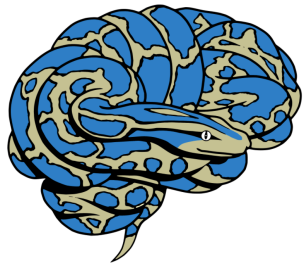
Preprocessing and data QC 2

Nipype (1) : What is Nipype?

Sungwoo Lee
M.S-Ph.D. student

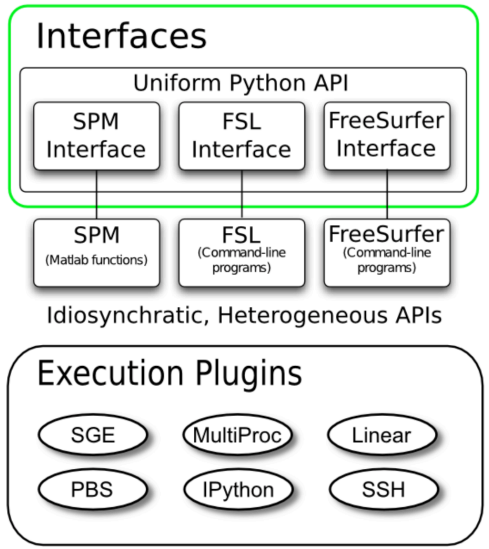
Nipype (1) : What is Nipype?

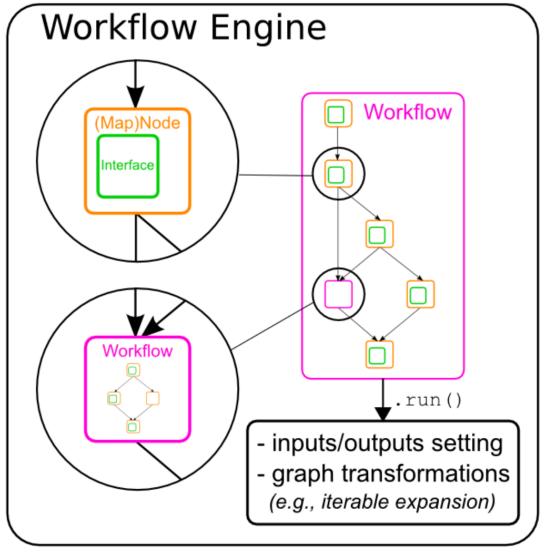


Nipype:
Neuroimaging in Python
Pipelines and Interfaces

What is Nipype?

- open-source, community-developed software package written in **Python**.
- Provides unified way of **interfacing** with heterogeneous neuroimaging software like SPM, FSL, FreeSurfer, AFNI, ANTS, Camino, MRtrix, MNE, Slicer and many more.
- Allows users to create **flexible, complex workflows** consisting of multiple processing steps using any software package above
- Efficient and optimized computation through **parallel execution** plugins

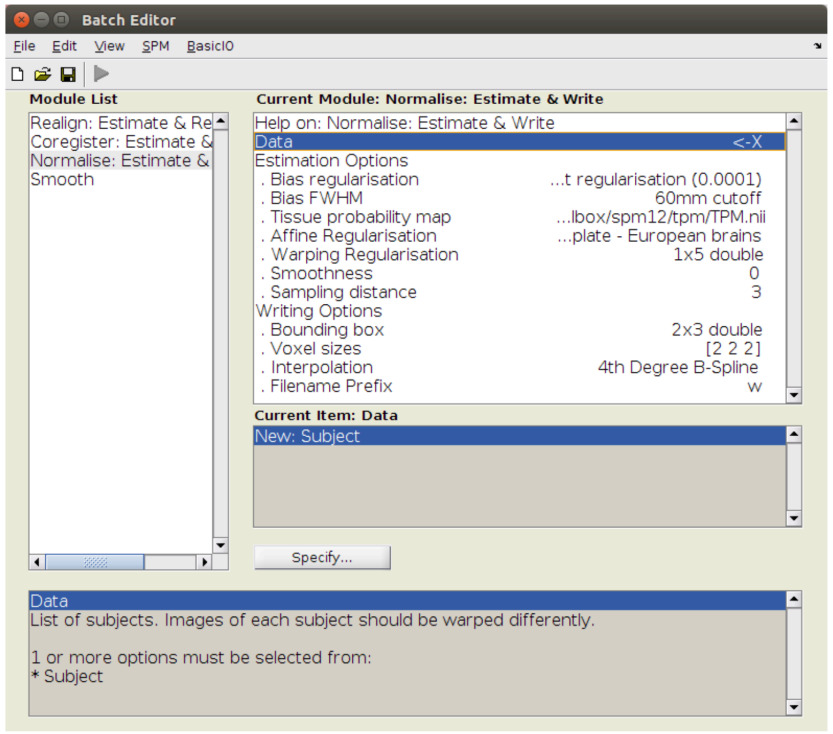




https://miykael.github.io/nipype_tutorial/notebooks/introduction_nipype.html#1

Nipype (1) : What is Nipype?

SPM12



SPM12 in Nipype

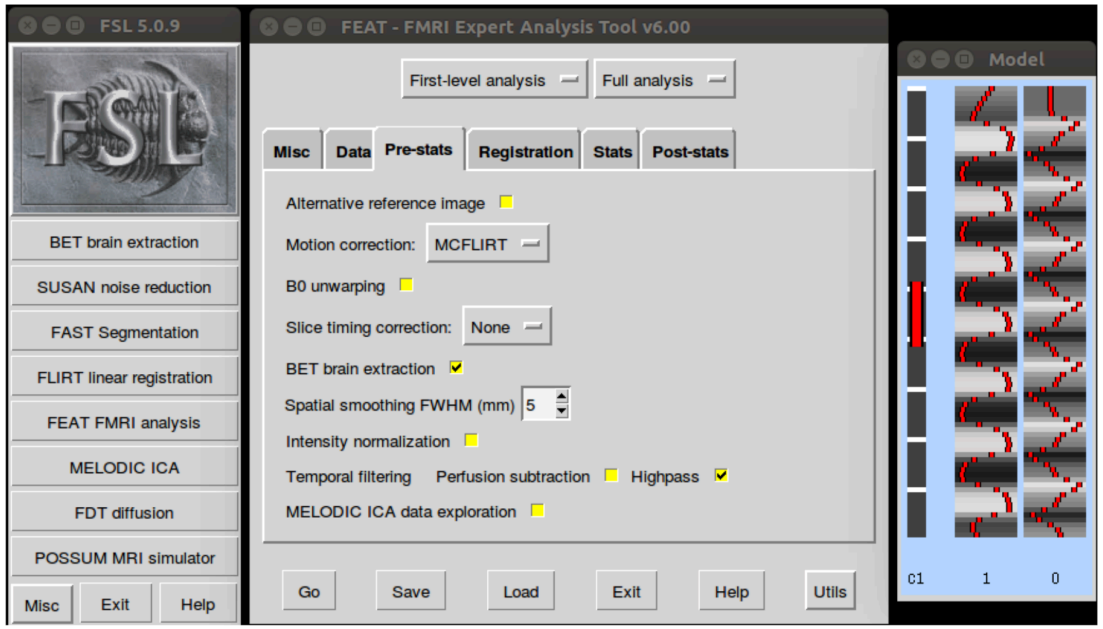
Using SPM12 with Nipype is simpler than any `matlabbatch` and it's intuitive to read:

```
from nipype.interfaces.spm import Smooth
smooth = Smooth()
smooth.inputs.in_files = 'functional.nii'
smooth.inputs.fwhm = 6
smooth.run()
```

https://miykael.github.io/nipype_tutorial/notebooks/introduction_nipype.html#1

Nipype (1) : What is Nipype?

FSL



FSL in Nipype

Nipype makes using FSL even easier:

```
from nipype.interfaces.fsl import MCFLIRT
mcflt = MCFLIRT()
mcflt.inputs.in_file = 'functional.nii'
mcflt.run()
```

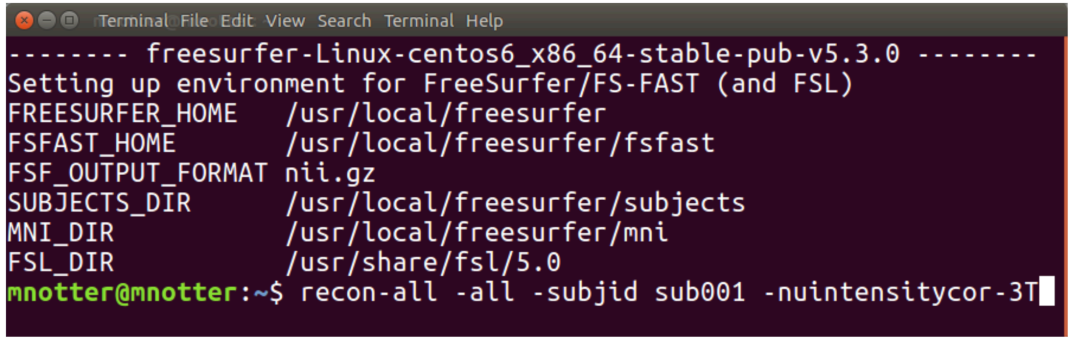
And gives you transparency to what's happening under the hood with one additional line:

```
In [1]: mcflt.cmdline
Out[1]: 'mcflirt -in functional.nii -out functional_mcf.nii'
```

https://miykael.github.io/nipype_tutorial/notebooks/introduction_nipype.html#1

Nipype (1) : What is Nipype?

FreeSurfer



FreeSurfer in Nipype

```
from nipype.interfaces.freesurfer import SurfaceSmooth
smoother = SurfaceSmooth()
smoother.inputs.in_file = "{hemi}.func.mgz"
smoother.iterables = [("hemi", ['lh', 'rh']),
                      ("fwhm", [4, 8]),
                      ("subject_id", ['sub01', 'sub02', 'sub03',
                                      'sub04', 'sub05', 'sub06']),
                      ]
smoother.run(mode='parallel')
```

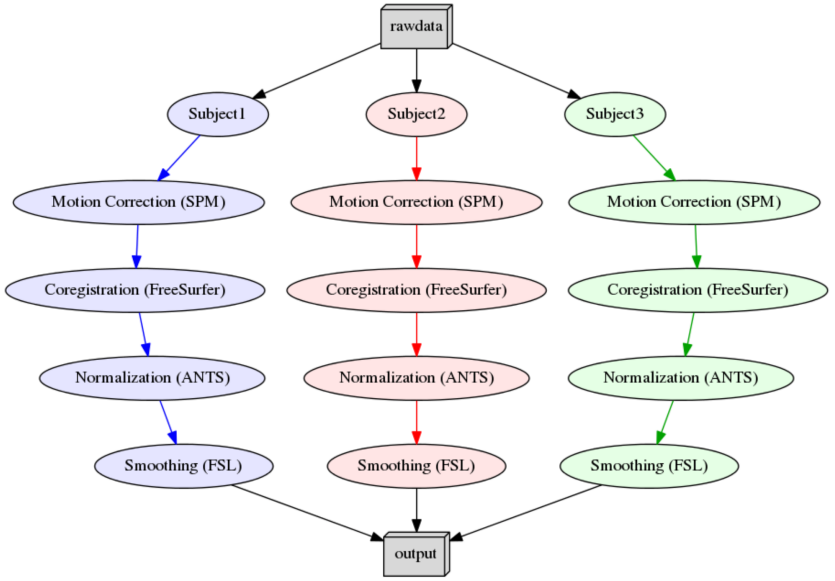
https://miykael.github.io/nipype_tutorial/notebooks/introduction_nipype.html#1

Nipype (1) : What is Nipype?

A Short Example

Let's assume we want to do preprocessing that uses **SPM** for *motion correction*, **FreeSurfer** for *coregistration*, **ANTS** for *normalization* and **FSL** for *smoothing*. Normally this would be a hell of a mess.

It would mean switching between multiple scripts in different programming languages with a lot of manual intervention. **Nipype comes to the rescue!**



https://miykael.github.io/nipype_tutorial/notebooks/introduction_nipype.html#1

Nipype (1) : What is Nipype?

Code Example

The code to create a Nipype workflow like the example before would look something like this:

```
# Import modules
from nipype.interfaces.freesurfer import BBRegister
from nipype.interfaces.ants       import WarpTimeSeriesImageMultiTransform
from nipype.interfaces.fsl        import SUSAN
from nipype.interfaces.spm        import Realign
```

```
# Motion Correction (SPM)
realign = Realign(register_to_mean=True)
```

```
# Coregistration (FreeSurfer)
coreg = BBRegister()
```

```
# Normalization (ANTS)
normalize = WarpTimeSeriesImageMultiTransform()
```

```
# Smoothing (FSL)
smooth = SUSAN(fwhm=6.0)
```

```
# Where can the raw data be found?
grabber = nipype.DataGrabber()
grabber.inputs.base_directory = '~/experiment_folder/data'
grabber.inputs.subject_id = ['subject1', 'subject2', 'subject3']
```

```
# Where should the output data be stored at?
sink = nipype.DataSink()
sink.inputs.base_directory = '~/experiment_folder/output_folder'
```

```
# Create a workflow to connect all those nodes
preprocflow = nipype.Workflow()
```

```
# Connect the nodes to each other
preprocflow.connect([(grabber   -> realign  ),
                     (realign   -> coreg    ),
                     (coreg     -> normalize),
                     (normalize -> smooth   ),
                     (smooth    -> sink     )
                     ])
```

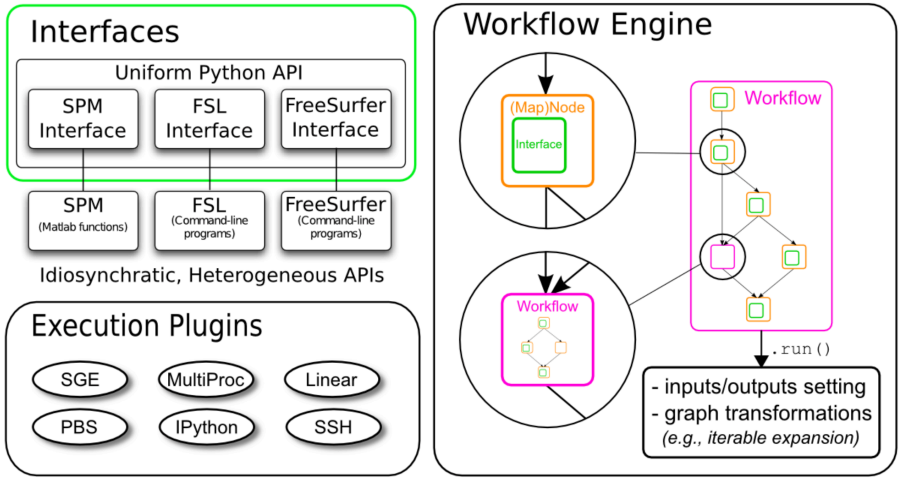
```
# Run the workflow in parallel
preprocflow.run(mode='parallel')
```

https://miykael.github.io/nipype_tutorial/notebooks/introduction_nipype.html#1

Nipype (1) : What is Nipype?

So again, what is Nipype?

Nipype consists of many parts, but the main ones are Interfaces, the Workflow Engine and the Execution Plugins:



- **Interface**: Wraps a program or function
- **Node/MapNode**: Wraps an `Interface` for use in a Workflow
- **Workflow**: A *graph* or *forest of graphs* whose edges represent data flow
- **Plugin**: A component that describes how a `Workflow` should be executed

https://miykael.github.io/nipype_tutorial/notebooks/introduction_nipype.html#1

Nipype (1) : What is Nipype?

Nipype Tutorial

Welcome to the Nipype Tutorial! It covers the basic concepts and most common use cases of Nipype and will teach you everything so that you can start creating your own workflows in no time. We recommend that you start with the introduction section to familiarize yourself with the tools used in this tutorial and then move on to the basic concepts section to learn everything you need to know for your everyday life with Nipype. The workflow examples section shows you a real example of how you can use Nipype to analyze an actual dataset. For a very quick non-imaging introduction, you can check the Nipype Quickstart notebooks in the introduction section.

All of the notebooks used in this tutorial can be found on github.com/miykael/nipype_tutorial. But if you want to have the real experience and want to go through the computations by yourself, we highly recommend you to use a Docker container. More about the Docker image that can be used to run the tutorial can be found here. This docker container gives you the opportunity to adapt the commands to your liking and discover the flexibility and real power of Nipype yourself.

To run the tutorial locally on your system, we will use a Docker container. For this you need to install Docker and download a docker image that provides you a neuroimaging environment based on a Debian system, with working Python 3 software (including Nipype, dipy, matplotlib, nibabel, nipy, numpy, pandas, scipy, seaborn and more), FSL, ANTs and SPM12 (no license needed). We used Neurodocker to create this docker image.

If you do not want to run the tutorial locally, you can also use Binder service. Binder automatically launches the Docker container for you and you have access to all of the notebooks. Note, that Binder provides between 1G and 4G RAM memory, some notebooks from Workflow Examples might not work. All notebooks from Introduction and Basic Concepts parts should work.

For everything that isn't covered in this tutorial, check out the main homepage. And if you haven't had enough and want to learn even more about Nipype and Neuroimaging, make sure to look at the detailed beginner's guide.

https://miykael.github.io/nipype_tutorial/

NeuroHackademy 2019: Satra Ghosh - Nipype

https://www.youtube.com/watch?v=f_3w0coEjzw&list=PLA6PlfxWZPLTLJ2qTN9enG0tkizpmwWaq&index=34&t=1734s

Neurohackademy: Satra Ghosh - Nipype (Workflows)

https://www.youtube.com/watch?v=YjJ9-gxlRJk&list=PLA6PlfxWZPLTLJ2qTN9enG0tkizpmwWaq&index=14

Preprocessing and data QC 2

Nipype (1) : What is Nipype?

Sungwoo Lee
M.S-Ph.D. student