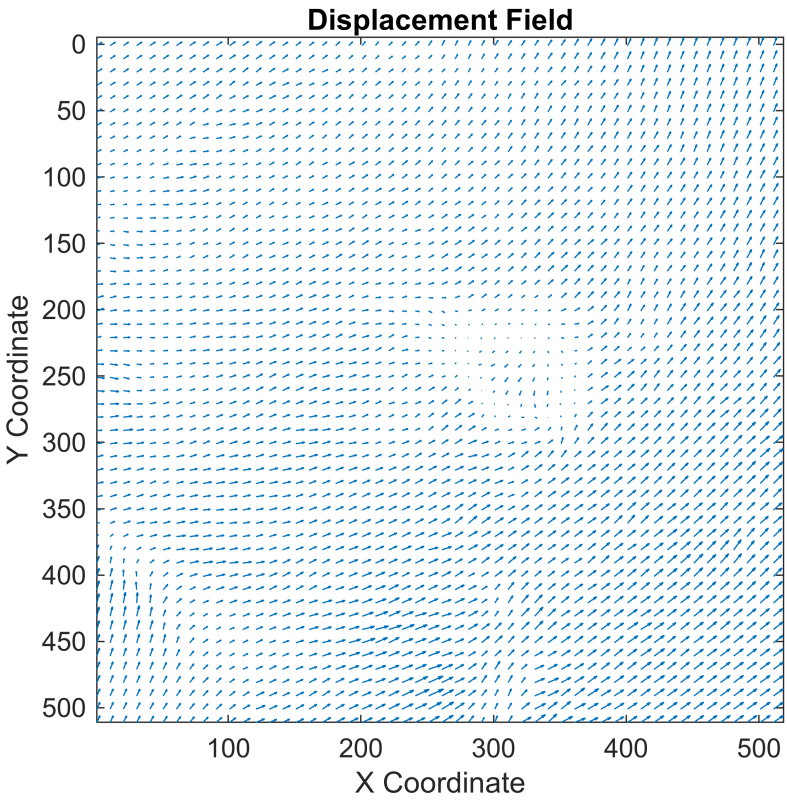



```
%% Save the Output Images
imwrite(live, 'live_new.png');
imwrite(mask, 'mask_new.png');
imwrite(mat2gray(diffBefore), 'diff_before_registration.png');
imwrite(mat2gray(diffAfter), 'diff_after_registration.png');
imwrite(mat2gray(maskReg), 'mask_registered.png');
```

matrix

```
% Now extract the displacement field as the first output
[dispField, maskReg] = imregdemons(mask, live, numIterations,
'AccumulatedFieldSmoothing', AFS);

%% Visualize the Displacement Field Using a Quiver Plot

% Get the size of the displacement field (assumed MxNx2)
[rows, cols, ~] = size(dispField);

% Create a grid of coordinates corresponding to the displacement field
[X, Y] = meshgrid(1:cols, 1:rows);

subsample = 10;
Xsub = X(1:subsample:end, 1:subsample:end);
```

```
Ysub = Y(1:subsample:end, 1:subsample:end);
u = dispField(1:subsample:end, 1:subsample:end, 1);  % x-component
v = dispField(1:subsample:end, 1:subsample:end, 2);  % y-component

% Create the quiver plot
figure;
quiver(Xsub, Ysub, u, v, 'AutoScale', 'on');
title('Displacement Field');
xlabel('X Coordinate');
ylabel('Y Coordinate');
axis image;   % Keep aspect ratio
set(gca, 'YDir','reverse');  % Optional: reverse y-axis if image coordinates are
used
```



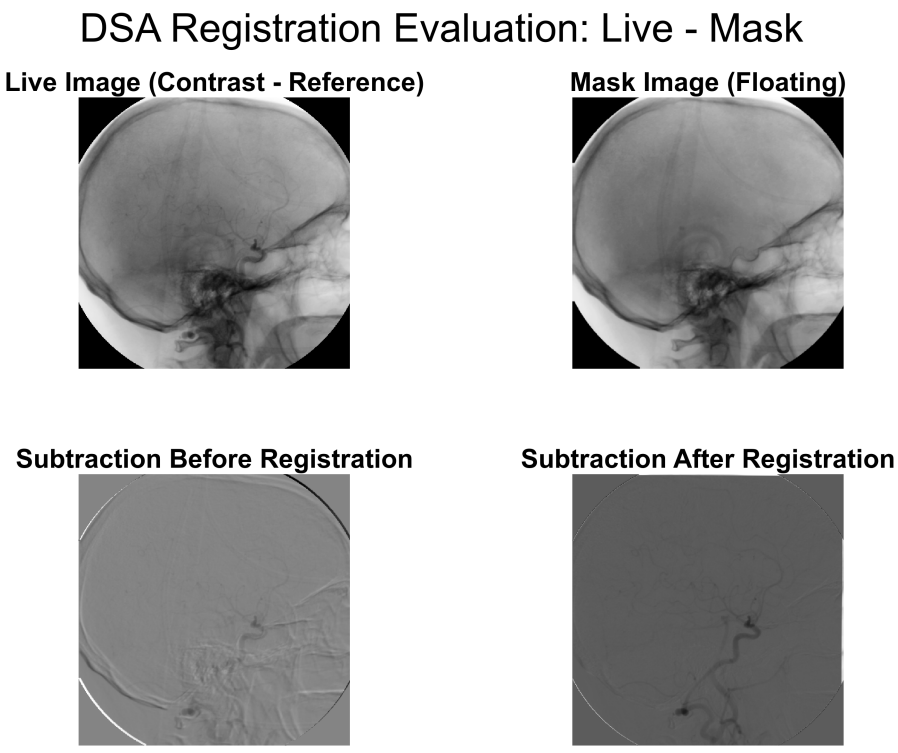
3- Iteration array=[50 20 5], Accumulated Field Smoothing (AFS)= 30

```
%% Perform Non-Rigid Registration
numIterations = [50, 20, 5]
```

```
numIterations = 1×3
    50    20     5
```

```
AFS = 30;

[~, maskReg] = imregdemons(mask, live, numIterations, 'AccumulatedFieldSmoothing',
AFS);
```

```
%% Compute the Subtracted Image WITH Registration
diffAfter = live - maskReg;


figure('Name','DSA Registration Evaluation','NumberTitle','off');

subplot(2,2,1);
imshow(live, []);
title('Live Image (Contrast - Reference)');

subplot(2,2,2);
imshow(mask, []);
title('Mask Image (Floating)');

subplot(2,2,3);
imshow(diffBefore, []);
title('Subtraction Before Registration');

subplot(2,2,4);
imshow(diffAfter, []);
title('Subtraction After Registration');

sgtitle('DSA Registration Evaluation: Live - Mask');
```

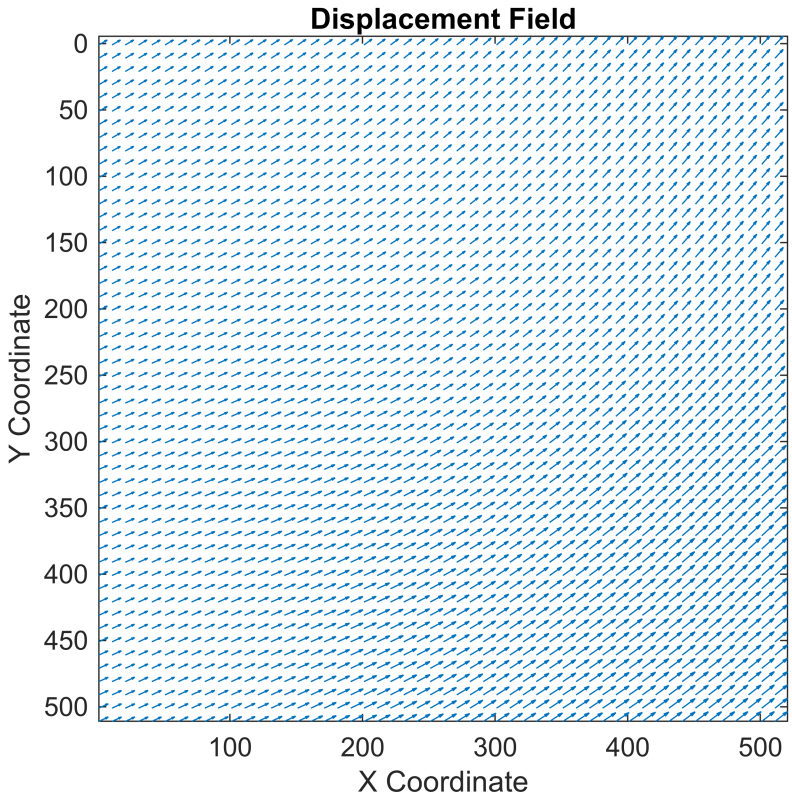



```
%% Save the Output Images
```

```
% Since subtraction images might have negative values, we use mat2gray to scale
them to [0,1]
imwrite(live, 'live_new.png');
imwrite(mask, 'mask_new.png');
imwrite(mat2gray(diffBefore), 'diff_before_registration.png');
imwrite(mat2gray(diffAfter), 'diff_after_registration.png');
imwrite(mat2gray(maskReg), 'mask_registered.png');
```

matrix

```
% Now extract the displacement field as the first output
[dispField, maskReg] = imregdemons(mask, live, numIterations,
'AccumulatedFieldSmoothing', AFS);

[rows, cols, ~] = size(dispField);

% Create a grid of coordinates corresponding to the displacement field
[X, Y] = meshgrid(1:cols, 1:rows);

subsample = 10;
Xsub = X(1:subsample:end, 1:subsample:end);
Ysub = Y(1:subsample:end, 1:subsample:end);
u = dispField(1:subsample:end, 1:subsample:end, 1);  % x-component
v = dispField(1:subsample:end, 1:subsample:end, 2);  % y-component

% Create the quiver plot
figure;
quiver(Xsub, Ysub, u, v, 'AutoScale', 'on');
title('Displacement Field');
xlabel('X Coordinate');
ylabel('Y Coordinate');
axis image;   % Keep aspect ratio
set(gca, 'YDir','reverse');  % Optional: reverse y-axis if image coordinates are
used
```



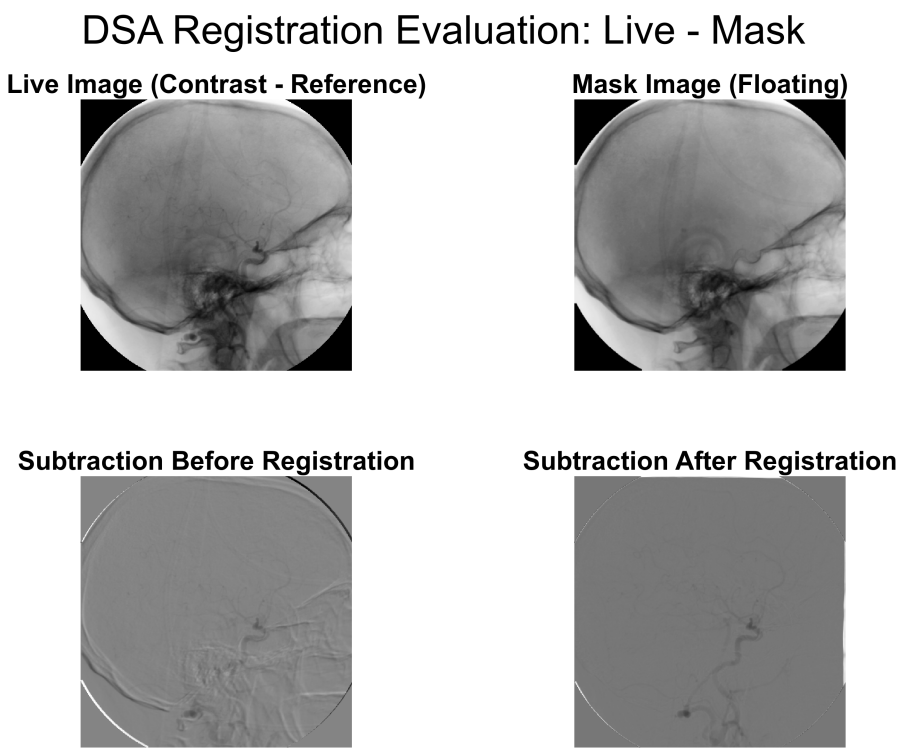
4- Iteration array=[50 20 5], Accumulated Field Smoothing (AFS)= 3

```
%% Perform Non-Rigid Registration

numIterations = [50, 20, 5]
```

```
numIterations = 1×3
    50    20     5
```

```
AFS = 3;

[~, maskReg] = imregdemons(mask, live, numIterations, 'AccumulatedFieldSmoothing',
AFS);

%% Compute the Subtracted Image WITH Registration
diffAfter = live - maskReg;


figure('Name','DSA Registration Evaluation','NumberTitle','off');

subplot(2,2,1);
imshow(live, []);
title('Live Image (Contrast - Reference)');

subplot(2,2,2);
imshow(mask, []);
```

```
title('Mask Image (Floating)');

subplot(2,2,3);
imshow(diffBefore, []);
title('Subtraction Before Registration');

subplot(2,2,4);
imshow(diffAfter, []);
title('Subtraction After Registration');

sgtitle('DSA Registration Evaluation: Live - Mask');
```




```
%% Save the Output Images
% Since subtraction images might have negative values, we use mat2gray to scale
them to [0,1]
imwrite(live, 'live_new.png');
imwrite(mask, 'mask_new.png');
imwrite(mat2gray(diffBefore), 'diff_before_registration.png');
imwrite(mat2gray(diffAfter), 'diff_after_registration.png');
imwrite(mat2gray(maskReg), 'mask_registered.png');
```

matix

```
% Now extract the displacement field as the first output
[dispField, maskReg] = imregdemons(mask, live, numIterations,
'AccumulatedFieldSmoothing', AFS);
```

```
[rows, cols, ~] = size(dispField);

% Create a grid of coordinates corresponding to the displacement field
[X, Y] = meshgrid(1:cols, 1:rows);

subsample = 10;
Xsub = X(1:subsample:end, 1:subsample:end);
Ysub = Y(1:subsample:end, 1:subsample:end);
u = dispField(1:subsample:end, 1:subsample:end, 1);  % x-component
v = dispField(1:subsample:end, 1:subsample:end, 2);  % y-component

% Create the quiver plot
figure;
quiver(Xsub, Ysub, u, v, 'AutoScale', 'on');
title('Displacement Field');
xlabel('X Coordinate');
ylabel('Y Coordinate');
axis image;   % Keep aspect ratio
set(gca, 'YDir','reverse');  % Optional: reverse y-axis if image coordinates are
used
```