PaDRe Documentation

Release 0.5

Bill Gross

Jan 30, 2018

Contents

1 The Contents **3**
1.1 How to use this library . . . 3
1.2 Examples of usage . . . 4
1.3 Usage tutorial . . . 5
1.4 padre - PaDRe Library Functions . . . 6
1.5 padre.subject - Subject Objects . . . 6
1.6 padre_buddy.py - Your friendly command-line helper . . . 6
1.7 Organization of the Data . . . 8
1.8 Installation instructions . . . 9

2 Indices and tables **11**

This library provides an organized way to store and analyze imaging data.

The *How to use this library* page explains the basics on how to use the library, and the *padre - PaDRe Library Functions* and *padre.subject - Subject Objects* pages have the documentation for all of the functions you'll need.

If you're interested in the nuts and bolts of the library, *Organization of the Data* explains the internals of the library, and *Installation instructions* explains how to setup a new repository or client machine.

Here are a couple quick examples to get you started, and more full-length examples in *Examples of usage* and *Usage tutorial*:

Tip: Loop through all the subjects and do something to all their datasets:

```
import padre as p

for subject in p.subjects():
        print 'Processing subject %s' % subject
        for dset in subject.dsets():
                do_something_to_a_dset(dset)
```

Find all the subjects for a particular experiment and do something to the functional task datasets:

```
for subject in p.subjects('experiment-name'):
        for dset in subject.dsets('functional-task'):
                do_something_to_a_dset(dset)
```

Find all subjects in an experiment and do something to all the anatomies that were collected in the same session as a functional task:

```
for subject in p.subjects('experiment-name'):
        for session in subject.sessions:
                if len(subject.dsets('functional-task',session=session)):
                        for dset in subject.dsets('anatomy',session=session):
                                do_something_to_an_anatomy(dset)
```

CHAPTER 1

The Contents

1.1 How to use this library

1.1.1 Accessing the data

Recommended Method Since Python is a great language, and the library provides a lot of convenience functions for you, the recommended method for accessing the data is using Python scripts that directly include the library. It's terribly easy. Just import the library in any script, and get going.

```
import padre as p
```

Since it's just a Python library, it can be imported from anywhere, and all of the file references are absolute, so it doesn't matter what folder you are in. Take a look at the functions and examples here for fun things to do with the library.

The Not-Recommended Method If you hate Python, kittens, and love, you could write all of your scripts in some awful language, like C shell + awk, and lookup the location of the data using the command-line script *padre_buddy.py*. The *padre_buddy.py* script is really designed to help you have quick access to the datasets on the command line, and wasn't meant to be a crutch to hold you back from migrating to Python. But, it's there if you need it.

1.1.2 Basic workflow concepts

The raw data is stored within padre, and is accessible from anywhere through the library functions. For efficiency, it's not recommended to copy the raw data out of the repository, but to try to work with it in place. This is pretty easy to do within the library itself, because all files are referred to by absolute file references, so it doesn't matter what directory you are in when you refer to a file. If you need to access the data easily outside the library, the *padre_buddy.py* script should allow you to work with the files without having to copy them.

1.2 Examples of usage

Here are some "real-life" examples of how I would process data. I also use the neural library for a lot of my processing, so you'll see it pop up here as well.

In these examples, my experiment is called "HitHeads" and it has two conditions: "hard" and "soft".

First, preprocess the data:

```python
import padre as p
import neural as nl

# Get me a list of subject objects for my experiment
subjects_list = p.subjects('HitHeads')

# Learning the "default argument" and "named argument" syntax of Python is important
# This line is the same as the previous one:
subjects_list = p.subjects(experiment='HitHeads')

for subj in subjects_list:
        # Loop through the subjects
        anatomy = subj.dsets('anatomy')[0]
        # Since I want to align all of the epis together, I want to collect them in a␣
↪list:
        epis = []
        for condition in ['hard','soft']:
                epis += subj.dsets(condition)
        # If you're a sharp Python programmer, you'll realize I also could have done:
        epis = [x for y in [subj.dsets(a) for a in ['hard','soft']] for x in y]
        # or maybe more readibly...
        epis = nl.flatten([subj.dsets(a) for a in ['hard','soft']])

        # Now we have our anatomy and a list of our epis, so align them!
        nl.afni.align_anat_epi(anatomy,epis)
```

Next, deconvolve:

```python
import padre as p
import neural as nl

# this is the name of a stimfile that we made somewhere else
# for example, neural as has very nice E-Prime file parser!
stimfile_name = lambda dset: '%s-stimfile.1D' % dset

# Same as before, but just a one-liner
for subj in p.subjects('HitHeads'):
        # for the details on the decon function, see the neural library
        decon = nl.afni.Decon()
        # Here's another way to get our epis. Not as "slick",
        # but maybe a little more readible
        decon.input_dsets = subj.dsets('hard') + subj.dsets('soft')
        decon.stim_files = [stimfile_name(dset) for dset in [subj.dsets('hard') +␣
↪subj.dsets('soft')]]
        decon.prefix = "%s-headhit_firstpass" % subj
        decon.run()
```

1.3 Usage tutorial

This is a quick run-through of how you might analyze a dataset. First, we need to import the library:

```
import padre as p
```

1.3.1 Data organization

Here's the organization of our sample data:

Name of experiment Exp1

Types of datasets anatomy, flair, task1, task2

We'll process all the data in a subdirectory of our home directory `~/Exp1`. The datasets are named:

raw data `[subject].run[#].nii.gz`

anatomy `[subject].anat.nii.gz`

stim files `[subject].[task].1D`

1.3.2 Get the subject list

The easiest way to get the list of subjects is to organize them by experiment, and get them by calling `padre.subjects()`:

```
# Return all subjects who have a session labeled with experiment "Exp1"
subjects = p.subjects('Exp1')
# The same thing, using keyword arguments
subjects = p.subjects(experiment='Exp1')
```

If you want to get any subject with who has any runs of the task `task1`, you could use this instead:

```
subjects = p.subjects(label='task1')
```

Or, you could keep a list of subject numbers the old fashion way, and load them individually:

```
subjects = []
with open('subjects.txt') as f:
        for line in f.readlines():
                subj = p.load(line.strip())
                if subj:
                        subjects.append(subj)
```

1.3.3 Preprocess data

Now that we have our list of subjects, we can loop through the data and run some preprocessing functions on them:

```
import neural as nl

for subj in subjects:
        # The dsets function will return the datasets we're interested in.
        # The first argument is the dset label, but you could also specify
        # session parameters like tags or session name
        # The "[0]" needs to be there because dsets() always returns a list
        anatomy = subj.dsets('anatomy')[0]
        # epis now is a list of all of the task1 and task2 datasets
        epis = subj.dsets('task1') + subj.dsets('task2')
        # run the "align_epi_anat.py" script (using the neural library)
        nl.afni.align_epi_anat(anatomy,epis)
```

1.4 padre - PaDRe Library Functions

1.4.1 Global Functions

Several functions are available at the root-level of the library, the most important of which is the `padre.subjects()` function (which is a shortcut to the function `padre.subject.subjects()`)

Once you have the subject objects, you can play with them using the methods in *padre.subject - Subject Objects*

1.4.2 Extra Stuff

1.4.3 Directory and File Locations

1.5 padre.subject - Subject Objects

1.5.1 Maintenance Functions

These functions require write-access to the repository, so should not be used in standard scripts

1.6 padre_buddy.py - Your friendly command-line helper

This script was created as a blending of my desire to make a very easy, user-friendly script, my interest in artificial intelligence, and my loose grip on reality. The result is a sometimes amazing, sometimes broken script.

1.6.1 Usage:

padre_buddy.py [action_word] [direct_objects] [modifiers]

The syntax is very loose. The library tries to match each word to a known concept. It will try to correct for simple spelling errors and has an internal dictionary of synonyms as well.

Actions (currently working):

list Currently the most useful. Try to match a direct object and list all of them that match the modifiers. For example, if the direct object was "dsets" and the modifier was "subject-4", it would print out all of the datasets for "subject-4"

link Will find all datasets matching the given modifiers and create symbolic links within the current working directory. Useful if you want to "play" with the data without moving it out of Padre or using really long filenames

Direct Objects:

- Atlases
- Subjects
- Experiments
- Labels
- Tags
- Sessions
- Dsets
- Metadata
- Edat, Eprime-txt

Example of command line usage:

```
padre_buddy.py list subjects ExperimentName

padre_buddy.py list labels

padre_buddy.py list dsets subject_one task_name
```

Example of script usage (although CSH burns my eyes…):

```
#!/bin/csh

echo 'Running the weird, crazy, padre machine to try to figure out where the heck my␣
↪data is...'

set subject = "11334"
set task = "sdtones"

set dsets = `padre_buddy.py list dsets $task $subject`

if( "$dsets" == "" ) then
        echo 'AHHH! No DATA! '
        echo ''
        echo 'Did you make sure everything looks good in Padre Web? '
        echo '...'
        echo '...'
        exit
endif

echo ''
echo 'Ok... phew. Found the data. Now I can return to the sanity of CSH'
echo ''
echo 'First SDT dataset:'
echo $dsets[1]

echo '...and the second:'
echo $dsets[2]

echo ''
echo 'not necessarily in chronological order'
```

1.7 Organization of the Data

1.7.1 Principles

Important features of how the data is organized:

- The majority of the data tree is read-only, it can be referenced, but not changed in-place
- There is a separate directory to store individual analyses in. These directories are read-write and all derived data go in those directories.
- Because the data is diverse and many subjects have idiosyncrasies, there are no static lists of subjects. You can request lists from the library, meeting whatever criteria you specify.

1.7.2 Directory structure

```
Data/
|-- [subj_id]/
|   |-- [subj_id].json                          # config file, containing␣
↪subject meta-data
|   |
|   |-- raw/                                    # directory with all of the␣
↪raw data, unorganized
|   |-- sessions/                               # data organized by scanning
↪"session",
|       |                                       # as in, one physical visit␣
↪to the scanner
|       |-- [session_name]/
|               |-- [datasets]          # all of the datasets, arbitrarily␣
↪named
|               |-- [scan_sheets]
|               |-- [behavioral files]
|               |-- [other meta]        # other files associated with a␣
↪session
|               |-- ...
|-- ..



Shared/
|-- Atlases/                                    # standard atlas volumes
|   |-- atlases.json                            # file describing the atlases
|   |-- [datasets]
|
|-- Stimfiles/                                  # shared stimfiles (the same␣
↪for each subject)
    |-- stimfiles.json                          # file describing the␣
↪stimfiles
    |-- [files]
```

1.7.3 Subject Objects

Most of the data from the library is returned in the form of *padre.subject - Subject Objects*. These objects contain the subject number, meta-data about the subject, as well as the disk location of all files associated with the subject. The information that is populated into these objects comes from the [subj id].json file in each subject directory. See the *padre.subject - Subject Objects* page for details on this class.

1.8 Installation instructions

1.8.1 User installation

If your library is already setup and you just need to attach your client machine, you just need 3 things:

data store Likely this is a network drive that needs to be mounted on some system directory (e.g., /mnt/server_name or /Network/server_name)

padre library This is just a standard Python library, hosted on PyPi and GitHub. You can install the latest version with "pip install git+https://github.com/azraq27/padre" or the latest release with "pip install padre"

PADRE_ROOT In whatever shell you are using you need to set the environment variable "PADRE_ROOT" to the directory where the data store is. For example, in bash you might put "export PADRE_ROOT /mnt/server/padre" in .bashrc

1.8.2 System installation

CHAPTER 2

Indices and tables

- genindex
- modindex
- search