Python and Neuroimaging: Creating a Video from Structural Data for Beginners

This is a short practical introduction to the process of visualising MRI data in Python with the aim of creating a video file.

There are many ways MRI data can be visualised in Python and several good tutorials already exist. One of the best is Carsten Kleins excellent series which was the inspiration and background for this text along with the code from the NiBabel tutorial.

However, the following short text not only focuses on a visualisation of the data but also on animating a video which can be used to get an overview over all slices from an MRT data set for further analysis.

The text is aimed at readers who have a theoretical understanding of the process and fundamental experience in Python programming, but do not have practical experiences in visualising such data in Python. Therefore, there is a short description on how to set up a necessary Python environment in Windows 10 (if requested I can add macOS later).

Why Python?

Python is a programming language that has been developed more and more into a valuable open source option for data analysis. As a high-level language Python is easier to understand, learn and tend to have higher execution speed than low level languages such a C. It is best suited to handle the large data structures that occur with neuroimaging with a variety of libraries and tools.

Moreover, alternative programming languages for analysis of neuroimaging data can be proprietary like MATLAB. These proprietary programming languages are not free to use and require licensing.

However, R is another great open source alternative for data analysis which has similar capabilities to Python. A quick overview on R vs Python can be found here.

What do you need?

1. Build Tools for Visual Studio 2017.

2. A Python distribution: For this project I used Anaconda, use it if you want to follow along, but any other distribution with the basic libraries for scientific work (NumPy, Matplotlib) will suffice.

3. FFmpeg: Free software for handling multimedia files in Python. Can be installed by starting „Anaconda Prompt “and entering:

```
conda install -c conda-forge ffmpeg
```

4. The NiBabel module provides read/write access to neuroimaging file formats in Python. More information about the neuroimaging in Python community and available packages can be found here. To install it start „Anaconda Prompt “and Install NiBabel using pip by entering:

```
pip install nibabel
```

5. The data for I choose is from a study of visual working memory and can be downloaded from the Openneuro project database. Be aware to choose the most recent version of your data from the left menu and take a look at the README to know what to expect of the data.

Download the Anatomical image for subject 1 by navigating the dataset file tree to sub-01/ anat/sub-01_T1w.nii.gz

How do you do it?

For this little project I used The Scientific Python Development Environment or Spyder, which is part of the Anaconda distribution. Start Spyder and set up your working environment and directory.

If the red „Reloaded modules “warnings keep annoying you try disabling them via Tools --> Preferences --> Python interpreter --> User Module Reloader (UMR) --> Show reloaded module list.

For the slideshow we are going to create you need to set IPythons backend to automatic. Go to Tools --> Preferences --> IPython console --> Graphics --> Graphics backend and set the backend to automatic.

1. Loading the file and returning the data: The Format of your file should be Nifty (.nii) to be imported using NiBabels load function, like in this example:

```
import nibabel as nib
img = nib.load('yourdata.nii')
data = img.get_data()
```

2. Rotating and flipping data with numpy: The data that we have loaded needs to be prepared to be visualised from three primarily used neuroimaging planes; transversal, sagittal and coronal.

Numpy is a module which lets you run such specific operations on the returned data array. In the following example I have rotated the array one time by 90 degrees and set it to the first variable. The second variable is set it to data that is three time rotated by 90 degrees. The third variable represents flipped and rotated data.

```
import numpy as np
datarot1 = np.rot90(data, 1)
datarot2 = np.rot90(data, 3)
dataflip = np.fliplr(datarot1)
```

3. Setting up a plot with subplots: For plotting the data I used the matplotlib library with the pyplot framework.

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 3, figsize=[18, 6])
fig.subplots_adjust(wspace =-0.5, hspace = 1)
```

4. Creating a function to display each slice with correct titles: First, I set up a global variable which determines the position of each slice and set it to 0. The function will take in slices of our rotated/flipped data and display a 2d image for each subplot using matplots imshow function.

Because we are going to call this function repeatedly later on it is important to clear the subplots (with .cla()) before displaying. If this is not done the processing will slow down too much. Aside from hiding axis ticks and setting titles, I also set a label on the y axis of the first sub plot which shows the current number of the slice using the pos variable.

```
pos = 0

def show_slices(slices):
    for i, dataslice in enumerate(slices):

        axes[i].cla()

        axes[0].set_yticks([])
        axes[1].set_yticks([])
        axes[2].set_yticks([])
        axes[0].set_xticks([])
        axes[1].set_xticks([])
        axes[2].set_xticks([])

        axes[0].set_title('Sagittal', fontsize=25, color='r')
        axes[1].set_title('Coronal', fontsize=25, color='r')
        axes[2].set_title('Transversal', fontsize=25, color='r')

        axes[0].set_ylabel('Slice {}'.format(pos),fontsize=25, color='r')

        axes[i].imshow(dataslice, cmap='gray', origin='lower')
```

5. Setting up the animation. The animation needs to progress through all slices of the data. To achieve this, I set the current slice to the number of times the animate function is called before the data is displayed by the show_slices function.

```
    def animate(i):
        global pos
        pos += 1

        slice_0 = dataflip[:, :, pos]
        slice_1 = datarot1[:, pos, :]
```

```
    slice_2 = datarot2[pos, :, :]
    (show_slices([slice_0, slice_1, slice_2]))
```

6. Animating and saving. For animation I used matplots FuncAnimation, which repeatedly calls the animate function. You can change the number of runs with the parameter “frames” and the time between images with “interval” (in milliseconds).

Lastly, I saved the animation using the previously installed ffmpeg.

```
import matplotlib.animation as animation

anim = animation.FuncAnimation(fig, animate, frames=190, interval=250)

anim.save('video.mp4', writer='ffmpeg')
```

If you are trying this out for yourself, I would be happy to help you with any problems along the way. Just comment here or write me an email.

You can download the full code of the project here and the video here.