```matlab
% Create a List of the image numbers used in each correlation pair.
firstImageNumbers = startImage : frameStep : endImage;
secondImageNumbers = firstImageNumbers + correlationStep;

% Specify the path to the image generation parameters file
% that was saved concurrently with the synthetic images
parametersPath = '/path/to/image/generation/parameters/file.mat';

% Load the image generation parameters file.
imageGenerationParameters = load(parametersPath);

% Extract the vortex parameters
% from the image generation parameters file.
vortexParameters = imageGenerationParameters.VortexParameters;

% Extract the image times from the parameters file.
% Image times in sort-of physical units (i.e. "seconds")
imageTimes = imageGenerationParameters.T;

% Determine the solution times corresponding to the frame numbers
firstImageTimes  = imageTimes(firstImageNumbers);
secondImageTimes = imageTimes(secondImageNumbers);

% Simulated time elapsed between the two images
interFrameTime = secondImageTimes(1) - firstImageTimes(1);

% Set the solution time to halfway between the image times.
% This results in a second-order accurate estimate
% of the Eulerian displacement.
solutionTimes = firstImageTimes + (secondImageTimes - firstImageTimes) / 2;

% This is the number of solution times calculated.
number_of_solution_times = length(solutionTimes);

% These are column vectors corresponding to the
% row (Y) and column (X) coordinates
% of the centers of the PIV interrogation regions
% The variables X and Y are expected to come from
% the external PIV software's solution file.
gridPointsX = X(:);
gridPointsY = Y(:);

% regionWidth and regionHeight are the width and height
% of each interrogation region in pixels. In this example,
% the IRs will be assumed to be 64x64.
regionWidth  = 64;
regionHeight = 64;
```

```matlab
% These are the coordinates of the geometric centroids of each PIV window.
% This corresponds to the physical location in the flow that the PIV
% correlation is supposed to measure. For odd sized windows, the geometric
% centroid of the window is at the center pixel. For even-sized windows, it
% is 0.5 pixels to the right of the pixel located at (regionHeight/2) or
% (regionWidth/2).
xCenter_01 = gridPointsX + 0.5 * (1 - mod(regionWidth,  2));
yCenter_01 = gridPointsY + 0.5 * (1 - mod(regionHeight, 2));

% Vector containing all the x and y grid points.
% This is an input to the vortex velocity function,
% and this format (a single column) is required by ODE45.
gridPointsVector = cat(1, gridPointsX, gridPointsY);

% These lines allocate matrices for the analytical
% velocity fields calculated at each solution time.
%
% Horizontal velocity component
uTrue = zeros([size(X), number_of_solution_times]);

% Vertical velocity component
vTrue = zeros([size(X), number_of_solution_times]);

% Out of plane component of vorticity
rTrue = zeros([size(X), number_of_solution_times]);

% This loops over the different solution times
% and calculates the analytical displacement at each time.
for t = 1 : number_of_solution_times

    % This calculates the true Eulerian velocity field for the t'th field.
    [trueVelocities, trueVorticity] = ...
    lambOseenVortexRingVelocityFunction(solutionTimes(t), ...
    [xCenter_01; yCenter_01], vortexParameters);

    % This is the ground-truth horizontal component
    % of the velocity field expressed as a vector.
    uTrueVect = interFrameTime * trueVelocities(1 : length(trueVelocities) / 2);

    % This is the ground-truth vertical component
    % of the velocity field expressed as a vector.
    vTrueVect = interFrameTime * trueVelocities(length(trueVelocities)/2 + 1 : end);

    % This is the ground-truth out-of-plane component
    % of the vorticity field expressed as a vector.
    rTrueVect = interFrameTime * trueVorticity(:);
    % This reshapes the horizontal component
    % of the ground-truth velocity field into
    % a matrix of the same size as the
    % input coordinates.
    uTrue(:, :, t) = flipud(reshape(uTrueVect, size(X)));

    % This reshapes the vertical component
    % of the ground-truth velocity field into
    % a matrix of the same size as the
    % input coordinates.
    vTrue(:, :, t) = -1 * flipud(reshape(vTrueVect, size(X)));

    % This reshapes the out-of-plane component
    % of the ground-truth vorticity field into
    % a matrix of the same size as the
    % input coordinates.
    rTrue(:, :, t) = flipud(reshape(rTrueVect, size(X)));

end
```