Advanced brain imaging methods: Practical 1

3rd February 2015

These practicals are based on SPM 12, so make sure that you have downloaded the software from canvas or there, or there.

Once downloaded and unzipped, start matlab and go in the spm directory you have just unzipped. Typing `addpath(pwd)` to the matlab prompt should make you able to run spm 12 for this matlab session. You can check if that worked by just typing `spm` to launch SPM in matlab.

1 Preliminary remarks

MRI images are composed of 2 parts:

1. a header that contains the information about the image (transformation matrix, subject information, scanner and sequences parameters and other types of meta data),
2. an image that can be a 3D matrix that contains one volume (an anatomical scan for example) or a 4D matrix for a time series of volume (a series of EPI volumes acquired during one fMRI session).

There are quite a few different formats of images. Most imaging softwares can easily deal with images in an Analyze format where header and image are stored in 2 different files (with a .hdr and a .img extension respectively). But the field tends more and more to use the Nifti format where header and image can be stored in one single .nii file. Scanners might often output images under a DICOM format that needs to be converted.

2 Pre-processing: button clicking.

First we are going to run the pre-processing of the single subject block auditory fMRI activation data set from the SPM website, so download the file MoAEpilot.zip from there or from canvas and unzip it. The folder 'fM00223' contains the functional images and the 'sM00223' the structural one.

Follow the instruction in the file SPM_Preprocessing.pdf (on canvas) taken from the SPM manual to run the pre-processing. There is also a flow chart on canvas that can guide through the different steps of the preprocessing.

Here after are some details that are not mentionned in the manual.

SPM can make some changes to images without creating a new ones: the changes are 'stored' in the header. When SPM rewrites a new image after having performed some transformation, it adds a prefix (usually a single letter) to the name of that image.

2.1 Changing the origin

Usually before starting the pre-processing on some images it is better to set their origin to the anterior commmissure. An easy way to change the origin: use the **Check Reg** button to open the image of interest and position the crosshair where you want the new origin to be, right-click on the image, select *Reorient image(s)* and then select *Set origin to Xhairs*. If you have no idea where the anterior commissure (AC) is check here. You can then select additional images you want to to apply this change of origin to.

So before starting the pre-processing of your images set the origin of the structural image to the AC. Once you have done that set the origin of all the funtional images to the AC.

2.2 Realign

This will realign all the selected images and create a new set of images with the prefix 'r'. For this part you can also use *Realign and Unwarp*) to correct for the distorsion in the image due to field inhomogeneity-by-movement interactions: in this case SPM will add the prefix 'u' to the images.

3 Pre-processing: command line

3.1 Reading header information

SPM uses the function `ImgHeader=spm_vol('ImageName.img')` to read information contained in the header of an image and store in a structure in the variable `ImgHeader`.

A matlab structure has several fields (usually with pretty transparent names), each associated with a value. For example, the value associated to the field 'dim' can be read with `ImgHeader.dim` contains the dimension of the image associated to this header. You can also directly get the value transformation matrix by using the function `spm_get_space(ImageName)`.

Using either of these methods, get the header information of the non-normalized structural scan from the data set you downloaded from the SPM website. From this header can you tell:

1. How many axial slices do you have in this image? What are the dimensions (the number of voxels) of each slice?
2. What is the transformation matrix of this image (i.e what are its values in each row and column)?

From this matrix:

1. How can you tell what are the dimensions of each voxel (in mm) of this structural scan?

2. How can you tell the real world coordinates (in mm) of the voxel with the subscripts [1,1,1]?
3. What matlab code would you use to generalize this to any coordinates in mm (i.e get the real world coordinates of a given [x,y,z] voxel)? This will require to multiply 2 matrices (check the slides of the pre-processing lecture to know which one).
4. How could you tell the [x,y,z] subscripts of the voxel that contains the origin of the image, i.e with world space coordinates of [0,0,0] mm?
5. What matlab code would you use to generalize this to any voxel (i.e get the voxel space coordinates a point with some given world space coordinates)?

Hints:

- Remember that if a matrix A perfoms a certain transformation, you can find the matrix that performs the 'reverse'transformation, by finding the inverse of A (check the Maths4SPM.pdf in additional material for a reminder of what the inverse of a matrix is).
- For those questions, you can easily check you answers or your code by using the **Display** button of the graphic interface of SPM. You can also use the **Check Reg** button and right-clicking on the image to get some of those informations.
- Questions 3 and 5 will require you to do some matrix multiplication, so remember that there are restrictions (in terms of dimension) on how to do those multiplications: you might need to add dummy columns or rows to do some of those multiplications.

Matlab tips:

- If you want help on how to use a certain function, just type in the command line `help` followed by the function name.
- To create a row vector `VoxCoord` containing, for example, the voxel coordinates $(125, 156, 25)$, you should type `VoxCoord=[125,156,25]` . If you wanted to have the same but as a column vector, you would type `VoxCoord=[125;156;25]` .
- To add values to a row vector, you could do `VoxCoord=[VoxCoord,25]` and similarly for a column vector `VoxCoord=[VoxCoord;25]` .
- You can transform a row vector into a column vector, by taking its transpose: `VoxCoord=trans(VoxCoord)` or more simply `VoxCoord=VoxCoord'` .
- The command `inv(A)` will give you the inverse of the matrix A.

3.2 Reading values from an image

This code will give you the content of an image and store in a variable `ImgContent`.

```
ImgHeader=spm_vol('ImageName.img')
ImgContent=spm_read_vols(ImgHeader)
```

Matlab tips:

- If you want to suppress the output to the screen (i.e make sure that matlab will not display the result of a certain command or calculation) add an `;` at the end of the line,
- If you want to stop an on-going action matlab is doing (displaying some results or doing some computation), press ctrl+C.

If you wanted to know the intensity in the voxel with coordinates $(125, 156, 25)$ you could do either this `ImgContent(125,156,25)` or that:

```
VoxCoord=[125 156 25]
ImgContent(VoxCoord(1),VoxCoord(2),VoxCoord(3))
```

Similarly if you wanted to have all the values of the 25th slice in the z direction (axial), you would type `ImgContent(:,:,25)` .

On the first EPI image, find a voxel that is roughly in the left auditory cortex (by using **Display** or **Check Reg**) and using the information above give the signal intensity found in this voxel.

3.3 Reading a time-series

Now try to plot (using the function `plot`) the time-series of signal intensities of that voxel across the all the realigned EPI volumes: i.e for each EPI acquired in this experiment, what is the BOLD signal value in this voxel?

To do the rest of this task, you can either use a for loop to open each image, get the value at the desired coordinates and then open the next image, like this:

```
for ImageIndex = 1:TotalNumberOfImages
  % Get image header
  ImgHeader=spm_vol(ImageList(ImageIndex));

  % Get image content
  ImgContent=spm_read_vols(ImgHeader);

  % Get signal value at coordinate i,j,k
  SignalValue=ImgContent(i,j,k);

  % Add it at the right position in the time series of values
  TimeSeries(ImageIndex)=SignalValue;
end
```

Alternatively, you can use the fact that `spm_vol` can accept a matrix of filenames (see below) and open all the files at once in one 4D matrix:

```
ImgsMat = ['file_name_1.img';
  'file_name_2.img';
  'file_name_3.img'];

ImgHeaders = spm_vol(imgsMat);

ImgContents=spm_read_vols(ImgHeader);
```

Hopefully SPM also has a function that can automatically get data at a specific voxel or set of voxels in a 3D or 4D volume : `spm_get_data(V,XYZ)` where V is the list of files you want to open and XYZ is the list of the coordinates of the voxels you are interested in.

Hints:

- To get the list of images in a folder, you can either use `ImageList=ls('*.img')` or `ImageList=dir('*.img')` to store this list in a structure in the variable `ImageList` . The name of each file will be store in the field 'name' of that structure, that you can read like this for the first image: `ImageList(1).name` .
- An easier, but less scriptable, way to list all the images you want to open is to use the following spm function `ImageList=spm_select(Inf,'image')` to have a matrix with all the absolute images names.

3.4 Estimating misalignment using different cost functions

As we have seen in class, estimating how misaligned two images are is done using cost functions.

The cost function typically used by SPM during realignment is the least-squares cost function:

$$C = \sum_{v=1}^{N} (A_v - B_v)^2$$

This function computes the sum, from the voxel (v) 1 to N, of the square of the difference in intensities between the image A and B.

FSL uses the normalized correlation for its realignment:

$$C = \frac{\sum_{v=1}^{N}(A_v B_v)}{\sqrt{\sum_{v=1}^{N} A_v^2}\sqrt{\sum_{v=1}^{N} B_v^2}}$$

As the misalignment between the 2 images diminishes, which value should each of these cost functions tend towards?

Get the data from the first ***raw*** EPI image and store it in a variable called `Volume_1`. Now create the following variable :

- `Volume_1_offset` where all the voxel intensities of `Volume_1` will be offseted by +10% of the mean value of this volume,
- `Volume_1_noise` where you will have added Gaussian random noise to `Volume_1` ,

- `Volume_1_invert` where voxels with a high intensity in `Volume_1` will now have a low intensity and vice versa.

Estimate the misalignment between `Volume_1` with itself and with each of the 3 other images using the normalized correlation and the least-squares cost functions. Which comparison illustrates best the fact that these cost functions are not appropriate for inter-modality coregistration?

Matlab tips:

- you can add Gaussian random noise to an image of dimension (X, Y, Z) by adding to your original image, an image of Gaussian noise of same dimension created with the command `Noise=randn(X,Y,Z,)*sigma+mu` where `sigma` and `mu` are the variables that contain the values of standart deviation and mean value you want to your Gaussian noise to have. For this exercice you can try different values for both parameters.

- if you want to perform some operation on the element of a matrices and not on the matrices themselves, you have to use a `.` in front of the operation sign you are perfoming. For example, say you want to multiply the value of each voxel of the image A with its corresponding value of the image B, you should then type `A.*B` because the operation `A*B` would multiply the whole matrice A by the matrice B.

- Usual mathematical functions in matlab that you might need: `sum`, `mean`, `sqrt`, `^`.

3.5 Generating coregistration histograms

At the end of a coregistration, SPM displays a mutual coregistration histogram also sometimes called joint histogram. Intermodality coregistration uses methods that try to minimize the entropy (the disorder) on these histograms: it is sometimes quite obvious when one compare the before and after histograms.

For two images of same dimensions, you can make a similar looking histogram by creating a scatter plot in matlab using the `scatter` function with as many dots in the scatter plot as there are voxels in the image and where the x coordinate of the i^{th} dot is given by intensity in the i^{th} voxel of the first image and the y coordinate of the $i^(th)$ dot is given by intensity in the i^{th} voxel of the second image:

```
scatter(Image_A(:), Image_B(:), '.');
```

What should the histogram look like when the two images are the same? And when one image is the 'negative' of the other?

Check this by plotting the histograms that compares the `Volume_1`with each of the 3 other images of the precedent question. Why is it better to have to rely on those histograms to evaluate the coregsitration of two images from different modality ?

SPM tips:

- The SPM function that does the coregistration of 2 images is `spm_coreg.m` and it relies on the function `spm_hist2`to generate coregistration histograms.

3.6 From native space to MNI space and back

The file Masks.zip from canvas contains 2 mask images: unzip it. The file Left_A1.nii and Right_A1.nii contain the masks for the left and right auditory cortices in MNI space. Voxels in the masks have a value of 1 when this voxel is within the region of interest (ROI) and a value of 0 when it is outside of this ROI. Use the **Check Reg** button to open the non-normalized structural scan and the two masks. Since the masks are in MNI space and the structural scan is still in native space (in the coordinate referential of the scanner), we need to transform those masks from MNI space to native space. To do this you need to use the deformation field created by the segmentation step of the pre-processing.

To transform the masks from MNI space to native space do the following

1. Click on the **Batch** button,
2. In the window that just opened delete any module that might be there: right click and then select *delete module*.
3. Go to the menu: *SPM* → *Util* → *Deformations*.
4. Set *Composition* to *Deformation Field* and then select the deformation field *y_sM00223_002.nii*.
5. Set *Output* to *Pullback*: this will apply the inverse transformation defined by the deformation field (i.e go from MNI space to native space)
6. In *Apply to* select the 2 mask images *Left_A1.img* and *Right_A1.img*.
7. In *Output destination*, select the directory where the mask images are.
8. Press *Run*

This will create 2 new images with the prefix 'w' that are now in native space.

You will then need to reslice the masks to have the same dimensions as the EPI images. To do this, select *Coregister (Reslice)*). Directly reslicing a given image to the same dimension of another one assumes that those 2 images are already coregistered, but if you have followed the SPM tutorial so far (i.e coregistered the structural to the mean EPI image) you should be fine. To reslice the masks, select the mean EPI image as the image defining space and select as images to reslice the two masks brought to native space (i.e that start with a 'w' prefix). Since you want your final images to only contain 1 or 0 values, select nearest-neighbour interpolation. After that click 'RUN' and SPM should quickly create 2 new images with the prefix 'r'.

3.7 Reading values within a region of interest

With those new masks, you are now going to create a figure showing how the average BOLD signal in each auditory cortices accross evolves accross the time series of realigned EPI images. You can now modify the code you created for the section 3.3 so that for every volume of the time series, you compute the mean of the values that are within the ROI defined by the mask with the following code `mean(Volume_Data(Mask_Data))`.

3.8 Applying affine transformations to an image

As its name suggests, the function `spm_write_vol(ImgHeader,ImgContent)` will create an image given some data and header information. For example, the following code will open an image and create a copy of it:

```
% Get the header
ImgHeader = spm_vol('filename.nii');
% Get the data associated
Data = spm_read_vols(ImgHeader);

% Creates a copy fo the header
NewImgHeader = imgInfo;

% Change the name in the header
NewImgHeader.fname = 'copy_of_filename.nii';
NewImgHeader.private.dat.fname = NewImgHeader.fname;

% Write the copy
spm_write_vol(NewImgHeader,Data);
```

The function `spm_matrix(P,'X')` , where X is the type of linear transformation you want to apply ('T' for translation, 'R' for rotation ...), returns an affine transformation matrix given a certain vector P of parameters:

- P(1) - x translation
- P(2) - y translation
- P(3) - z translation
- P(4) - x rotation about - pitch (radians)
- P(5) - y rotation about - roll (radians)
- P(6) - z rotation about - yaw (radians)
- P(7) - x scaling
- P(8) - y scaling
- P(9) - z scaling
- P(10) - x affine
- P(11) - y affine
- P(12) - z affine

Write the codes that will open the structural image do the following:

- move the image by by 15 mm along the x dimension, -58 mm along the y dim and 65 mm along the z dimension and then save this new image,
- rotate an image by 90 degrees around the x axis (pitch) and then save this new image,

- applies the translation, then the rotation and then save this new image,
- applies the rotation, then the translation and then save this new image.

Once you have done that use the **Check reg** button to open the 4 images you have created and compare them to the original.