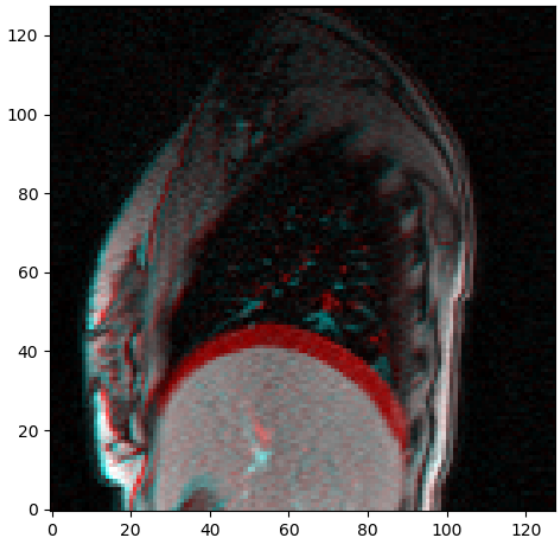
Information Processing for Medical Imaging
MPHY0025 - 2019/2020

Registration exercises 3

Colour overlay of two images

Displaying two images using a colour overlay can be a useful means of visually assessing the similarities and differences between two images. Write a function, `dispColourOverlayImage`, which takes two grey-scale images as input (expected to be in standard orientation) and displays a colour overlay image where the first image appears in red and the second image appears in cyan, so that the colour overlay image appears in grey-scale where the two input images agree and in colour where there is disagreement, e.g.:

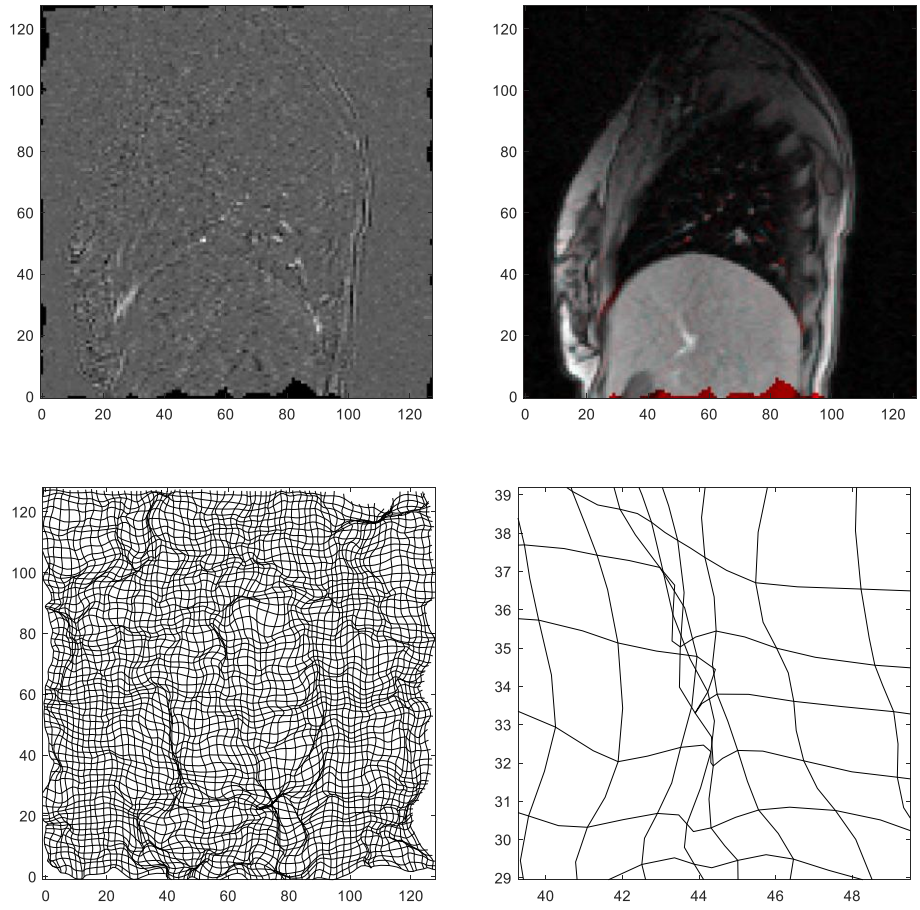


Note – there are several ways to do this, but one of the simplest is to create an array representing an RGB image of size ny * nx * 3 (note – the RGB image must be in matrix orientation so it will be displayed correctly using the MATLAB / matplotlib functions). In an RGB image, there are 3 values at each pixel representing the red, green, and blue components of the colour of the pixel. The values are expected to be scaled between 0 and 1 (unless integer data types are being used), e.g. [0, 0, 0] is black, [1, 1, 1] is white, [1, 0, 0] is red, [0, 1, 1] is cyan, and [1, 0.5, 0.5] is pink. Therefore, the red component of every pixel can be set using the first input image, and the green and blue components set using the second input image (note, the input images must first be scaled to have intensities between 0 and 1, and transposed to account for the differences between matrix and 'standard' orientations). The RGB image can then be displayed using the `image` or `imshow` functions.

Test your function is working by displaying colour overlay images using the `cine_MR_1.png` and `cine_MR_2.png` from last week – the results should appear like the figure above.

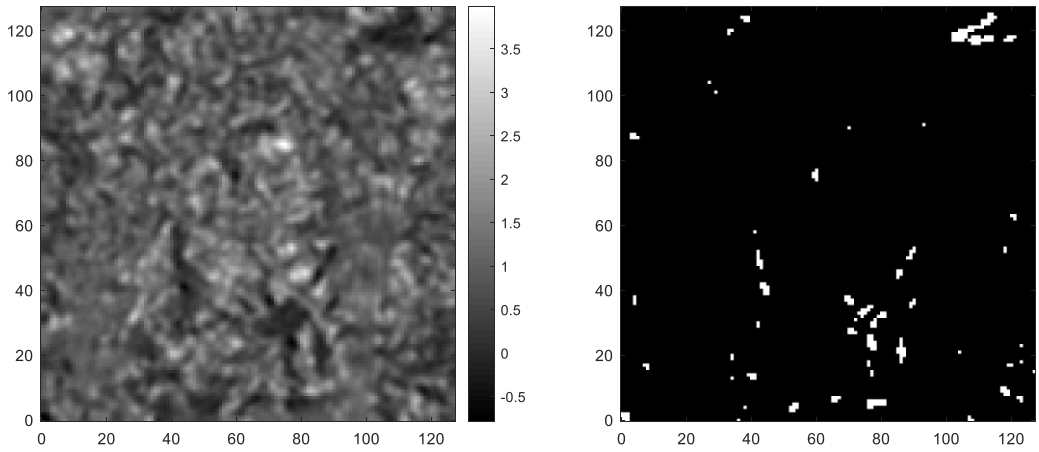
Diffeomorphic Demons

When investigating the effects of the different parameters on the performance of the demons algorithm you should discovered that some sets of parameters can produced transformed images that look very similar to the target image, but the resulting reformation field may not look very plausible. For example, run a registration with `cine_MR_1.png` as target image and `cine_MR_2.png` as source image, and `sigma_elastic = 0.5, sigma_fluid = 1, num_lev = 3`. You should notice that the registration appears to align the images very well – as can be seen by displaying a difference image or a colour overlay image between the target image and transformed images resulting from the registration. However, if you look closely at the deformation field you will see that folding has occurred in some places – this is easiest to notice when displaying the deformation field as a deformed grid and zooming in on the affected regions.



You have also been provided with a function, `calcJacobian`, to calculate the Jacobian determinant (and optionally the full Jacobian matrix) at each pixel in the deformation field, and return an image or map of the Jacobian determinants. Have a look at this function and make sure you understand how it works.

If you use this function to create an image of the Jacobian determinants for the deformation field from the registration above, and then display it (e.g. using the `dispImage` function so that the image is orientated correctly) you will notice that the image is very 'patchy', and does not show smooth expansions/contractions as would be expected during breathing. Furthermore, if you add a colourbar you will see that it contains some very large values (>3.5, indicating the tissue at those pixels has more than tripled in size) as well as negative values, indicating that folding has occurred. Calculate the total number of pixels that have a Jacobian determinant <= 0 and display a binary image indicating where they are located.



Left – Jacobian determinant image, Right – pixels with Jacobian determinant <= 0

We are now going to modify the demons script/function to compose the updates rather than add them, so that the deformation field will always represent a diffeomorphic transformation (as the updates are themselves diffeomorphic), and hence will not contain any folding.
To compose the update, we need to calculate:

$$T_{comp}(x) = T_{current}\left(T_{update}(x)\right)$$

One way to think of, and implement, the composition of two transformations is resampling one transformation using the other transformation. E.g.

$$I_{resamp}(x) = I_{orig}\big(T(x)\big)$$

So, to calculate T_{comp} we need to resample $T_{current}$ using T_{update}.

Note, the update is currently stored as a displacement field, $\mathbf{u}_{disp}$. To resample the image (using the provided `resampImageWithDefField`) we will need to convert this to a deformation field, $\mathbf{u}_{def}$, by adding the pixel coordinates, i.e.

$$T_{update}(x) = \mathbf{u}_{def}(x) = \mathbf{u}_{disp}(x) + x$$

Therefore, to modify the demons script/function you should:

1) Locate the lines of code that initialise the variables containing the update displacement fields, `update_x` and `update_y`. These are lines 91 and 92 in demonsSrcipt.m and lines 52 and 53 in demons.py (note – I suggest downloading the latest version of these files provided with exercises 3, as I made some minor edits to the versions provided last time and the line numbers correspond to the latest versions). Add a new line which initialises a new variable, `update_def_field`, as an array of 0s which is the same size as `def_field`.
2) Delete (or comment out) the lines of code which add the update. These are lines 187 and 188 in demonsScript.m (although may now be 188 and 189 as you've added a line!) and lines 136 and 137 in demons.py. Replace these with lines of code that:
3) Calculate the update deformation field, `update_def_field`, from the displacements stored in the update variables, `update_x` and `update_y`, by adding the pixel coordinates,

stored in `X` and `Y`. (if you are not sure how to do this look a few lines down, after the elastic like regularisation, to where `def_field` is calculated from `disp_field_x` and `disp_field_y` in a similar way.)

4) Calculate the composed deformation field by resampling the current deformation field with the update deformation field using the `resampImageWithDefField` function. The composed transformation can be stored directly in `def_field`, there is no need to create a new variable. The x and y components of the deformation field must be resampled separately as the `resampImageWithDefField` function can only work with 2D images. The x component is `def_field(:,:,1)` or `def_field[:,:,0]`, the y component is `def_field(:,:,2)` or `def_field[:,:,1]`.
5) Calculate the displacement field variables, `disp_field_x` and `disp_field_y`, from the composed deformation field by subtracting the pixel locations stored in `X` and `Y`. This is necessary as the displacement field, rather than the deformation field, is smoothed when applying elastic regularisation.
6) When the deformation field is resampled pixels outside the original deformation field are given a value of NaN. The displacement field variables will also have NaNs in the corresponding locations, and these need to be replaced with values of 0 (note, they do not need replacing in the deformation field, as the deformation field values are recalculated from the displacement field after the elastic regularisation is applied).
7) Now save your modified script/function, e.g. as diffeoDemonsScript.m or diffeoDemons.py

If you have followed the steps above correctly when you run your new script/function (with the same parameters as used above for the non-diffeomorphic demons) you will see that the registration again appears to align the images very well, but this time the deformation field does not contain any folding. Confirm this by checking that none of the values of the Jacobian determinant are less than or equal to 0.
While the deformation field does not contain any folding, it is still not very smooth, as can be seen from the deformed grid or by displaying the map of the Jacobian determinants. Try modifying the registration parameters to see if you can obtain a result which appears to align the images well but also gives a smoother and more physically plausible looking deformation field.
Now load the other image from last week, `cine_MR_3.png`, and see if you can find parameters that give good results when registering image 1 and image 3 using the diffeomorphic demons (remember to covert the image to double and re-orientate it before running the registration).

Once you think you think you have fully explored the possibilities of registering lung MR images (or just want to try something more fun / useful) you can try registering your own images, either from your research or from elsewhere (e.g. cats and dogs, photos of you and/or your friends) – but remember, the demons algorithm is a purely intensity based algorithm which tries to match pixels with the same intensity in the two images, so will not align corresponding features unless they have similar intensities. It also only works for grey scale images and requires the images to be the same size and shape.