Registration Grand Challenge

CIS 537 / BE 537

February 17, 2015

This assignment is due via Canvas on March 23 at 3pm.

1 Overview

This grand challenge assignment involves performing groupwise affine registration between a set of brain images. The input to this assignment is a set of brain images, and the output is a set of affine transformations into a common reference space. The quality of a set of registrations will be measured in terms of how closely structures labeled in the input images are lined up by the transformations.

2 Data

- The data consist of a set of anonymized, skull-stripped MRI images of the brain, named **sub001_mri.nii** to **sub040_mri.nii**.
- All images have been resampled to have voxel size of $2.4 \times 2.0 \times 2.0\text{mm}^3$and resolution of $80 \times 96 \times 80$ voxels.
- The intensity range of all images has been normalized to range $[0, 1024]$.
- Images are divided into a training subset and test subset, with 20 images each.
- For the images in the training dataset, manual segmentations of the left and right hippocampus are included. These segmentations are stored in files **sub*_seg.nii**.

3 Requirements

The following are the requirements for the assignment

1. Implement the basic component described below. This is required for all groups.
2. Implement two or more enhancements suggested below, or think of your own ways to improve the basic algorithm. Remember, that your goal is to come up with the best possible registration.
3. Evaluate your approach, and the enhancements that you have made, on training data
4. Organize your final groupwise registration results for the test dataset in a consistent fashion
5. Write a short report describing your approach and your results.

3.1 The Basic Component

This basic component must be completed by all groups.

3.1.1 Extend myView to Display Registration Results

To help visualize registration results, write a function with the following signature:

$$\texttt{myViewAffineReg}(\texttt{fixed}, \texttt{moving}, \texttt{spacing}, \texttt{A}, \texttt{b})$$

where **fixed** and **moving** are images, spacing is a 3×1 vector and **A** and **b** is an affine transform that maps a moving image into the fixed image. The function should visualize the alignment between the fixed image and the transformed moving image. There are multiple ways to visualize registration results, such as using partial transparency, checkerboard pattern, or the approach we used in class, which is to overlay a segmentation of one image over the other (e.g., using k-means segmentation and the **contour** function, see the code from the Optical Flow lecture). Pick whatever works best for you. Use this function to visualize your registration results below.

3.1.2 Write the Objective Function for 3D Affine Registration

Write a MATLAB function that evaluates, and computes the gradient, of the following objective function:

$$E(A, \mathbf{b}) = \int_{\Omega} \left[I(\mathbf{x}) - J(A\mathbf{x} + \mathbf{b}) \right]^2 \mathrm{d}\mathbf{x} \,,$$

where $\Omega \in \mathbb{R}^3$ denotes the image domain, I is the fixed image, and J is the moving image, A is a 3×3 matrix and $\mathbf{b}$ is a 3×1 translation vector. When implementing this function, use as the model the function **reg1D_affine_objective** that was examined in class. Of course, the main difference is that your function will need to compute the objective for 3D registration. Your function will have the following signature:

$$[\texttt{E}, \texttt{g}] = \texttt{myAffineObjective3D}(\texttt{p}, \texttt{I}, \texttt{J})$$

where `E` is the value of the objective function, `g` is the gradient of the objective function (∇E), and `p` is a 12 x 1 parameter vector that consists of the elements of A and $\mathbf{b}$, i.e.,

$$A = \begin{bmatrix} p_1 & p_4 & p_7 \\ p_2 & p_5 & p_8 \\ p_3 & p_6 & p_9 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} p_{10} \\ p_{11} \\ p_{12} \end{bmatrix}.$$

The gradient `g` is a 12 x 1 array whose j-th element is equal to the partial derivative

$$\frac{\partial E}{\partial p_j}.$$

The gradient should be computed analytically, i.e., by writing down the formula for the the partial derivatives of E , and using MATLAB to compute that formula.

3.1.3 Test the Correctness of Gradient Computation

To test whether your function computes the gradient correctly, compare the "analytical" gradient returned by your function to a "numerical" gradient approximated using the central finite difference approximation.

$$\left.\frac{\partial E}{\partial p_j}\right|_{\mathbf{p}} \simeq \frac{E(\mathbf{p}+\epsilon\mathbf{e}_j) - E(\mathbf{p}-\epsilon\mathbf{e}_j)}{2\epsilon},$$

where ϵ is a small number (e.g., 0.0001) and $\mathbf{e}_j$ is the j-th unit coordinate vector and $\mathbf{p}$ is the parameter vector. In MATLAB, compute the numerical and analytical gradient of E, with image **sub001** used as the fixed image, image **sub002** used as the moving image, and $\mathbf{p} = [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]^T$, i.e., corresponding to the identity affine transform. Apply a small amount of Gaussian smoothing ($\sigma = 1.0$) to the fixed and moving images before doing this experiment. Smoothing helps avoid numerical errors due to the discontinuity of the image function. Use the function you wrote as part of your first homework assignment to perform Gaussian smoothing.

The numerical approximation should be close to the analytical gradient, i.e., the relative error should be less than 1%. Report your results in your write-up.

3.1.4 Test Your Objective Function by Registering Two Images

Pick a pair of images as the fixed image and the moving image. Note that at this stage, registration may not work very well for the pair of images that you picked, particularly if their intensity ranges are different. Try to find a pair of images that have roughly similar histograms. You can use ITK-SNAP for this purpose.

Apply some smoothing to the two images, as in the section above. Also, as you are testing, you might find it helpful to subsample the images by a factor of 2 in each dimension, as this will speed up computations considerably. When subsampling images, you should apply enough smoothing to prevent substantial aliasing (remember the low-pass filtering material from class).

Minimize your objective function using the MATLAB command **fminunc**. Look at the code I showed in class for 1D registration for an example of how to use this function. Make sure to use the command **optimset** to enable the use of the analytical gradient (**GradObj** parameter), and to set the number of iterations (**MaxIter** parameter). You can also use the **OutputFcn** parameter to **optimset** to display the intermediate results of your registration using **myViewAffineReg**.

Include in your report the results of this registration. Visualize the results before and after running the registration with your **myViewAffineReg** function.

3.1.5 Write a Function for Multi-Resolution Affine Registration

Write a function that executes affine registration between two images using the multi-resolution framework. By multi-resolution, I mean to perform registration between a pair of images in k iterations, where at the first iteration, the images are subsampled by the factor 2^{k-1} in each dimension, at the second iteration the images are subsampled by the factor of 2^{k-2}, and so on until the k-th iteration, when the images are registered at their full resolution. The results of each iteration are used to initialize the registration for the next iteration. When subsampling the image by the factor 2^{k-j} at the j-th iteration, you should first low-pass filter the fixed and moving images with a Gaussian with the standard deviation $2^{k-j}\sigma$, in order to avoid aliasing.

Your function should have the following signature:

$$[\texttt{A}, \texttt{b}] = \texttt{myAffineRegMultiRes3D}(\texttt{I}, \texttt{J}, \texttt{iter}, \texttt{sigma})$$

where `iter` specifies the maximum number of optimization iterations (**MaxIter** parameter in **optimset**) to perform at each iteration of multi-resolution registration. For example, $\texttt{iter} = [20, 10, 5]$ means do 20 optimization iterations at level 1 (where the images are subsampled by the factor of 4); 10 iterations of optimization at level 2, and 5 iterations at level 3 (full-resolution images). The parameter `sigma` refers to the Gaussian smoothing factor discussed in the previous paragraph.

Your function should interpret $\texttt{iter} = [20, 10, 0]$ as "do not run optimization at the full resolution level." You will probably find that optimization at full resolution is very slow, at least until we make some efficiency improvements in Section 4.

Include in your report the results of multi-resolution registration. Visualize the results before and after running the registration with your **myViewAffineReg** function. Describe the parameters you used and give the resulting affine transformation.

Important Note! The transformations $[A, \mathbf{b}]$ that you obtain by minimizing your objective function are not invariant to the size of the images. If you computed $[A, \mathbf{b}]$ as the optimal affine transformation between the fixed and moving images at resolution level j, you will find that applying the same transformation to the images at level $j + 1$ gives wrong results! This has to do with the fact that the transformations are applied to voxel coordinates, which change by the factor of 2 at each iteration. This means that after running the registration at level j, you will need to modify the resulting affine transformation before using it to initialize the registration at level $j + 1$. It's a simple modification, but it is part of your assignment to work it out.

3.1.6 Incorporate Speedups to your Objective Function

There are some simple ways to reduce the computational time spent evaluating the objective function. This is not a required part of the assignment, but rather a suggestion to make things run faster.

First, there are some terms (like the gradient of the moving image) that can be precomputed ahead of time and reused on subsequent calls to **myAffineObjective3D**. Precompute these terms, store them in a MATLAB **struct** object, and pass them in as an additional parameter to **myAffineObjective3D**.

Second, most of the time in computing the objective function is probably being spent executing **interpn**. I have provided a replacement function for **interpn** that allows you to interpolate multiple image volumes at the same set of coordinate locations more efficiently than by calling **interpn** multiple times. Instead of calling

$$\begin{gathered}\texttt{J1} = \texttt{interpn}(\texttt{I1}, \texttt{X}, \texttt{Y}, \texttt{Z}, '*\texttt{linear}', 0); \\ \ldots \\ \texttt{Jn} = \texttt{interpn}(\texttt{In}, \texttt{X}, \texttt{Y}, \texttt{Z}, '*\texttt{linear}', 0);\end{gathered}$$

to interpolate n images of the same size, you can use my functions as follows. First call

$$\texttt{data} = \texttt{my\_interp3\_precompute}(\texttt{size}(\texttt{I1}), \texttt{X}, \texttt{Y}, \texttt{Z});$$

to precompute terms used for repeated interpolation. Then to interpolate multiple image volumes, call

$$\begin{gathered}\texttt{J1} = \texttt{my\_interp3}(\texttt{I1}, \texttt{data}); \\ \ldots \\ \texttt{Jn} = \texttt{my\_interp3}(\texttt{In}, \texttt{data});\end{gathered}$$

You can use the MATLAB commands **tic** and **toc** to test how much of a speedup you got.

3.1.7 Generate a 'Baseline' Groupwise Registration

In this section, you will perform groupwise registration between all of the training images by using a single image as the 'template' and registering the rest of the images to it. This is the most naive approach to groupwise image registration, and it is unlikely to yield very good accuracy. But it should serve as the baseline result, in the sense that your subsequent attempts at groupwise registration should improve on this result.

Pick one of the test images as the template, and apply multi-resolution affine registration to all the training images (i.e., treat the template as the 'fixed' image, the training images as 'moving'). Store your A matrices and $\mathbf{b}$ vectors in a pair of MATLAB cell arrays. Here is an example of how to create a cell array:

```
% Create a 20 x 1 cell array
A_cell = cell(20,1);
b_cell = cell(20,1);

% Populate cell array
for i = 1:20
    % Replace '...' with your code
    [A_cell{i} b_cell{i}] = myAffineRegMultiRes3D(...)
end
```

To evaluate the performance of this approach, use a function that is provided as part of the assignment. Its signature is

$$\texttt{evaluateOnTrainingSet}(\texttt{path}, \texttt{A\_cell}, \texttt{b\_cell})$$

where **path** is the path to the 'train' directory that is provided in the assignment, and the other inputs are described above. There must be exactly 20 elements in each cell array, and their order should correspond to the order of the images, (i.e., `A_cell{12}` must be the affine transform matrix for the registration where the template is the fixed image, and image **sim012_mri** is the moving image). This function will measure how well manual segmentations of the left and right hippocampus are aligned by your affine registration. It does so by computing volumetric overlap between every pair of images (190 pairs). To avoid bias due to the size of the template, overlap is computed in the native space of each image. The function will return the average overlap, and it will also plot the histogram of the overlaps over all pairs of images. The larger the overlap, the better.

If you perform no registration (pass in the identity transform in the elements of `A_cell` and `b_cell`), you will get the average overlap of 0.14. Hopefully, after running your registration code, the value of this average pairwise overlap should increase. However, since we have so far used a 'dumb' strategy for groupwise registration (registration to a single fixed template), we can expect that we will be able to greatly improve on this result.

Include in your report the result of running **evaluateOnTrainingSet** on the transformations computed in this step.

3.2 Extensions to the Base Component

To win the grand challenge, you will certainly have to improve on the base component. Here are some suggested approaches, and you can think of your own improvements too. If you do, and are unsure if the improvement will be considered significant enough, check with the TA.

3.2.1 Histogram Matching

Some of your segmentations are probably failing. This is most likely due to the objective function for the registration being the mean squared intensity difference. When the two images you are registering have different intensity distributions (say the gray matter in one image is in the range 200-300 and in the other image it is in the range 400-500), this metric becomes very ineffective (be sure you understand why). One approach is to change the objective function, for example to use mutual information. Another is to try to *match* the intensity histograms of the images before registration. Histogram matching involves applying a monotonic transformation η to the intensities of one of the images, so as to make its histogram as similar as possible to the other image.

A good description of the standard approach to histogram matching is given in Sections 2-3 of the article

- J. P. Rolland, V. Vo, B. Bloss and C. K. Abbey, "Fast algorithms for histogram matching: Application to texture synthesis", *J. Electron. Imaging* 9, 39 (2000); http://dx.doi.org/10.1117/1.482725

Hint: as with other tasks, the MATLAB **interp1** function can be useful for implementing histogram matching (for computing an inverse of a monotonically increasing function).

Note: it is not ok to implement this extension using the MATLAB **histeq** command.

3.2.2 Registration to an Unbiased Population-Derived Template

Joshi et al. revolutionized the idea of unbiased templates, which are derived through iterative application of averaging and registration to the average. Although their method uses non-linear registration, it can also be applied to affine groupwise registration. To implement this method, perform the following iterative algorithm

1. Compute an initial average A_0 of the input images by averaging the intensities. Let $j = 0$.
2. Register each image to the current average A_j.
3. Compute the new average A_{j+1} by averaging the intensities of transformed images from step 2.
4. Repeat steps 2-3 until the average converges (or just for 4-5 iterations).

Implement this algorithm on the training data, and observe how the average overlap changes as you iterate.

3.2.3 Defining a Mask

In this assignment, we are interested in aligning the hippocampi well. Doing registration using the whole brain is not necessarily optimal for aligning hippocampi. One way to address this is to introduce a mask into the energy term. Let M be a function defined in the image domain, such that $M = 1$ in the parts of the fixed image that are "of interest," (i.e., around the hippocampi) and $M = 0$ elsewhere. Then, you can redefine the energy term as

$$E(A, \mathbf{b}) = \int_{\Omega} M(\mathbf{x}) \cdot [I(\mathbf{x}) - J(A\mathbf{x} + b)]^2 \, \mathrm{d}\mathbf{x} \,,$$

which will restrict the intensity similarity computation just to the places inside the mask. You can also speed up the computation of the energy function considerably by only computing $J(A\mathbf{x}+b)$ at points where $M \neq 0$. This means calling **interpn** with fewer points, and thus a considerable computational speedup. Same goes for the gradient computation.

How to derive the mask? For the training images, you have manual segmentations given, so you can use the region around the manual segmentations to define a mask. For instance, if you use the unbiased population approach above, you can derive a mask in the template space by averaging masks computed for the input images. Then you can reuse that mask when registering the test images to the template.

3.2.4 Speeding Up Registration Using Analytic Hessian Computation

While making your code more computationally efficient will not directly affect registration quality, it will allow you to run more iterations at higher resolution, which might help. One way you can speed up your code (and make optimization more robust too) is by helping the MATLAB optimizer by computing the second derivatives of the objective function analytically. The command **fminunc** uses a second-order optimization scheme, which can converge faster than the simple first-order gradient descent methods we studied in class.

- In addition to computing the gradient of the objective function in **myAffineObjective3D**, also compute the Hessian of the objective function, and return it as the third output. The Hessian is a matrix of the second derivatives (i.e., the element (i, j) in the Hessian is given by

$$H_{i,j}(\mathbf{p}) = \frac{\partial^2 E}{\partial p_i \partial p_j}$$

- As you did above for the analytical gradient computation, you can test the accuracy of your Hessian computation by comparing the analytically derived Hessian to the numeric approximation of the Hessian. The numeric approximation of the j-th column of the Hessian is given by

$$H_{*,j}(\mathbf{p}) \simeq \frac{\nabla E(\mathbf{p} + \epsilon \mathbf{e}_j) - \nabla E(\mathbf{p} - \epsilon \mathbf{e}_j)}{2\epsilon}.$$

- To use the analytic Hessian in your optimization, pass the parameter **Hessian** with value **on** to **optimset**. Otherwise, MATLAB automatically uses the numerical approximation, which requires $2 \cdot 12$ calls to the objective function to compute the Hessian.

3.2.5 True Groupwise Registration

This is the most interesting extension, and also probably the most challenging one. The idea is to compute pairwise registration between all pairs of images in your data, perhaps at low resolution, so that you can do this in a reasonable amount of time. We also compute the similarity between each pair of images after registration, which would give us an idea of how good the registration was for that pair. This allows us to construct a graph in which all images are vertices, and edges represent registration, with the weight of each edge given by the similarity. Given such a graph structure, we can try to bring all the images into alignment by "bypassing" poor registrations. There are endless possibilities for how to exploit this graph structure, but here is one idea:

1. Find a subset of images that align very well to each other. Let's call it the *core* subset.
2. Derive a population-specific template for just this core subset of images.

3. For each of the remaining images find the "shortest path" in the graph to the core subset. This shortest path would consist of or more edges in the graph connecting the out-of-core image to one of the core images. This edge or edges correspond to an affine transformation or a sequence of affine transformations (the latter can be composed into a single affine transformation). Take this transformation, compose it with the transformation from the in-core image to the template, and use it to initialize the registration between the out-of-core image and the template.

This is just one of a myriad ideas that are available. Try to be creative and think of your own.

3.3 Evaluate Your Improvements

For each improvement that you propose to the basic approach, evaluate it using the function **evaluateOnTrainingSet**. Your report should include a table with descriptions of experiments that you ran in one column, and the average overlap reported by **evaluateOnTrainingSet** in the other.

3.4 Submitting Your Data to the Competition

This is the most important part! Take whatever groupwise registration approach that worked best for you on the training data and apply it to both the training dataset and the test dataset. *Use the same set of parameters (**iter, sigma, etc)** for all images!* Store the resulting transformations in four cell arrays of size 20x1:

$$\texttt{A\_train}, \texttt{b\_train}, \texttt{A\_test}, \texttt{b\_test}$$

Then save these arrays to a file using the MATLAB command

$$\texttt{save}('\texttt{myteamname\_submission.mat}','\texttt{A\_train}','\texttt{b\_train}','\texttt{A\_test}','\texttt{b\_test}');$$

Submit this file as part of your assignment on Blackboard. This will be used as your entry for the grand challenge competition.

3.5 What to Submit

1. The **.mat** file.
2. All of your MATLAB source code. Make sure that it is documented, so one can follow what you are doing.
3. A short report including
   (a) State to what extent you were able to complete the base component, e.g., "The base component was fully and successfully implemented."
      i. Include the items marked by underlined text in Section 3.1.
   (b) Description of the improvements you have made to the base component:
      i. Make sure to reference your MATLAB code entry points for the improvements you describe.
      ii. You can also mention improvements that did not work.
      iii. Indicate which set of improvements and what values of parameters are used for the submission you made to the grand challenge.
   (c) Table of experiments performed on the training data and corresponding average overlaps.
      i. Indicate (in bold face) the experiment that is being submitted for the grand challenge.

4 How the Assignment Will Be Graded

The assignment will be graded on the following criteria:

1. Ability to complete the base component
2. The complexity and number of improvements attempted
   (a) As a guideline, you should invest as much or more effort into the improvements as in the base component
3. The success of the improvements
4. The quality of the report and submitted code

Your performance on the grand challenge (i.e. your team's rank) is not a factor in grading the assignment. It can, however, impact your overall grade in the course (see the syllabus).

5 Working as a Team

Please follow these guidelines:

- Set an internal timeline and milestones. You have about a month to complete the assignment. You should have the base component finished two weeks into the assignment, otherwise you likely will not be able to complete the extensions. Pick a team leader who will keep track of the milestones and push the rest of the team to meet them.
- Everyone should do their fair share. If a team member is not, try to address that within the team promptly (this is why it's important to have internal deadlines). If that does not help, let the instructor know promptly. Alerting the instructor at the last moment is a bad idea because there will be little opportunity to resolve the issue.
- If an issue of unfair division of effort within a group is raised, students will be asked to provide a self-assessment describing how much they and their teammates contributed to the project.