```
function newton_joshua_a04
    clear all; close all;

    % Dimension for square search window
    inputSearchRange = 5;

    % Choose target image, anchor image, execute EBMA algorithm
    [fileA,pathA] = uigetfile('*.tif');
    if ~isequal(fileA,0)
        [fileT,pathT] = uigetfile('*.tif');
        if ~isequal(fileT,0)
            inputAnchorFrame = imread(strcat(pathA,fileA));
            inputTargetFrame = imread(strcat(pathT,fileT));
            ebma(inputTargetFrame, inputAnchorFrame, inputSearchRange);
        end
    end
end

function [predictedFrame, errorFrame] = ...
    ebma(targetFrame, anchorFrame, searchRange, blockSize)
    %% Initialization steps for given input values
    % Setting a default value of 16 for blockSize when not supplied.
    if (~exist('blockSize', 'var'))
        blockSize = 16;
    end

    % Precalculating values needed for iteration
    [frameHeight,frameWidth] = size(anchorFrame);
    maxHeight = frameHeight-(blockSize-1);
    maxWidth = frameWidth-(blockSize-1);
    windowSize = (searchRange-1)/2;

    % Preallocating arrays for displacement vectors, predicted frame
    displacementV = zeros(25,32);
    displacementH = zeros(25,32);
    predictedFrame = zeros(frameHeight,frameWidth);
```

```
%% EBMA algorithm
% Iterating through location of current anchor frame block
for pAnchorV = 1:blockSize:maxHeight
    for pAnchorH = 1:blockSize:maxWidth
        % Reset lowest error condition
        lowestError = Inf;

        % Storing values contained in current anchor frame block
        anchorBlock = anchorFrame(pAnchorV:pAnchorV+blockSize-1,...
                                  pAnchorH:pAnchorH+blockSize-1);

        % Iterating through search location (relative to current block)
        for pSearchV = -windowSize:windowSize
            for pSearchH = -windowSize:windowSize

                % Calculating location of current target frame block
                pTargetV = pAnchorV + pSearchV;
                pTargetH = pAnchorH + pSearchH;

                % Ensuring target frame pointers are within boundaries
                if (pTargetV > 0 && pTargetV < maxHeight+1 && ...
                    pTargetH > 0 && pTargetH < maxWidth+1)

                    % Storing values contained in current target block
                    targetBlock = targetFrame(pTargetV:pTargetV+blockSize-1, ...
                                              pTargetH:pTargetH+blockSize-1);

                    % MAD criterion calculation
                    differenceBlock = targetBlock - anchorBlock;
                    errorTerm = sum(abs(differenceBlock),'all');

                    % Check to see if lowest error in search window
                    if errorTerm < lowestError
                        % Updating error conditionv value
                        lowestError = errorTerm;

                        % Storing current best predicition in frame
                        predictedFrame(pAnchorV:pAnchorV+blockSize-1,...
                                       pAnchorH:pAnchorH+blockSize-1)...
                                       =targetBlock;

                        % Convert block location to indexes for quiver
                        % (how quiver interprets coordinates
                        % necessitates tweaks for vertical values)
                        pIndexV = (frameHeight/blockSize) + 1 - ...
                                  ((pAnchorV-1)/blockSize + 1);
                        pIndexH = ((pAnchorH-1)/blockSize + 1);

                        % Update displacement for lowest error block
                        displacementV(pIndexV,pIndexH) = -pSearchV;
                        displacementH(pIndexV,pIndexH) = pSearchH;
```

```
                    end
                end
            end
        end
    end
end

%% Display results of EBMA algorithm
% Convert predicted frame to uint8 grayscale image formatting
predictedFrame = uint8(predictedFrame);

% Display displacement vector plot
figure(1);
quiver(displacementH,displacementV);
axis([0 33 0 26]);

% Display predicted frame
figure(2);
imshow(predictedFrame);

% Display error between predicted and actual frames
figure(3);
errorFrame = anchorFrame - predictedFrame;
imshow(imcomplement(errorFrame));

% Calculate PSNR for predicted frame
PSNR = 10*log10(255*255/mean(mean((anchorFrame - predictedFrame).^2)));
disp(PSNR);
end
```