NiPype: A Pypelined Approach to Neuroimaging Data Analysis

http://nipy.sourceforge.net/nipype

Satrajit Ghosh satra@mit.edu
Massachusetts Institute of Technology
Acknowledgements

**People**

core *nipype* developers:
chris burns, dav clark, satrajit ghosh, cindee madison

other contributors (*financial, social, technical and testing support)*:
matthew brett, mark d’esposito, stefan ehrlich, the gablab, john gabrieli, randy gollub, scott gorlin, chris gorgolewski, doug greve, yaroslav halchenko, michael hanke, oliver hinds, arno klein, jarrod millman, alfonso nieto-castanon, mark pearrow, fernando perez, jean baptiste-poline, alexis roche, yannick schwartz, jonathon taylor, jason tourville, bertrand thirion, rosalia tungaraza, gael varoquaux, susan whitfield-gabrieli ...

my closest supporters:
amie, katrien

**Funding**

<table>
<thead>
<tr>
<th>NIBIB</th>
<th>R03 EB008673</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIs:</td>
<td>Satrajit Ghosh, Susan Whitfield-Gabrieli, MIT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NIMH</th>
<th>R01 MH081909</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIs:</td>
<td>Marc D’Esposito, UC, Berkeley</td>
</tr>
</tbody>
</table>

**Software**

| FSL |
| FreeSurfer |
| NiPy |
| SPM |

**Institutions**

- Boston University
- Dartmouth College
- Massachusetts Institute of Technology
- Neurospin, France
- Stanford
- University of California, Berkeley
- University of Edinburgh
- University of Washington, Seattle
Neuroimaging Pypelines

Outline

- origin
- python
- goals
- architecture
- challenges
Region-of-interest Analysis of Parcellated Imaging Data
Reducing inter-subject anatomical variability

ARtifact detection Tools
Quality assurance
Region-of-interest Analysis of Parcellated Imaging Data
Reducing inter-subject anatomical variability

Artifact detection Tools
Quality assurance

MATLAB-based
SPM-focused
Lab-centric
<p>| Neuroimaging Pypelines | distribution |</p>
<table>
<thead>
<tr>
<th>Neuroimaging Pypelines</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>solutions?</td>
<td></td>
</tr>
</tbody>
</table>
Neuroimaging Pypelines

solutions?
- Write it in C/C++
Neuroimaging Pipelines

Distribution solutions?

- Write it in C/C++
- Wrap it with swig/mex
solutions?

- Write it in C/C++
- Wrap it with swig/mex
- Distribute with MatlabRuntime
Neuroimaging Pipelines

distribution

solutions?

- Write it in C/C++
- Wrap it with swig/mex
- Distribute with MatlabRuntime

or more problems?
Neuroimaging Pipelines distribution

solutions?

- Write it in C/C++
- Wrap it with swig/mex
- Distribute with MatlabRuntime

or more problems?

- Code maintenance
Neuroimaging Pypelines

distribution

solutions?
- Write it in C/C++
- Wrap it with swig/mex
- Distribute with MatlabRuntime

or more problems?
- Code maintenance
- Code readability
Neuroimaging Pipelines - distribution

solutions?

- Write it in C/C++
- Wrap it with swig/mex
- Distribute with MatlabRuntime

or more problems?

- Code maintenance
- Code readability
- Interoperability
Neuroimaging Pipelines

distribution

solutions?
- Write it in C/C++
- Wrap it with swig/mex
- Distribute with MatlabRuntime

or more problems?
- Code maintenance
- Code readability
- Interoperability
- Data formats
<table>
<thead>
<tr>
<th>Neuroimaging Pipelines</th>
<th>data formats</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SPM</strong></td>
<td>nii (3D/4D)</td>
</tr>
<tr>
<td></td>
<td>img/hdr (analyze, nifti)</td>
</tr>
<tr>
<td></td>
<td>MAT (matlab)</td>
</tr>
<tr>
<td><strong>FSL</strong></td>
<td>nii(.gz)</td>
</tr>
<tr>
<td><strong>FreeSurfer</strong></td>
<td>nii(.gz)</td>
</tr>
<tr>
<td></td>
<td>mgh.gz</td>
</tr>
</tbody>
</table>
Neuroimaging Pypelines | goals (part i)
--- | ---
Easy to program and document
Good numerical support
Cross-platform
Easy to distribute
Existing neuroimaging infrastructure
(support for multiple data formats)
Neuroimaging Pipelines

python!
6.00 Introduction to Computer Science and Programming

As taught in: Fall 2008

Level:
Undergraduate

Instructors:
Prof. Eric Grimson
Prof. John Guttag

Course Features
Course Description
Technical Requirements

Many of the problem sets focus on specific topics, such as virus population dynamics, word games, protein sequences, or simulating the movement of a Roomba (Roomba photograph courtesy of Stephanie Booth on Flickr; virus image courtesy of the CDC; Boggle photograph courtesy of Angelina on Flickr; protein image courtesy of the Lawrence Berkeley National Laboratory.)

Course Features

- Video lectures
- Exams and Solutions
- Assignments (no solutions)

Course Description

This subject is aimed at students with little or no programming experience. It aims to provide students with an understanding of the role computation can play in solving problems. It also aims to help students, regardless of their major, to feel justifiably confident of their ability to write small programs that allow them to accomplish useful goals. The class will use the Python™ programming language.
• High-level language (encourages code reuse)
• Well-designed language
• Cross-platform
• Implementations are freely available
• Extensive community and commercial support
• Strong numerical and scientific computing support
• Interactive command line (instant development)
• Test-Driven Development
• Documentation generation made easy
<table>
<thead>
<tr>
<th>Neuroimaging Pipelines</th>
<th>Scientific Python</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ipython</strong></td>
<td><strong>scipy + numpy</strong></td>
</tr>
<tr>
<td>![IPython logo]</td>
<td>![SciPy logo]</td>
</tr>
<tr>
<td><strong>sympy</strong></td>
<td><strong>matplotlib</strong></td>
</tr>
<tr>
<td>![SymPy logo]</td>
<td>![Matplotlib logo]</td>
</tr>
</tbody>
</table>

**Logos and Graphics**

- IPython logo
- SciPy logo
- NetworkX logo
- SymPy logo
- Matplotlib logo
- Mayavi logo
scipy:
- statistics
- optimization
- numerical integration
- linear algebra
- Fourier transforms
- signal processing
- image processing
- genetic algorithms
- ODE solvers
- special functions

numpy:
- a powerful N-dimensional array object
- basic linear algebra functions
- basic Fourier transforms
- sophisticated random number capabilities
```python
from scipy import *  
from numpy import *  
import pylab  

x = arange(0.1, 0.101, 0.01)  

for k in arange(0.1, 5.5):  
    y = special.jv(k, x)  
    plot(x, y)  
    f = lambda x: -special.jv(k, x)  
    x_max = optimize.fminbound(f, 0.6)  
    plot([x_max], [special.jv(k, x_max)], 'ro')
```

```
Neuroimaging Pipelines

matplotlib
Neuroimaging Pipelines

networkx
Neuroimaging Pypelines

mayavi + tvtk
| Neuroimaging Pipelines | mayavi + tvtk |
Neuroimaging Pipelines

mayavi + tvtk
Neuroimaging Pypelines: mayavi + tvtk
Neuroimaging Pipelines

mayavi + tvtk
Neuroimaging Pipelines: mayavi + tvtk
Neuroimaging Pypelines  mayavi + tvtk
Neuroimaging Pipelines

```python
>>> from sympy import *
>>> x = Symbol('x')
>>> diff(sin(x), x)
cos(x)
>>> diff(sin(2*x), x)
2*cos(2*x)
>>> diff(tan(x), x)
1 + tan(x)**2

You can check, that it is correct by:

```python
>>> limit((tan(x+y)-tan(x))/y, y, 0)
1 + tan(x)**2
```
interactive shell with history
but comes with a lot of magic

kernel for distributed computation
(ssh, mpi, torque and threading)
interactive shell with history
but comes with a lot of magic

kernel for distributed computation
(ssh, mpi, torque and threading)
<table>
<thead>
<tr>
<th>Neuroimaging Pipelines</th>
<th>compneurobiopy</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdp</td>
<td>visionegg</td>
</tr>
<tr>
<td>mdp</td>
<td>visionegg</td>
</tr>
<tr>
<td>pymvpa</td>
<td>pyschopy</td>
</tr>
<tr>
<td>pymvpa</td>
<td>pyschopy</td>
</tr>
<tr>
<td>biopython</td>
<td>biopython</td>
</tr>
</tbody>
</table>
Python in neuroscience

Hosted by
Rolf Kotter, rk@donders.ru.nl

Guest editors
James A. Bednar (University of Edinburgh, UK)
Andrew Davison (UNIC, CNRS, France)
Markus Diesmann (RIKEN Brain Science Institute, Japan)
Marc-Oliver Gewaltig (Honda Research Institute Europe GmbH, Germany)
Michael Hines (Yale University, USA)
Ellif Muller (LCN-EPFL, Switzerland)

ABOUT THIS SPECIAL TOPIC

Python is rapidly becoming the de facto standard language for systems integration. Python has a large user and developer-base external to the neuroscience community, and a vast module library that facilitates rapid and maintainable development of complex and intricate systems.

In this special topic, we highlight recent efforts to develop Python modules for the domain of neuroscience software and neuroinformatics:

- simulators and simulator interfaces
- data collection and analysis
- sharing, re-use, storage and databasing of models and data
- stimulus generation
- parameter search and optimisation
- visualization
- VLSI hardware interfacing

Moreover, we seek to provide a representative overview of existing mature Python modules for neuroscience and neuroinformatics, to demonstrate a critical mass and show that Python is an appropriate choice of interpreter interface for future neuroscience software development.
Neuroimaging Pipelines

NiPy

http://nipy.sourceforge.net/

image courtesy dr. arno klein
Neuroimaging Pypelines

why

LONI

Camba
Neuroimaging Pipelines

Compare algorithms

- Motion Correction
- Coregistration
- Normalization
- Smoothing
- Model Specification
- Model Estimation
- Statistical Inference

Smoothing
- SPM
- FreeSurfer

Model Specification
- Model Estimation

Model Specification
- Model Estimation

Statistical Inference
Neuroimaging Pipelines

Compare algorithms

- Motion Correction
- Coregistration
- Normalization
- Smoothing
- Model Specification
- Model Estimation
- Statistical Inference

- Smoothing
  - SPM

- Smoothing
  - FreeSurfer

Accessibility

- Preprocessing
- ART: Quality Assurance
- Model Specification
- Model Estimation
- Statistical Inference

Custom Software
• Lightweight
• Open-source, community supported and developed
• Extensible (plugin) framework based on a high-level language
• Integrates different neuroimaging software (FSL, SPM, etc.,)
• Interoperability (including SPM)
• Distributed, non-redundant processing
• Minimize data redundancy
• Bring consistency, repeatability and clarity into analysis
• Teaching tool
Neuroimaging Pypelines approach

Target three levels of users

drag and droppers
scripters
coders
Target three levels of users

- Drag and droppers
- Scripters
- Coders
<p>| Neuroimaging Pipelines | architecture |</p>
<table>
<thead>
<tr>
<th>Neuroimaging Pipelines</th>
<th>architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td></td>
</tr>
<tr>
<td>Algorithms</td>
<td></td>
</tr>
</tbody>
</table>
Neuroimaging Pipelines

NodeWrapper

- Interfaces
- Algorithms

architecture

- Interfaces
- Algorithms
<table>
<thead>
<tr>
<th>Neuroimaging Pipelines</th>
<th>architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td>Pipeline</td>
</tr>
<tr>
<td>Algorithms</td>
<td>NodeWrapper</td>
</tr>
<tr>
<td></td>
<td>Interfaces</td>
</tr>
<tr>
<td></td>
<td>Algorithms</td>
</tr>
<tr>
<td>Neuroimaging Pypelines</td>
<td>the atoms</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------</td>
</tr>
</tbody>
</table>

# Interfaces + Algorithms

- Provides wrappers/interfaces to call external software (e.g., SPM, FSL)
- Each process defines its own inputs and outputs
- Provides information about the execution of the underlying process
Neuroimaging Pipelines

Interfaces + Algorithms

- Provides wrappers/interfaces to call external software (e.g., SPM, FSL)
- Each process defines its own inputs and outputs
- Provides information about the execution of the underlying process

<table>
<thead>
<tr>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- CommandLine</td>
</tr>
<tr>
<td>-- MatlabCommandLine</td>
</tr>
</tbody>
</table>

the atoms
Example. Interfaces

```python
import nipype.interfaces.fsl as fsl
mybet = fsl.Bet(infile='foo.nii', outfile='bar.nii')
result = mybet.run()
```

```python
import nipype.interfaces.fsl as fsl
mybet = fsl.Bet()
mybet.inputs.infile = 'foo.nii'
mybet.inputs.outfile = 'bar.nii'
result = mybet.run()
```

```python
import nipype.interfaces.fsl as fsl
mybet = fsl.Bet()
result = mybet.run(infile='foo.nii', outfile='bar.nii', frac=0.5)
```

```python
import nipype.interfaces.fsl as fsl
realigner = fsl.McFlirt()
realigner.infile='timeseries4D.nii'
result = realigner.run()
```

```python
import nipype.interfaces.spm as spm
from glob import glob
allepi = glob('epi*.nii') # this will return an unsorted list
allepi.sort()
realigner = spm.Realign()
realigner.inputs.infile = allepi
result = realigner.run()
```
| Neuroimaging Pipelines | the pipeline |
Optimized processing

- Motion Correction
- Coregistration
- Normalization
- Smoothing
- Model Specification
- Model Estimation
Neuroimaging Pipelines
Neuroimaging Pipelines

- Processing pipeline represented as a directed acyclic graph (DAG)
- Leverages graph algorithms for scheduling
- Each process defines its own inputs and outputs and wraps external software (SPM, FSL)
- Separates data specification and code execution
- Can iterate pipeline-segment over parameterizations of modules
Neuroimaging Pypelines

c. Use `nipyte.interfaces.spm.Realign` for motion correction and register all images to the mean image.
```python
realign = nw.NodeWrapper(interface=spm.Realign(), diskbased=True)
realign.inputs.register_to_mean = True
```

e. Use `nipyte.interfaces.fsl.Bet` for skull strip structural images.
```python
skullstrip = nw.NodeWrapper(interface=fsl.Bet(), diskbased=True)
skullstrip.inputs.mask = True
```

h.2. Use `nipyte.interfaces.fs.Smooth` to smooth the functional data.
```python
smooth = nw.NodeWrapper(interface=fs.Smooth(), diskbased=True)
smooth.inputs.surface_fwhm = 5
smooth.inputs.vol_fwhm = 6
smooth.iterfield = ['sourcefile']
```

```python
modelspec = nw.NodeWrapper(interface=model.SpecifyModel(), diskbased=True)
modelspec.inputs.concatenate_runs = True
modelspec.inputs.input_units = 'secs'
modelspec.inputs.output_units = 'secs'
modelspec.inputs.time_repetition = 3.
modelspec.inputs.high_pass_filter_cutoff = 120
```
Neuroimaging Pipelines

the wrapper

```
  c. Use :class:`nipype.interfaces.spm.Realign` for motion correction
  and register all images to the mean image.
  
  realign = nw.NodeWrapper(interface=spm.Realign(), diskbased=True)
  realign.inputs.register_to_mean = True
```

```
  e. Use :class:`nipype.interfaces.fsl.Bet` for skull strip
  structural images.
  
  skullstrip = nw.NodeWrapper(interface=fsl.Bet(), diskbased=True)
  skullstrip.inputs.mask = True
```

```
  h.2. Use :class:`nipype.interfaces.fs.Smooth` to smooth the
  functional data.
  
  smooth = nw.NodeWrapper(interface=fs.Smooth(), diskbased=True)
  smooth.inputs.surface_fwhm = 5
  smooth.inputs.vol_fwhm = 6
  smooth.iterfield = ['sourcefile']
```

```
  modelspec = nw.NodeWrapper(interface=model.SpecifyModel(), diskbased=True)
  modelspec.inputs.concatenate_runs = True
  modelspec.inputs.input_units = 'secs'
  modelspec.inputs.output_units = 'secs'
  modelspec.inputs.time_repetition = 3.
  modelspec.inputs.high_pass_filter_cutoff = 120
```
11pipeline.connect(
    [(datasource, realign, [(‘func’, ‘infile’)]),
     # register mean functional to subject surface
     (datasource, surfregister, [(‘subject_id’, ‘subject_id’)]),
     (realign, surfregister, [(‘mean_image’, ‘sourcefile’)]),
     # smooth using freesurfer’s mixed-mode smoothing
     (surfregister, smooth, [(‘outregfile’, ‘regfile’)]),
     (realign, smooth, [(‘realigned_files’, ‘sourcefile’)]),
     # generate a mask
     (smooth, skullstrip, [(‘outfile’, pickone), ‘infile’])],
    # find outliers
    (realign, art, [(‘realignment_parameters’, ‘realignment_parameters’),
                    (‘realigned_files’, ‘realigned_files’)]),
    (skullstrip, art, [(‘maskfile’, ‘mask_file’)]),
    # design the model
    (datasource, modelspec, [(‘subject_id’, ‘subject_id’)]),
    (realign, modelspec, [(‘realignment_parameters’, ‘realignment_parameters’)]),
    (smooth, modelspec, [(‘outfile’, ‘functional_runs’)]),
    (art, modelspec, [(‘outlier_files’, ‘outlier_files’)]),
    # generate the SPM design matrix
    (modelspec, level1design, [(‘session_info’, ‘session_info’)]),
    (skullstrip, level1design, [(‘maskfile’, ‘mask_image’)]),
    # estimate the model parameters
    (level1design, level1estimate, [(‘spm_mat_file’, ‘spm_design_file’)]),
    # evaluate the contrasts
    (level1estimate, contrastestimate, [(‘spm_mat_file’, ‘spm_mat_file’),
                                         (‘beta_images’, ‘beta_images’),
                                         (‘residual_image’, ‘residual_image’),
                                         (‘RPVimage’, ‘RPVimage’)])
)
}
Neuroimaging Pypelines
<table>
<thead>
<tr>
<th>Neuroimaging Pipelines</th>
<th>challenges</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Neuroimaging Pypelines challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interoperability</strong></td>
</tr>
<tr>
<td>- Translate data formats. Ideally no conversion node</td>
</tr>
<tr>
<td>- Easily change interfaces with similar functionality</td>
</tr>
</tbody>
</table>
Neuroimaging Pipelines | challenges

Interoperability
- Translate data formats. Ideally no conversion node
- Easily change interfaces with similar functionality

Usability
- There are still python components in scripts
- Syntax needs to be semantically cleaner
- A more visual interface is needed
Neuroimaging Pipelines

**Challenges**

**Interoperability**
- Translate data formats. Ideally no conversion node
- Easily change interfaces with similar functionality

**Usability**
- There are still python components in scripts
- Syntax needs to be semantically cleaner
- A more visual interface is needed

**Debugging**
- What happens when things break?
- Are we responsible for FSL support?
- How can we help users ask the right questions
## Neuroimaging Pipelines Challenges

| Interoperability                  | - Translate data formats. Ideally no conversion node  
<table>
<thead>
<tr>
<th></th>
<th>- Easily change interfaces with similar functionality</th>
</tr>
</thead>
</table>
| Usability                         | - There are still python components in scripts  
|                                  | - Syntax needs to be semantically cleaner  
|                                  | - A more visual interface is needed  |
| Debugging                         | - What happens when things break?  
|                                  | - Are we responsible for FSL support?  
|                                  | - How can we help users ask the right questions  |
| Documentation                     | - Using Sphinx is a good start  
<p>|                                  | - How do maintain it as external executables change?  |</p>
<table>
<thead>
<tr>
<th>Neuroimaging Pypelines</th>
<th>challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interoperability</strong></td>
<td>- Translate data formats. Ideally no conversion node</td>
</tr>
<tr>
<td></td>
<td>- Easily change interfaces with similar functionality</td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td>- There are still python components in scripts</td>
</tr>
<tr>
<td></td>
<td>- Syntax needs to be semantically cleaner</td>
</tr>
<tr>
<td></td>
<td>- A more visual interface is needed</td>
</tr>
<tr>
<td><strong>Debugging</strong></td>
<td>- What happens when things break?</td>
</tr>
<tr>
<td></td>
<td>- Are we responsible for FSL support?</td>
</tr>
<tr>
<td></td>
<td>- How can we help users ask the right questions</td>
</tr>
<tr>
<td><strong>Documentation</strong></td>
<td>- Using Sphinx is a good start</td>
</tr>
<tr>
<td></td>
<td>- How do maintain it as external executables change?</td>
</tr>
<tr>
<td><strong>Maintenance</strong></td>
<td>- How do we maintain available interfaces?</td>
</tr>
<tr>
<td></td>
<td>- How do we handle deprecation and addition?</td>
</tr>
<tr>
<td>Neuroimaging Pipelines</td>
<td><strong>challenges</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
</tr>
</tbody>
</table>
| **Interoperability**   | - Translate data formats. Ideally no conversion node  
|                        | - Easily change interfaces with similar functionality |
| **Usability**          | - There are still python components in scripts  
|                        | - Syntax needs to be semantically cleaner  
|                        | - A more visual interface is needed |
| **Debugging**          | - What happens when things break?  
|                        | - Are we responsible for FSL support?  
|                        | - How can we help users ask the right questions |
| **Documentation**      | - Using Sphinx is a good start  
|                        | - How do maintain it as external executables change? |
| **Maintenance**        | - How do we maintain available interfaces?  
|                        | - How do we handle deprecation and addition? |
| **Dependence**         | - Reliance on third party software  
|                        | - What if they change delivery platform? |
Neuroimaging Pipelines

future

technical

• More interfaces
• XNAT compatibility through pyxnat
• Traited/trait-like input checking
• Web interface
• Amazon web-services compatibility

social

• Publications adding nipype scripts/graphs as supplementary material
• Collaborative development
• Self-sustaining
Thanks for the great link...I think one of the things Satra has been pointing out that we need is:

"The design suite provides a visual scripting application for authoring and sharing workflows and preparing the components that are to be incorporated as executable steps. The aim is to shield the author from the complexities of the underlying applications and enable the author to design and understand workflows without recourse to commissioning specialist and specific applications or hiring software engineers."

I think this is a good goal for the project, but we need to stabilize the infrastructure before we jump on this...

In the short term, maybe what we can shoot for is:

"Hence there is significant benefit in establishing shared collections of workflows that contain standard processing pipelines for immediate reuse or for repurposing in whole or in part. These aggregations of expertise and resources can help propagate techniques and best practices."

email from cindee madison, nipy(pe) developer

• Generalizable framework for neuroimaging analysis
• Integrates several neuroimaging software
• Python-based (easy to learn, easy to use)
• Pedagogical (you should know what was done)
• Engages the community (opensource)