NiBetaSeries

Release 0.1.0

Dec 14, 2018

Contents

CHAPTER 1

NiBetaSeries

docs	
tests	
package	

NiBetaSeries is BIDS compatible application that calculates betaseries correlations. In brief, a beta coefficient is calculated for each trial (or event) resulting in a series of betas that can be used to correlate regions of interest with each other.

NiBetaSeries takes preprocessed data as input that satisfy the BIDS deriviatives specification. In practical terms, NiBetaSeries uses the output of fmriprep, a great BIDS compatible preprocessing tool. NiBetaSeries requires the input and the atlas to already be in MNI space since currently no transformations are applied to the data. You can use any arbitrary atlas as long as it is in MNI space (the same space as the preprocessed data).

With NiBetaSeries you can receive:

- betaseries images (TODO)
- correlation matrices

This is a very young project that still needs some tender loving care to grow. That's where you fit in! If you would like to contribute, please read our code of conduct and contributing page.

This project heavily leverages nipype, nilearn, pybids, and nistats for development. Please check out their pages and support the developers.

- Free software: MIT license

1.1 Installation

```
pip install nibetaseries
```

1.2 Documentation

https://NiBetaSeries.readthedocs.io/

If you're interested in contributing to this project, here are some guidelines for contributing. Another good place to start is by checking out the open issues.

1.3 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append`
	`tox`

Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

Betaseries

2.1 Introduction

Betaseries track the trial-to-trial modelled hemodynamic fluctuations that occur within task functional magnetic resonance imaging (task fMRI). This fills an important analytical gap between measuring hemodynamic fluctuations in resting state fMRI and measuring regional activations via cognitive subtraction in task fMRI.

2.2 Relationship to Resting State

Betaseries is similar to resting state because the same analytical strategies applied to resting state data can obstensibly be applied to betaseries. At the core of both resting state and betaseries we are working with a list of numbers at each voxel. We can correlate, estimate regional homogeniety, perform independent components analysis, or conduct a number of different analyses with these voxels. However, betaseries deviates from resting state in two important ways. First, you can do cognitive subtraction using betaseries. Second, interpretation of the lists of numbers are different for resting state and betaseries. Resting state measures the unmodelled hemodynamic fluctuations that occur without explicit stimuli. Betaseries, on the other hand, measures the modelled hemodynamic fluctuations that occur in response to an explicit stimulus. Both resting state and betaseries may measure intrinsic connectivity, but betaseries may also measure the task evoked connectivity (e.g. connectivity between regions that is increased during some cognitive process).

2.3 Relationship to Traditional Task Analysis

Betaseries is also similar to traditional task analysis because cognitive subtraction can be used in both. As with resting state, betaseries deviates from traditional task analysis in several important ways. Say we are interested in observing how the brain responds to faces versus houses. The experimenter has a timestamp of exactly when and how long a face or house is presented. That timestamp information is typically convolved with a hemodynamic response function (HRF) to represent how the brain stereotypically responds to any stimulus resulting in a model of how we expect the brain to respond to places and/or faces. This is where traditional task analysis and betaseries diverge. In traditional task analysis all the face trials are estimated at once, giving one summary measure for how strongly each voxel was activated (same for house trials). The experimenter can subtract the summary measure of faces from houses to see which voxels are more responsive to houses relative to faces (i.e. cognitive subtraction). In betaseries each trial is estimated separately each voxel has as many estimates at there are trials (which can be labelled as either face or house trials). The experimenter can now reduce the series of estimates (a betaseries) for each voxel into a summary measure such as a correlation with region(s) of interest. The correlation map for faces can be subtracted from houses, giving voxels that are more correlated with the region of interest for houses relative to faces. Whereas traditional task analysis treats the variance of brain responses between trials of the same type (e.g. face or house) as noise, betaseries leverages this variance to make conclusions about which brain regions may communicate with each other during a particular trial type (e.g. faces or houses).

2.4 Summary

Betaseries is not in opposition to resting state or traditional task analysis, the methods are complementary. For example, network parcelations derived from resting state data can be used on betaseries data to ascertain if the networks observed in resting state follow a similar pattern with betaseries. Additionally, regions determined from traditional task analysis can be used as regions of interest for betaseries analysis. Betaseries straddles the line between traditional task analysis and resting state, observing task data through a network lens.

2.5 Conceptual Background

Jesse Rissman *[Rissman2004]* was the first to publish on betaseries correlations describing their usage in the context of a working memory task. In this task, participants saw a cue, a delay, and a probe, all occurring close in time. A cue was presented for one second, a delay occurred for seven seconds, and a probe was presented for one second. Given the HRF takes approximately six seconds to reach its peak, and generally takes over 20 seconds to completely resolve, we can begin to see a problem. The events within the trials occur too close to each other to discern what brain responses are related to encoding the cue, the delay, or the probe. To discern how the activated brain regions form networks, Rissman conceptualized betaseries correlations. Instead of having a single regressor to describe all the cue events, a single regressor for all the delay events, and a single regressor for all the probe events (as is done in traditional task analysis), there is an individual regressor for every event in the experiment. For example, if your experiment has 40 trials, each with a cue, delay, and probe event, the model will have a total of 120 regressors, fitting a beta estimate for each trial. Complete this process for each trial of a given event type (e.g. cue), at the end you will have `4D` volume where each volume represents the beta estimates for a particular trial, and each voxel represents a specific beta estimate.

Having one regressor per trial in a single model is known as least squares all. This method, however, has limitations in the context of fast event related designs. Since each trial has its own regressor, trials that occur very close in time are colinear (e.g. are very overlapping). Jeanette Mumford et al. *[Mumford2012]* investigated this issue in the context of image classification. In this article, Mumford introduces another modelling strategy known as least squares separate. In this modelling strategy, instead of having one GLM with a regressor per trial, least squares separate implements a GLM per trial with two regressors: 1) one for the trial of interest, and 2) one for every other trial in the experiment. This process reduces the colinearity of the regressors and creates a more valid estimate of how each regressor fits the data.

2.6 Math Background

$Y = X\beta + \epsilon$(2.1)

The above equation is the General Linear Model (GLM) presented using matrix notation. Y represents the time-series we are attempting to explain. The β assumes any value that minimizes the squared error between the modeled data and the actual data, Y. Finally, ϵ (epsilon) refers to the error that is not captured by the model. Within a GLM, trial-to-trial betas may be averaged for a given trial type and the variance is treated as noise. However, those trial-to-trial fluctuations may also contain important information the typical GLM will ignore/penalize. With a couple modifications to the above equation, we arrive at calculating a betaseries.

$$Y = X_1\beta_1 + X_2\beta_2 + ... + X_n\beta_n + \epsilon (2.2)$$

With the betaseries equation, a beta is estimated for every trial, instead of for each trial type (or whatever logical grouping). This gives us the ability to align all the trial betas from a single trial type into a list (i.e. series). This operation is completed for all voxels, giving us as many lists of betas as there are voxels in the data. Essentially, we are given a `4-D` dataset where the fourth dimension represents trial number instead of time (as the fourth dimension is represented in resting state). And analogous to resting state data, we can perform correlations between the voxels to discern which voxels (or which aggregation of voxels) synchronize best with other voxels.

There is one final concept to cover in order to understand how the betas are estimated in `NiBetaSeries`. You can model individual betas using a couple different strategies; with a least squares all (LSA) approximation which the equation above represents, and a least squares separate (LSS) approximation in which each trial is given it's own GLM. The advantage of LSS comes from reducing the colinearity between closely spaced trials. In LSA, if trials occurred close in time then it would be difficult to model whether the fluctuations should be attributed to one trial or the other. LSS reduces this ambiguity by only having two regressors: one for the trial of interest and another for every other trial. This reduces the colinearity between regressors and makes each beta estimate more reliable.

```
for trial, beta in zip(trials, betas):
    data = X[trial] * beta + error
```

This python psuedocode demonstrates LSS where each trial is given it's own model.

2.7 Relevent Software

BASCO (BetA Series COrrelations) is a matlab program that also performs betaseries correlations

2.8 Other Relevant Readings

- [Cisler2014]
- [Gottlich2015]
- [Abdulrahman2016]

2.9 References

CHAPTER 3

Installation

At the command line:

```
pip install nibetaseries
```

CHAPTER 4

Usage

4.1 Command-Line Arguments

NiBetaSeries BIDS arguments

```
usage: nibs [-h] [-v] [-sm SMOOTHING_KERNEL] [-lp LOW_PASS]
            [-c CONFOUNDS [CONFOUNDS ...]] [-w WORK_DIR]
            [--participant_label PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]
            [--session_label SESSION_LABEL] [-t TASK_LABEL]
            [--run_label RUN_LABEL] [-sp {MNI152NLin2009cAsym}]
            [--variant_label VARIANT_LABEL] [--exclude_variant_label]
            [--hrf_model {glover,spm,fir,glover + derivative,glover + derivative +␣
↪dispersion,spm + derivativespm + derivative + dispersion}]
            [-a ATLAS_IMG] -l ATLAS_LUT [--graph]
            bids_dir derivatives_pipeline output_dir {participant,group}
```

4.1.1 Positional Arguments

bids_dir	The directory with the input dataset formatted according to the BIDS standard.
derivatives_pipeline	The pipeline that contains minimally preprocessed img, brainmask, and confounds.tsv
output_dir	The directory where the output directory and files should be stored. If you are running group level analysis this folder should be prepopulated with the results of theparticipant level analysis.
analysis_level	Possible choices: participant, group Level of the analysis that will be performed Multiple participant level analyses can be run independently (in parallel) using the same output_dir

4.1.2 Named Arguments

-v, --version	show program's version number and exit
--participant_label	The label(s) of the participant(s) that should be analyzed. The label corresponds to sub-<participant_label> from the BIDS spec (so it does not include "sub-"). If this parameter is not provided all subjects should be analyzed. Multiple participants can be specified with a space separated list.

4.1.3 Options for preprocessing

-sm, --smoothing_kernel	select a smoothing kernel (mm)
-lp, --low_pass	low pass filter (Hz)
-c, --confounds	The confound column names that are to be included in nuisance regression. write the confounds you wish to include separated by a space
-w, --work_dir	directory where temporary files are stored

4.1.4 Options for selecting images

--session_label	select a session to analyze
-t, --task_label	select a specific task to be processed
--run_label	select a run to analyze
-sp, --space_label	Possible choices: MNI152NLin2009cAsym select a bold derivative in a specific space to be used
--variant_label	select a variant bold to process
--exclude_variant_label	exclude the variant from FMRIPREP

4.1.5 Options for processing beta_series

--hrf_model	Possible choices: glover, spm, fir, glover + derivative, glover + derivative + dispersion, spm + derivativespm + derivative + dispersion convolve your regressors with one of the following hemodynamic response functions
-a, --atlas-img	input atlas nifti where each voxel within a "region" is labeled with the same integer and there is a unique integer associated with each region of interest
-l, --atlas-lut	atlas look up table (tsv) formatted with the columns: index, regions which correspond to the regions in the nifti file specified by –atlas-img

4.1.6 misc options

--graph	generates a graph png of the workflow

CHAPTER 5

Workflows

5.1 Participant Workflow

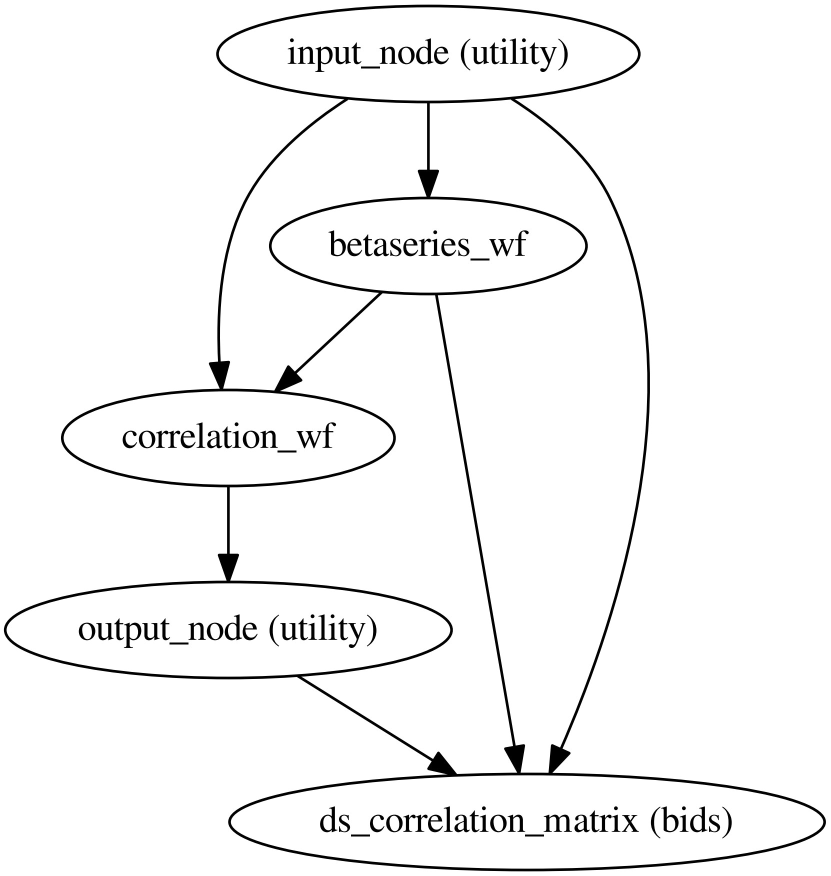
The general workflow for a participant models the betaseries for each trial type for each bold file associated with the participant. Then betas within a region of interest are based off a parcellation are averaged together. This occurs as many times as there are trials for that particular trial type, resulting in a psuedo-timeseries (e.g. each point in "time" represents an occurrence of that trial). All the psuedo time-series within a trial type are correlated with each other, resulting in a final correlation (adjacency) matrix.

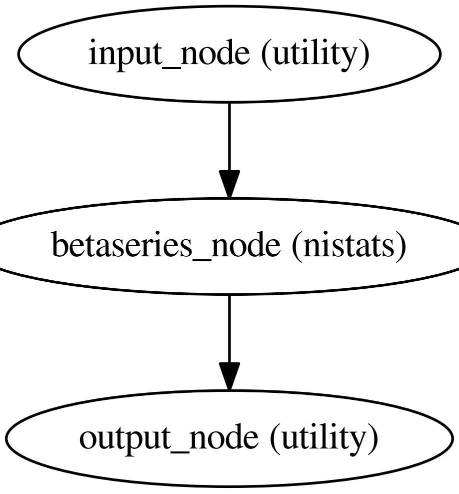
5.2 BetaSeries Workflow

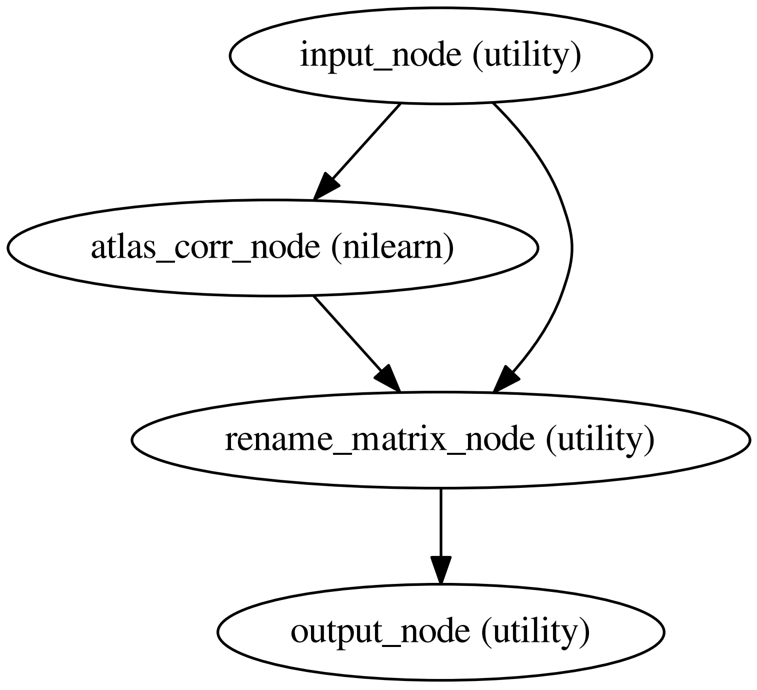
The bold file is temporally filtered by nilearn (high pass and/or low pass) before being passed into nistats for modelling by least squares separate.

5.3 Correlation Workflow

The betaseries file has signal averaged across trials within a region defined by an atlas parcellation. After signal extraction has occurred for all regions, the signals are all correlated with each other to generate a correlation matrix.


input_node (utility)
betaseries_wf
correlation_wf
output_node (utility)
ds_correlation_matrix (bids)


input_node (utility)
betaseries_node (nistats)
output_node (utility)


input_node (utility)
atlas_corr_node (nilearn)
rename_matrix_node (utility)
output_node (utility)

CHAPTER 6

Reference

6.1 nibetaseries

CHAPTER 7

Contributing to NiBetaSeries

Welcome to the NiBetaSeries repository!

We're so excited you're here and want to contribute.

NiBetaSeries calculates betaseries correlations using python! We hope that these guidelines are designed to make it as easy as possible to contribute to NiBetaSeries and the broader Brain Imaging Data Structure (BIDS) community. If you have any questions that aren't discussed below, please let us know through one of the many ways to *get in touch*.

7.1 Table of contents

Been here before? Already know what you're looking for in this guide? Jump to the following sections:

7.2 Joining the BIDS community

BIDS - the Brain Imaging Data Structure - is a growing community of neuroimaging enthusiasts, and we want to make our resources accessible to and engaging for as many researchers as possible. NiBetaSeries will hopefully play a small part towards introducing people to BIDS and help the broader community.

We therefore require that all contributions **adhere to our** Code of Conduct.

How do you know that you're a member of the BIDS community? You're here! You know that BIDS exists! You're officially a member of the community. It's THAT easy! Welcome!

7.3 Check out the BIDS Starter Kit

If you're new to BIDS make sure to check out the amazing BIDS Starter Kit.

7.4 Get in touch

There are lots of ways to get in touch with the team maintaining NiBetaSeries if you have general questions about the BIDS ecosystem. For specific questions about NiBetaSeries, please see the practical guide (Currently the main contact is James Kent)

- Message jdkent on the brainhack slack
- Click here for an invite to the slack workspace.
- The BIDS mailing list
- **Via the Neurostars forum.**
    - **This is our preferred way to answer questions so that others who have similar questions can benefit too!** Even if your question is not well-defined, just post what you have so far and we will be able to point you in the right direction!

7.5 Contributing through GitHub

git is a really useful tool for version control. GitHub sits on top of git and supports collaborative and distributed working.

We know that it can be daunting to start using git and GitHub if you haven't worked with them in the past, but the NibetaSeries maintainer(s) are here to help you figure out any of the jargon or confusing instructions you encounter!

In order to contribute via GitHub you'll need to set up a free account and sign in. Here are some instructions to help you get going. Remember that you can ask us any questions you need to along the way.

7.6 Writing in markdown

GitHub has a helpful page on getting started with writing and formatting on GitHub.

Most of the writing that you'll do in github will be in Markdown. You can think of Markdown as a few little symbols around your text that will allow GitHub to render the text with a little bit of formatting. For example you could write words as bold (`**bold**`), or in italics (`*italics*`), or as a link to another webpage.

7.7 Writing in ReStructuredText

This file and the rest of the documentation files in this project are written using ReStructuredText. This is another markup language that interfaces with sphinx, a documentation generator. Sphinx is used on ReadTheDocs, a documentation hosting service. Putting it all together, ReadTheDocs is an online documentation hosting service that uses sphinx, and sphinx is a documentation generation service that uses ReStructuredText to format the content. What a mouthfull!

7.8 Where to start: issue labels

The list of labels for current issues can be found here and includes:

- If you feel that you can contribute to one of these issues, we especially encourage you to do so!
- There are *lots of ways to ask questions* but opening an issue is a great way to start a conversation and get your answer. Ideally, we'll close it out by adding the answer to the docs!
- *These issues are particularly appropriate if it is your first contribution to NiBetaSeries, or to GitHub overall.*

  If you're not sure about how to go about contributing, these are good places to start. You'll be mentored through the process by the maintainers team. If you're a seasoned contributor, please select a different issue to work from and keep these available for the newer and potentially more anxious team members.
- If you want to ask for something new, please try to make sure that your request is distinct from any others that are already in the queue (or part of NiBetaSeries!). If you find one that's similar but there are subtle differences please reference the other enhancement in your issue.
- These are usually really great issues to help out with: our goal is to make it easy to understand BIDS without having to ask anyone any questions! Documentation is the ultimate solution
- *These issues are reporting a problem or a mistake in the project.*

  The more details you can provide the better! If you know how to fix the bug, please open an issue first and then *submit a pull request*

  We like to model best practice, so NiBetaSeries itself is managed through these issues. We may occasionally have some to coordinate some logistics.

7.9 Making a change with a pull request

We appreciate all contributions to NiBetaSeries. **THANK YOU** for helping us build this useful resource.

Remember that if you're adding information to the *wiki* you **don't need to submit a pull request**. You can just log into GitHub, navigate to the wiki and click the **edit** button.

If you're updating the *code*, the following steps are a guide to help you contribute in a way that will be easy for everyone to review and accept with ease.

7.9.1 1. Comment on an existing issue or open a new issue referencing your addition

This allows other members of the NiBetaSeries team to confirm that you aren't overlapping with work that's currently underway and that everyone is on the same page with the goal of the work you're going to carry out.

This blog is a nice explanation of why putting this work in up front is so useful to everyone involved.

7.9.2 2. Fork the NiBetaSeries repository to your profile

This is now your own unique copy of NiBetaSeries. Changes here won't affect anyone else's work, so it's a safe space to explore edits to the code!

Make sure to keep your fork up to date with the master repository, otherwise you can end up with lots of dreaded merge conflicts.

7.9.3 3. Clone your forked NiBetaSeries to your work machine

Now that you have your own repository to explore you should clone it to your work machine so you can easily edit the files:

```
# clone the repository
git clone https://github.com/YOUR-USERNAME/NiBetaSeries
# change directories into NiBetaSeries
cd NiBetaSeries
# add the upstream repository (i.e. https://github.com/HBClab/NiBetaSeries)
git remote add upstream https://github.com/HBClab/NiBetaSeries
```

7.9.4 4. Make the changes you've discussed

Try to keep the changes focused. If you submit a large amount of work in all in one go it will be much more work for whomever is reviewing your pull request. Help them help you

This project requires you to "branch out" and make new branch and a new issue to go with it if the issue doesn't already exist.

Example:

```
# create the branch on which you will make your issues
git checkout -b your_issue_branch
```

7.9.5 5. Run the tests

When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

If the checks fail and you know what went wrong, make the change and run tox again. If you are not sure what the error is, go ahead to step 6.

Note: tox doesn't work on everyone's machine, so don't worry about getting the tests working on your machine for now

7.9.6 6. Add/Commit/Push the changes to the NiBetaSeries repository

Once you've made the changes on your branch you are ready to 1) add the files to be tracked by git 2) commit the files to take a snapshot of the branch, and 3) push the changes to your forked repository. You can do complete the add/commit/push process following this github help page.

7.9.7 7. Submit a pull request

A member of the NiBetaSeries team will review your changes to confirm that they can be merged into the main codebase.

A review will probably consist of a few questions to help clarify the work you've done. Keep an eye on your github notifications and be prepared to join in that conversation.

You can update your fork of the NiBetaSeries repository and the pull request will automatically update with those changes. You don't need to submit a new pull request when you make a change in response to a review.

GitHub has a nice introduction to the pull request workflow, but please *get in touch* if you have any questions.

7.9.8 NiBetaSeries coding style guide

Whenever possible, instances of Nodes and Workflows should use the same names as the variables they are assigned to. This makes it easier to relate the content of the working directory to the code that generated it when debugging.

Workflow variables should end in *_wf* to indicate that they refer to Workflows and not Nodes. For instance, a workflow whose basename is *myworkflow* might be defined as follows:

```
from nipype.pipeline import engine as pe

myworkflow_wf = pe.Workflow(name='myworkflow_wf')
```

If a workflow is generated by a function, the name of the function should take the form *init_<basename>_wf*:

```
def init_myworkflow_wf(name='myworkflow_wf):
    workflow = pe.Workflow(name=name)
    ...
    return workflow

myworkflow_wf = init_workflow_wf(name='myworkflow_wf')
```

If multiple instances of the same workflow might be instantiated in the same namespace, the workflow names and variables should include either a numeric identifier or a one-word description, such as:

```
myworkflow0_wf = init_workflow_wf(name='myworkflow0_wf')
myworkflow1_wf = init_workflow_wf(name='myworkflow1_wf')

# or

myworkflow_lh_wf = init_workflow_wf(name='myworkflow_lh_wf')
myworkflow_rh_wf = init_workflow_wf(name='myworkflow_rh_wf')
```

7.10 Recognizing contributions

BIDS follows the all-contributors specification, so we welcome and recognize all contributions from documentation to testing to code development. You can see a list of current contributors in the BIDS specification.

7.11 Thank you!

You're awesome.

— *Based on contributing guidelines from the* STEMMRoleModels *project.*

CHAPTER 8

Authors

- James Kent - https://github.com/jdkent

CHAPTER 9

0.2.1 (November 13, 2018)

Large thanks to everyone at neurohackademy that helped make this a reality. This release is still a bit premature because I'm testing out my workflow for making releases.

- [ENH] Add link to Zenodo DOI (#57) @kdestasio
- [ENH] run versioneer install (#60) @jdkent
- [FIX] connect derivative outputs (#61) @jdkent
- [FIX] add CODEOWNERS file (#63) @jdkent
- [FIX] Fix pull request template (#65) @kristianeschenburg
- [ENH]Update CONTRIBUTING.rst (#66) @PeerHerholz
- [FIX] ignore sourcedata and derivatives directories in layout (#69) @jdkent
- [DOC] Added zenodo file (#70) @ctoroserey
- [FIX] file logic (#71) @jdkent
- [FIX] confound removal (#72) @jdkent
- [FIX] Find metadata (#74) @jdkent
- [FIX] various fixes for a real dataset (#75) @jdkent
- [ENH] allow confounds to be none (#76) @jdkent
- [ENH] Reword docs (#77) @jdkent
- [TST] Add more tests (#78) @jdkent
- [MGT] simplify and create deployment (#79) @jdkent

CHAPTER 10

0.2.0 (November 13, 2018)

- [MGT] simplify and create deployment (#79)
- [TST] Add more tests (#78)
- [ENH] Reword docs (#77)
- [ENH]: allow confounds to be none (#76)
- various fixes for a real dataset (#75)
- [FIX]: Find metadata (#74)
- [FIX] confound removal (#72)
- [WIP, FIX]: file logic (#71)
- [DOC] Added zenodo file (#70)
- [FIX]: ignore sourcedata and derivatives directories in layout (#69)
- Update CONTRIBUTING.rst (#66)
- Fix pull request template (#65)
- FIX: add CODEOWNERS file (#63)
- FIX: connect derivative outputs (#61)
- run versioneer install (#60)
- Fix issue #29: Add link to Zenodo DOI (#57)
- Fix issue #45: conform colors of labels (#56)
- fix links in readme.rst (#55)
- Added code of conduct (#53)
- Add link to contributing in README (#52)
- removed acknowledgments section of pull request template (#50)
- [TST]: Add functional test (#49)

- [FIX]: remove references to bootstrap (#48)
- FIX: test remove base .travis.yml (#47)
- removed data directory (#40)
- Add pull request template (#41)
- Update issue templates (#44)
- Update contributing (#43)
- README (where's the beef?) (#37)
- change jdkent to HBClab (#38)
- [FIX]: pass tests (#14)
- [ENH]: improve docs (#13)
- add documentation (#11)
- FIX: add graph (#10)
- Refactor NiBetaSeries (#9)
- Refactor (#2)

CHAPTER 11

0.1.0 (2018-06-08)

- First release on PyPI.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Bibliography

[Rissman2004] https://www.ncbi.nlm.nih.gov/pubmed/15488425

[Mumford2012] https://www.ncbi.nlm.nih.gov/pubmed/21924359

[Cisler2014] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4019671/

[Gottlich2015] https://www.frontiersin.org/articles/10.3389/fnsys.2015.00126/full

[Abdulrahman2016] https://www.ncbi.nlm.nih.gov/pubmed/26549299

Python Module Index

Index