Welcome to NeuroPype

Dmitrii Altukhov, Karim Jerbi, David Meunier and Annalisa Pascarella

May 16th, 2017

Université de Montréal

Nipype

‹ Articles

ORIGINAL RESEARCH ARTICLE

Front. Neuroinform., 22 August 2011 | https://doi.org/10.3389/fninf.2011.00013

Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in Python

Krzysztof Gorgolewski[1]*, Christopher D. Burns[2], Cindee Madison[2], Dav Clark[3], Yaroslav O. Halchenko[4], Michael L. Waskom[5,6] and Satrajit S. Ghosh[7]

[1] Neuroinformatics and Computational Neuroscience Doctoral Training Centre, School of Informatics, University of Edinburgh, Edinburgh, UK
[2] Helen Wills Neuroscience Institute, University of California, Berkeley, CA, USA
[3] Department of Psychology, University of California, Berkeley, CA, USA
[4] Department of Psychological and Brain Sciences, Dartmouth College, Hanover, NH, USA
[5] Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA, USA
[6] McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA, USA
[7] Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA, USA

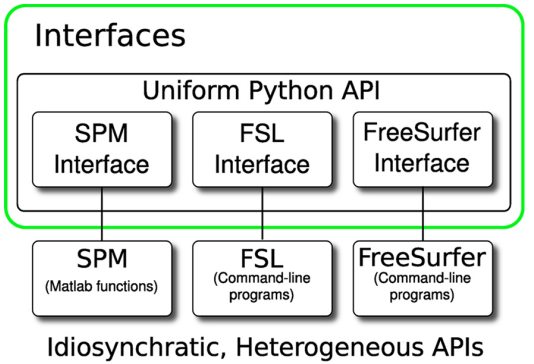
Current neuroimaging software offer users an incredible opportunity to analyze their data in different ways, with different underlying assumptions. Several sophisticated software packages (e.g., AFNI, BrainVoyager, FSL, FreeSurfer, Nipy, R, SPM) are used to process and analyze large and often diverse (highly multi-dimensional) data. However, this heterogeneous collection of specialized applications creates several issues that hinder replicable, efficient, and optimal use of neuroimaging analysis approaches: (1) No uniform access to neuroimaging analysis software and usage information; (2)

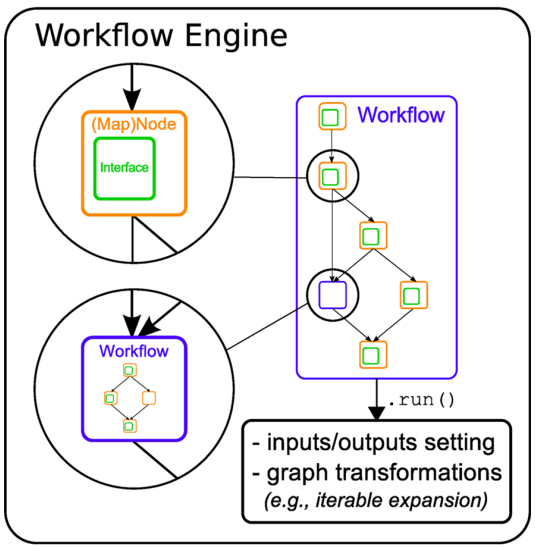
Nipype

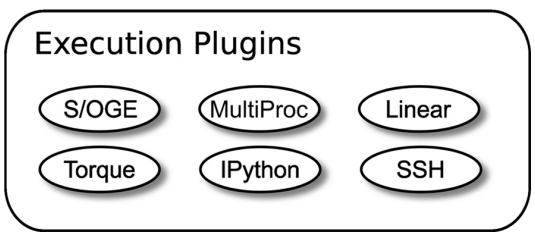
- The workflow design is based on the Nipype framework, allowing in a very readable fashion the design of processing pipelines
- Nipype (Neuroimaging in Python: Pipelines and Interfaces) is an open-source, community developed, **Python based** software package that easily interfaces with existing software for efficient analysis of neuroimaging data and **rapid comparative development of algorithms**.
- Nipype provides **Interfaces** to existing neuroimaging software with uniform semantics and facilitates interactions between these packages using **Workflow**
- The workflow execution engine has a plug-in architecture and supports both local execution on multi-core machines and remote execution on clusters

Nipype:
Neuroimaging in Python
Pipelines and Interfaces

Nipype: main components





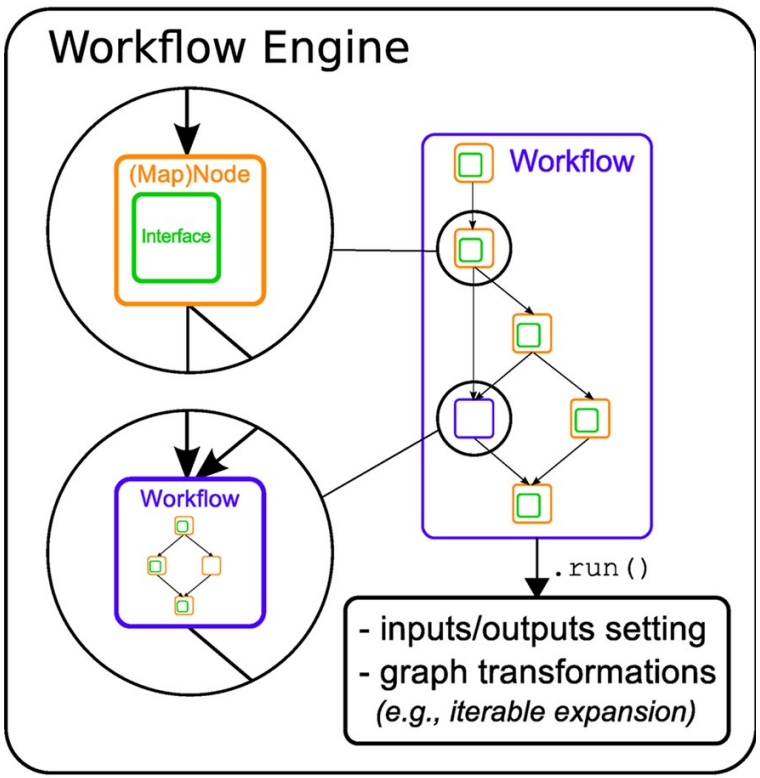


Nipype consists of three main components:

- **Interfaces** to external tools that provide a unified way for setting inputs, executing and retrieving outputs
  - the goal of Interfaces is to provide a uniform mechanism for accessing analysis tool from neuroimaging software packages (e.g. Freesurfer, FSL, SPM, ...)
- a **workflow engine** that allows to create analysis pipelines by connecting inputs and outputs of interfaces as a directed acyclic graph (DAG)
- **plug-ins** that execute workflows either locally or in a distributed processing environment
  - no changes are needed to the Workflow to switch between these execution modes. The user simply calls the Workflow **run** function with a different plug-in and its arguments

Interfaces and Nodes

Nipype provides a framework for connecting Interfaces to create a **data analysis Workflow**



- In order to be used in a Workflow the Interfaces have to be encapsulated in **Node objects**
  - they execute the underlying Interface in their own **uniquely named directories**, thus providing a mechanism to isolate and track the outputs resulting from the Interface execution
- Interfaces encapsulated into Nodes can be connected together within a **Workflow**: by connecting the outputs of some Node to inputs of another one, the user implicitly specifies dependencies
  - the dependencies in a Workflow are represented internally as a **DAG**
- Workflow themselves can be a Node of the Workflow graph
- Node provides also a easy way to implement functions defined by the user

Scratch code

```
main_workflow = pe.Workflow(name=correl_analysis_name)
main_workflow.base_dir = main_path

infosource = create_infosource()  # info source Node
datasource = create_datasource()  # data source Node

main_workflow.connect(infosource, 'subject_id', datasource, 'subject_id')

create_ts_node = pe.Node(interface=Function(input_names=['raw_fname'],
                                            output_names=['ts_file'],
                                            function=create_ts),
                         name='create_ts')

main_workflow.connect(datasource, 'raw_file', create_ts_node, 'raw_fname')

spectral_workflow = \
    create_pipeline_time_series_to_spectral_connectivity(main_path,
                                                         con_method=con_method)

main_workflow.connect(create_ts_node, 'ts_file', spectral_workflow, 'inputnode.ts_file')

main_workflow.run(plugin='MultiProc', plugin_args={'n_procs': 8})
```

Data Grabber

- The DataGrabber Interface allows the user to define **flexible search patterns** which can be parameterized by user defined inputs (such as subject ID, session, etc.)
- This Interface can adapt to a wide range of directory organization and file naming conventions
- If we parameterize it with subject ID, we can run the same Workflow for different subjects. We automate this by iterating over a list of subject IDs, by setting the iterables property of the DataGrabber Node for the input subject_id.

Iterables

- For various neuroimaging tasks, there is often a need to explore the impact of variations in parameter settings
- To enable such parametric exploration, Nodes have an attribute called iterables
- When an iterable is set on a Node input, the Node, and its subgraph are executed for each value of the iterable input
- Iterables can also be set on multiple inputs of a Node:
  somenode.iterables = [('input1', [1,2,3]), ('input2, ['a,' 'b'])])
  In such cases, every combination of those values is used as a parameter set (the prior example would result in the following parameter sets: (1, 'a'), (1, 'b'), (2, 'a'), etc).

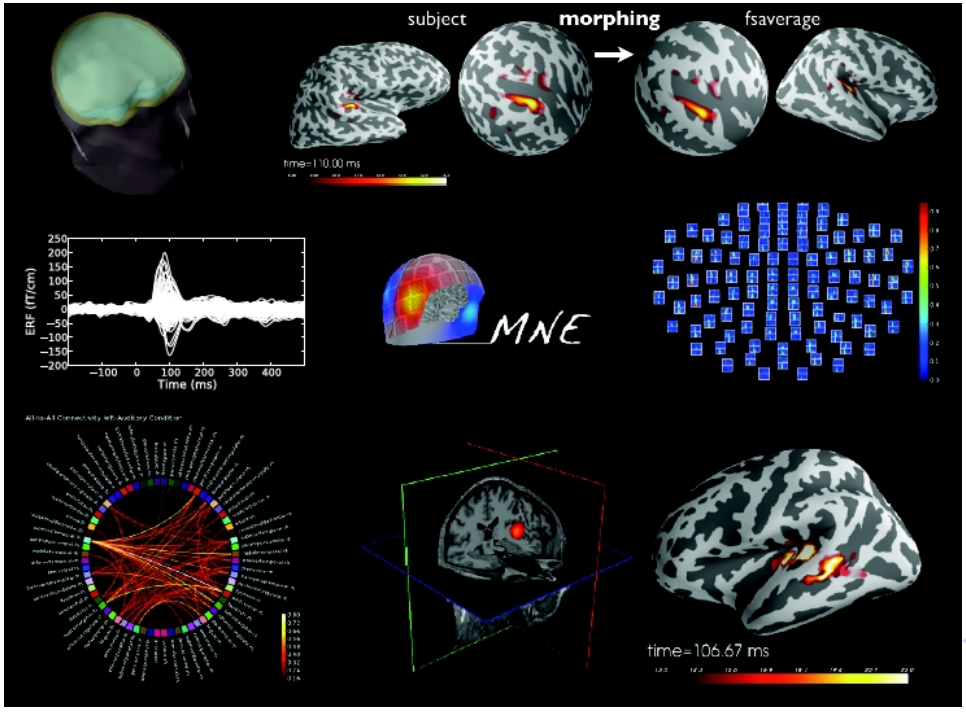
A common use-case of iterables is to execute the same Workflow for many subjects in an experiment and to simultaneously look at the impact of parameter variations on the results of the Workflow.

Neuropype Packages

NeuroPype-ephy

NeuroPype-ephy is a package based on **MNE-python** software http://martinos.org/mne/ and includes pipelines for MEG/EEG data analysis. Current implementations allow for

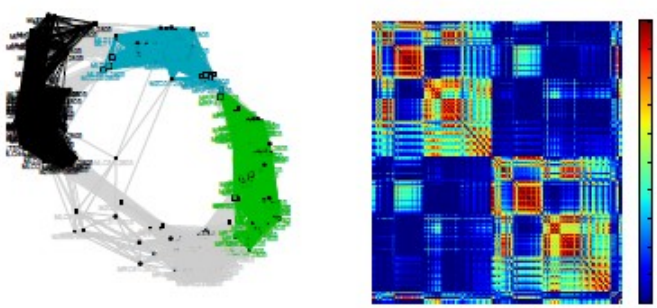
- MEG/EEG data import
- MEG/EEG data pre-processing and cleaning by an automatic removal of eyes and heart related artefacts
- sensor or **source-level** connectivity analyses



NeuroPype-graph

NeuroPype-graph is a package based on **radatools** software http://deim.urv.cat/ sergio.gomez/radatools.php and includes pipelines for graph theoretical analysis of neuroimaging data. Current implementations allow to construct pipelines

- from nifti 4D (after preprocessing) to connectivity matrices
- from connectivity matrices to graph analysis
- from integer matrices (normally coclassification matrices) to graph analysis



NeuroPype-cli

Command line interface for neuropype

Key idea: Use globbing to improve flexibility of the pipeline

- Flexible input
- Connect nodes on the go
- Same pattern for launching remotely
- Works on clusters
- **help** pages for each option and command

Installation

Install neuropype_ephy

```
git clone https://github.com/annapasca/neuropype_ephy.git
cd neuropype_ephy
pip install .
cd ..
```

see README for more information

Install neuropype_cli

```
git clone https://github.com/dmalt/neuropype_cli.git
cd neuropype_cli
pip install .
cd ..
```

see README for more information.

Install neuropype_graph

```
git clone https://github.com/davidmeunier79/neuropype_graph.git
cd neuropype_graph
pip install .
cd ..
```

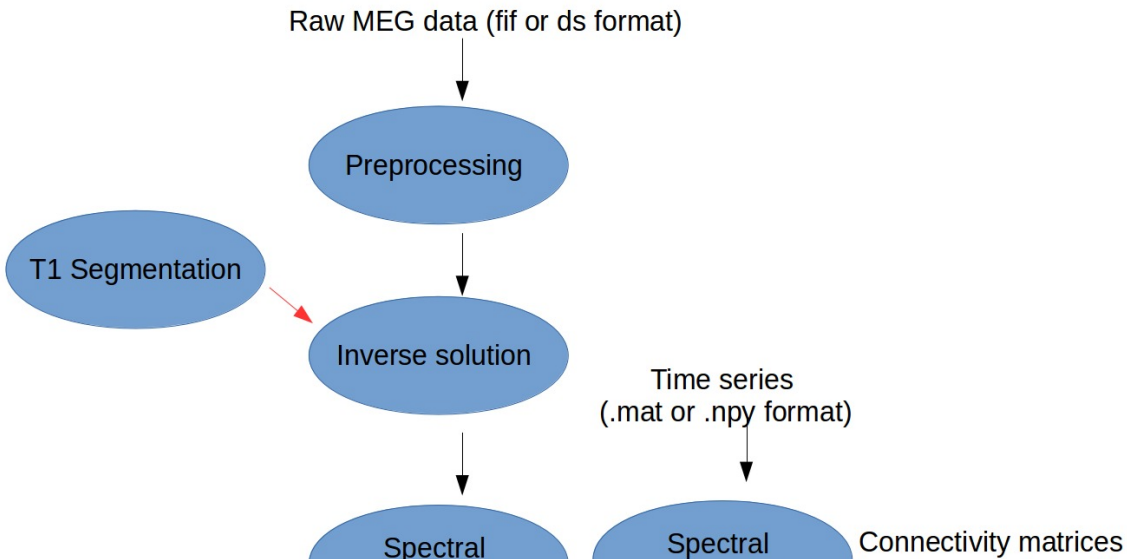
Look at the NeuroPype website

NeuroPype online tutorial

NeuroPype

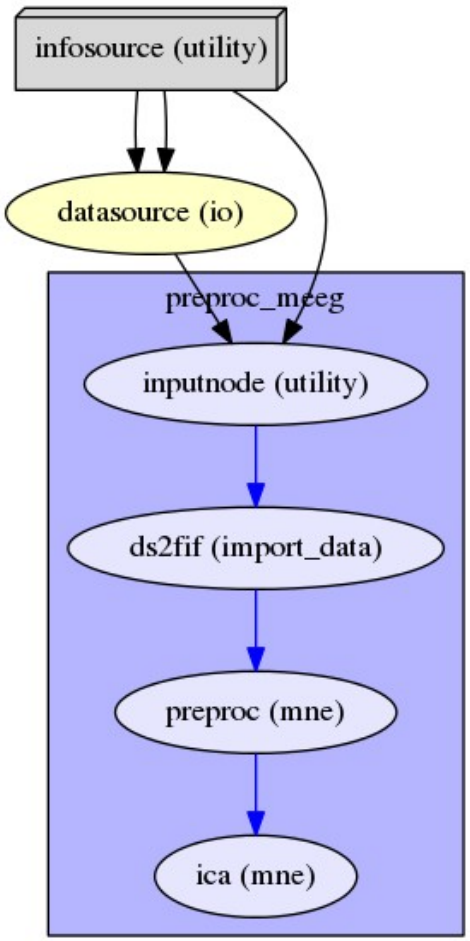
Neuropype packages define a set of different pipelines (also called workflow) that can be used stand-alone or as brick of a bigger workflow: the input of a pipeline will be the output of another pipeline.
For each possible workfdlow the input data can be specified in three different ways:

- raw MEG data in .fif and .ds format
- time series of data in sensor or source space in .mat (Matlab) or .npy (Numpy) format
- connectivity matrices in .mat (Matlab) or .npy (Numpy) format



Preprocessing Pipeline

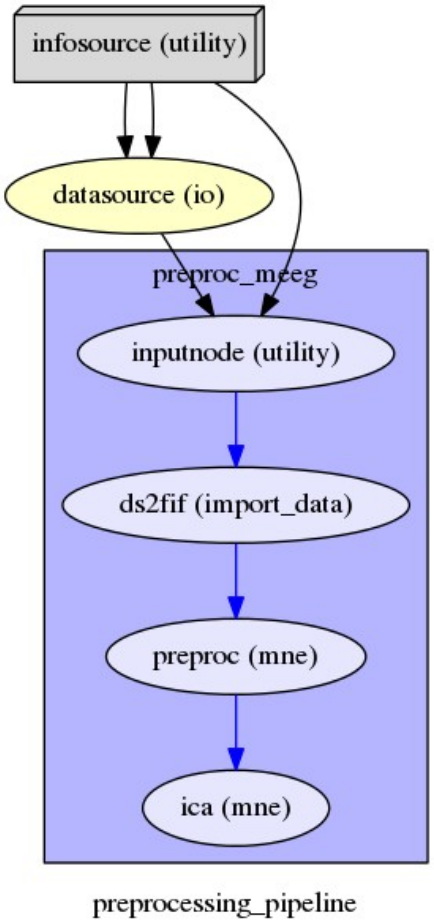
The **preprocessing pipeline** runs the **ICA algorithm** for an automatic removal of eyes and heart related artefacts. A report is automatically generated and can be used to correct and/or fine-tune the correction in each subject



preprocessing_pipeline

- The Nodes of the pipeline (**ds2fif, preproc, ica**) are based on the **MNE Python** functions performing the decomposition of the MEG/EEG signal using an ICA method
- **input:**
  - MEG raw data (**.ds** or **.fif** format)
- **output:**
  - **ica solution**
  - **cleaned raw** data
  - **report** in html format
  - input of another pipeline
    - ★ **power** pipeline
    - ★ **spectral connectivity** pipeline
    - ★ **source reconstruction** pipeline

Preprocessing Nodes

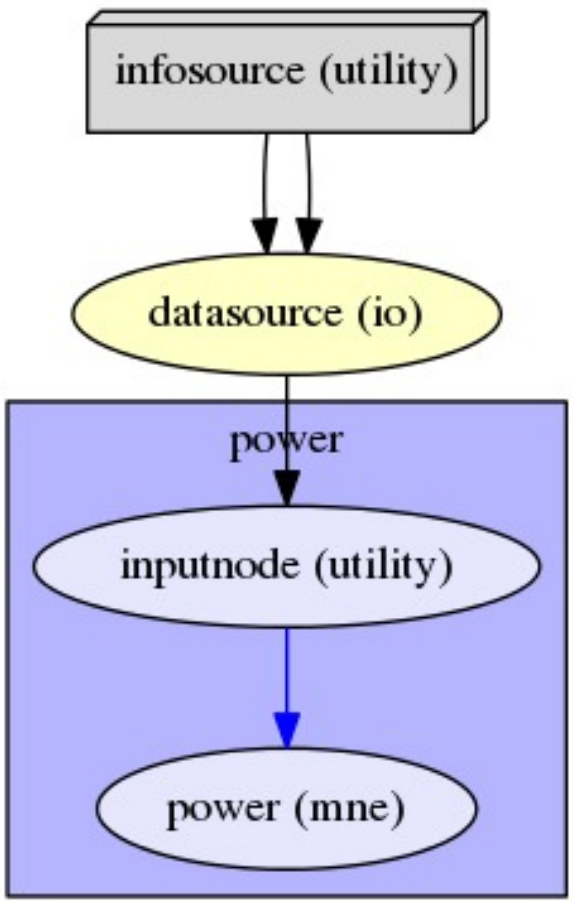


- **ds2fif:** converts from ds to fif data format if the raw data is recorded by a CTF device
  - **data_type** variable is True
- **preproc:** performs filtering and downsampling of input raw data
- **ica:** computes ICA solution on raw fif data

It's a good rule to inspect the report file saved in the workflow directory to look at the excluded ICA components. It is also possible to include and exclude more components by using either a jupyter notebook or the preprocessing pipeline with different flag parameters.

Power Pipeline

The power pipeline computes the power spectral density (**PSD**) on epochs or raw data on sensor space.



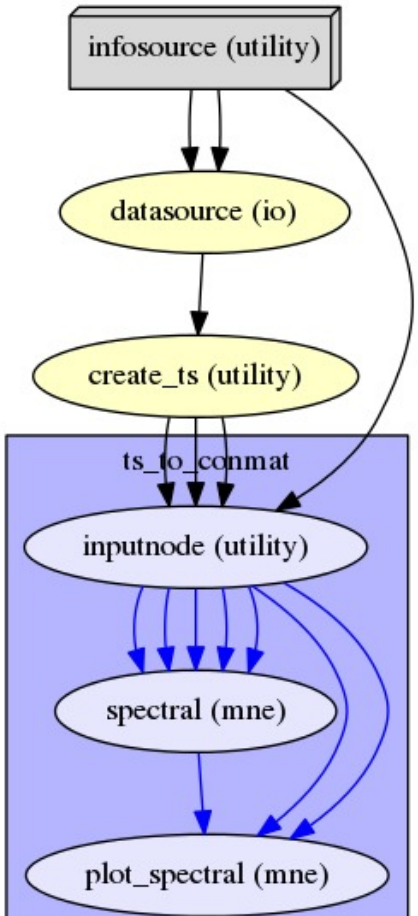
power_pipeline

- The Node of the pipeline (**power**) wraps the **MNE Python** functions computing the PSD using Welch's or multitapers method
- **input:**
  - MEG raw data **.fif** format)
- **output:**
  - **psd tensor and frequencies in .npz format**

Connectivity Pipeline

The connectivity pipeline computes connectivity matrix on sensor or source space.

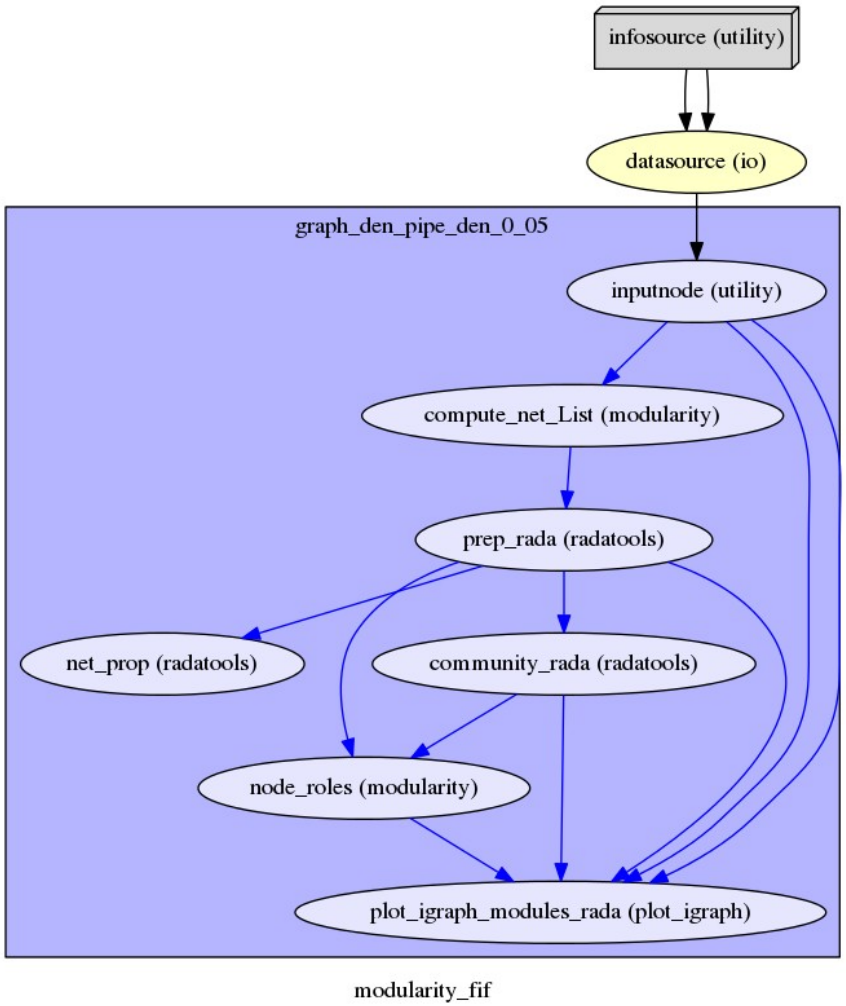
- **the frequency-domain:** Coherence, Coherency, Imaginary Coherence, Phase Locking Value, Phase-Lag Index
- **the time-domain:** correlation of evelope, phase-amplitude coupling (using Brainpipe)



- The Nodes of the pipeline (**spectral, plot_spectral**) wrap the **MNE Python** functions computing the connectivity measures
- **input:**
  - time series matrix in sensor or source space (**.mat** format)
  - output of preprocessing pipeline or source reconstruction pipeline (time series in **.npy** format)
- **output:**
  - **spectral connectivty matrix in .npy format**

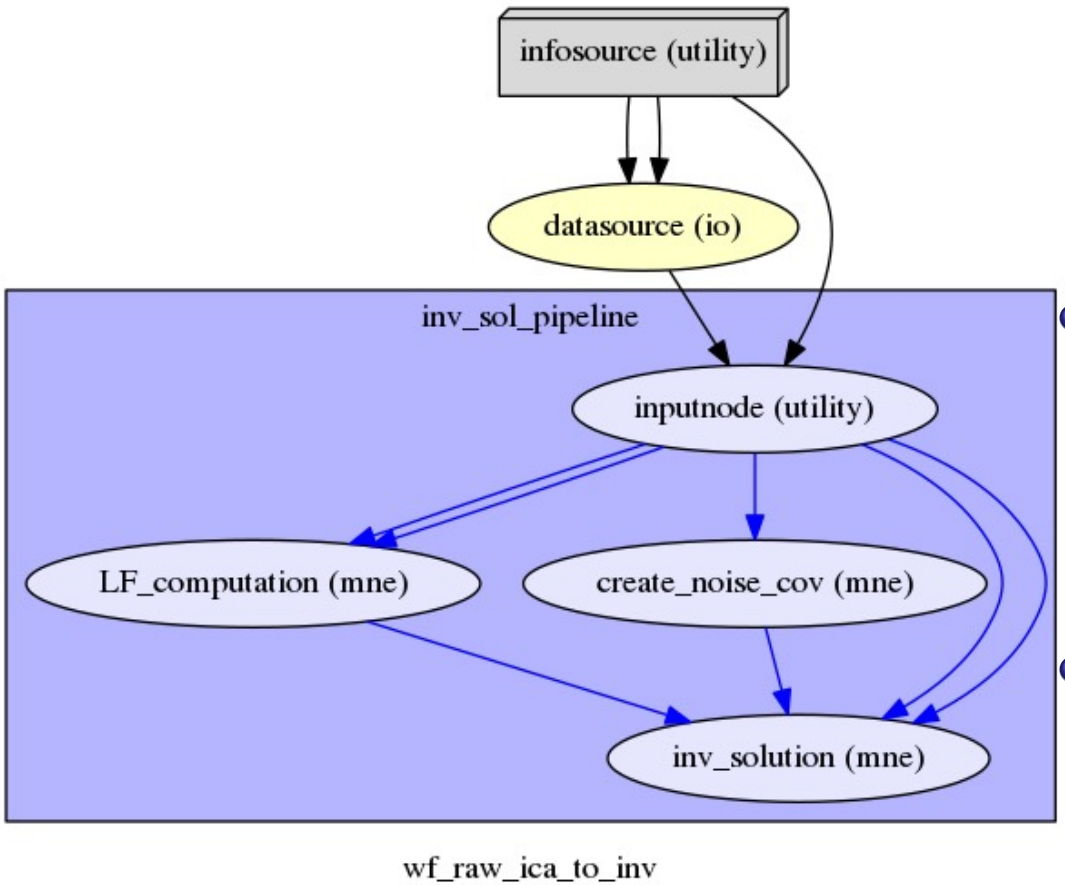
Modularity Pipeline

Pipeline from connectivity matrices to graph analysis



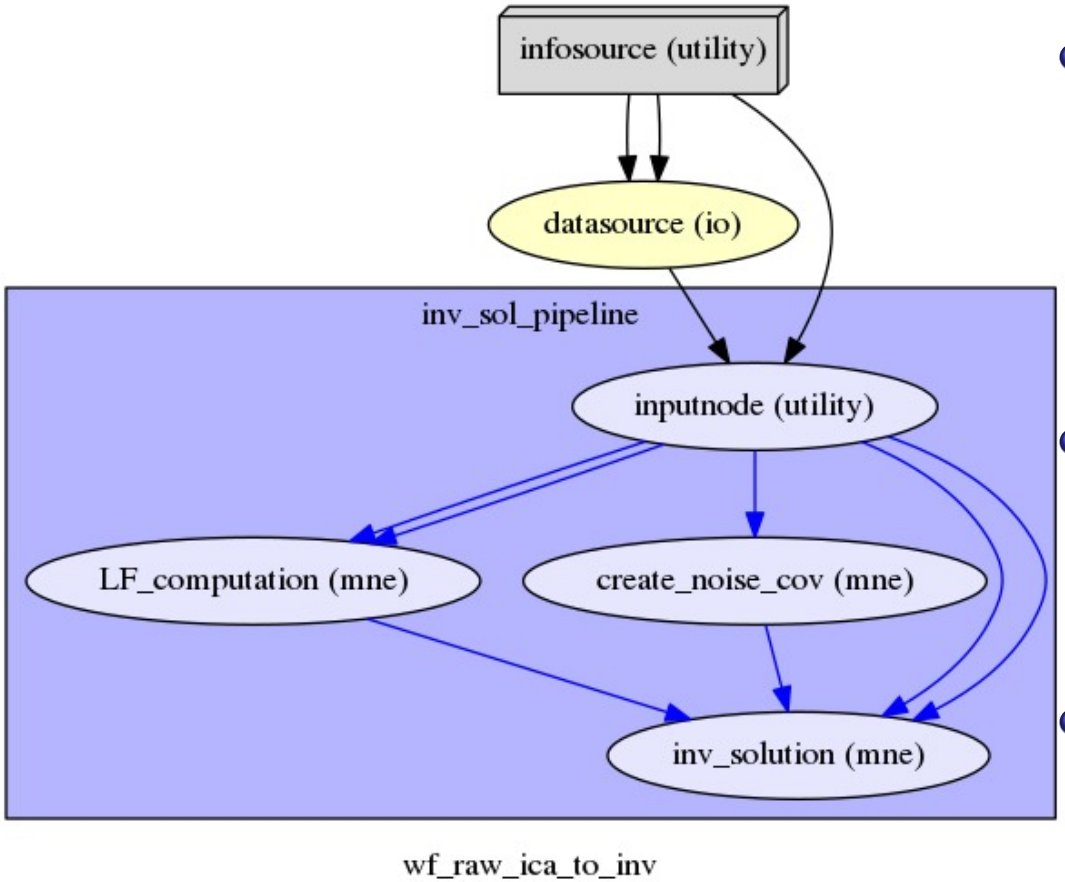
Inverse solution Pipeline

The **inverse solution pipeline** performs source reconstruction from raw/epoched data.



- The Nodes of pipeline (**LF_computation**, **create_noise_cov**, **inv_solution**) wrap the **MNE Python** functions performing the source reconstruction steps
- **input:**
  - raw/epoched data given by the user (**.fif** format)
  - output of the **preprocessing pipeline** (cleaned raw data)
- **output:**
  - source reconstruction matrix
  - input of the **spectral connectivity pipeline**

Inverse solution Nodes



- **Lead Field Node:** computes the Lead Field matrix
  - **BEM** method
  - surface vs **mixed** source space
  - the **output** of the Node will be the Lead Field Matrix
- **Noise Covariance Node:** computes the noise covariance matrix
  - **empty room data**
  - pre-stimulus baseline
- **Inverse Solution Node:** estimates the time series of the neural sources
  - wMNE, sLORETA, dSPM

We need a **template MRI** or an **individual MRI**: to **segment the MRI data** we can use a **FreeSurfer workflow** . . .

Note: the outputs of the Lead Field and Noise Covariance Nodes are the inputs of the Inverse Solution Node

Workflow

Workflow

Nipype explicitly implements a pipeline as a graph

- it is easy to follow what steps are being executed and in what order
- it makes easier to go back and change things by simply reconnecting different outputs and inputs or by inserting new Nodes
- this alleviates the tedious component of scripting where one has to manually ensure that the inputs and outputs of different processing calls match and that operations do not overwrite each other's outputs

A Workflow provides a detailed description of the processing steps and how data flows between Interfaces.

The Workflow engine validates that all nodes have unique names, ensures that there are no cycles, and prevents connecting multiple outputs to a given input.