abagen

Release 0.1.3-doc+0.g2aeab5b.dirty

abagen developers

Jul 23, 2021

CONTENTS

This package provides a Python interface for fetching and working with the Allen Human Brain Atlas (AHBA) microarray expression data.

CHAPTER

ONE

OVERVIEW

In 2013, the Allen Institute for Brain Science released the Allen Human Brain Atlas, a dataset containing microarray expression data collected from six human brains (Hawrylycz et al., 2012) . This dataset has offered an unprecedented opportunity to examine the genetic underpinnings of the human brain, and has already yielded novel insight into e.g., adolescent brain development and functional brain organization.

However, in order to be effectively used in most analyses, the AHBA microarray expression data often needs to be (1) collapsed into regions of interest (e.g., parcels or networks), and (2) combined across donors. While this may potentially seem trivial, there are a number of analytic choices in these steps that can dramatically influence the resulting data and any downstream analyses. Arnatkevičiūte et al., 2019 provided a thorough treatment of this in a recent publication, demonstrating how the techniques and code used to prepare the raw AHBA data have varied widely across published reports. We extended this work in a recent preprint (Markello et al., 2021) to quantify how different processing choices can impact statistical analyses of the AHBA.

The current Python package, `abagen`, aims to provide reproducible workflows for processing and preparing the AHBA microarray expression data for analysis.

CHAPTER

TWO

INSTALLATION REQUIREMENTS

Currently, `abagen` works with Python 3.6+ and requires a few dependencies:

- nibabel
- numpy (>=1.14.0)
- pandas (>=0.25.0), and
- scipy

There are some additional (optional) dependencies you can install to speed up some functions:

- fastparquet, and
- python-snappy

These latter packages are primarily used to facilitate loading the (rather large!) microarray expression dataframes provided by the Allen Institute,

For detailed information on how to install `abagen`, including these dependencies, refer to our installation instructions.

QUICKSTART

At it's core, using `abagen` is as simple as:

```
>>> import abagen
>>> expression = abagen.get_expression_data('myatlas.nii.gz')
```

where `'myatlas.nii.gz'` points to a brain parcellation file.

This function can also be called from the command line with:

```
$ abagen --output-file expression.csv myatlas.nii.gz
```

For more detailed instructions on how to use `abagen` please refer to our user guide!

CHAPTER FOUR

DEVELOPMENT AND GETTING INVOLVED

If you've found a bug, are experiencing a problem, or have a question about using the package, please head on over to our GitHub issues and make a new issue with some information about it! Someone will try and get back to you as quickly as possible, though please note that the primary developer for `abagen` (@rmarkello) is a graduate student so responses make take some time!

If you're interested in getting involved in the project: welcome ! We're thrilled to welcome new contributors. You should start by reading our code of conduct; all activity on `abagen` should adhere to the CoC. After that, take a look at our contributing guidelines so you're familiar with the processes we (generally) try to follow when making changes to the repository! Once you're ready to jump in head on over to our issues to see if there's anything you might like to work on.

CHAPTER FIVE

CITING ABAGEN

For up-to-date instructions on how to cite abagen please refer to our documentation.

CHAPTER SIX

LICENSE INFORMATION

This codebase is licensed under the 3-clause BSD license. The full license can be found in the LICENSE file in the `abagen` distribution.

Reannotated gene information located at `abagen/data/reannotated.csv.gz` and individualized donor parcellations for the Desikan-Killiany atlas located at `abagen/data/native_dk` are taken from Arnatkevičiūte et al., 2018 and are separately licensed under the CC BY 4.0; these data can also be found on figshare.

Corrected MNI coordinates used to match AHBA tissues samples to MNI space located at `abagen/data/corrected_mni_coordinates.csv` are taken from the alleninf package, provided under the 3-clause BSD license.

All microarray expression data is copyrighted under non-commercial reuse policies by the Allen Institute for Brain Science (© 2010 Allen Institute for Brain Science. Allen Human Brain Atlas. Available from: Allen Human Brain Atlas).

All trademarks referenced herein are property of their respective holders.

CHAPTER SEVEN

CONTENTS

7.1 Installation and setup

Note: Using the instructions for the *IO installation* is highly recommended! It may take a little more set-up depending on your operating system, but the benefits during processing are noticeable!

7.1.1 Basic installation

This package requires Python 3.6+. Assuming you have the correct version of Python installed, you can install abagen by opening a terminal and running the following:

```
pip install abagen
```

Alternatively, you can install the most up-to-date version of from GitHub:

```
git clone https://github.com/rmarkello/abagen.git
cd abagen
pip install .
```

Important: If you are going to install the version directly from GitHub make sure that you are using the most up-to-date documentation!

7.1.2 IO installation

The data supplied by the Allen Human Brain Atlas is quite large—on the order of ~4GB for all six donors. Because loading these datasets into memory can be quite time-consuming, abagen has integrated support for parquet and can do some on-the-fly conversions to speed things up. However, using parquet is completely optional, and therefore support for it is not installed when using the "vanilla" installation procedures.

If you would like to enable parquet support, you will need to install some additional dependencies. This can be done using pip:

```
pip install abagen[io]
```

Note: If you are using a Linux OS, you will need to install the `libsnappy-dev` (Debian) or `snappy-devel` (Fedora) package before running the above installation command!

You can also install these extra packages from the GitHub source:

```
git clone https://github.com/rmarkello/abagen.git
cd abagen
pip install .[io]
```

Important: If you are going to install the version directly from GitHub make sure that you are using the most up-to-date documentation!

7.2 What's new

7.2.1 0.1.3 (June 18, 2021)

More bug fixes for (identified thanks to @richardajdear and @Silflame!) as well as some minor QOL updates for getting sample-level information from workflows.

- [REF] Handle atlases w/decimal vox sizes (#197), @rmarkello
- [REF] Re-orient img in *leftify_atlas()* (#193), @rmarkello
- [FIX] Fixes logger error w/reports (#192), @rmarkello
- [FIX] Error with donor-specific atlases and tolerance=0 (#191), @rmarkello
- [REF] Fix coordinates of get_samples_in_mask (#189), @rmarkello

7.2.2 0.1.1 (March 29, 2021)

Small bug fix when *probe_selection='average'* and minor QOL update to *check_atlas()*.

- [ENH] Add geometry/space option to check_atlas (#186), @rmarkello
- [FIX] Fixes 'average' probe selection bug (#185), @rmarkello

7.2.3 0.1 (March 25, 2021)

First minor release! :tada: This update brings some major changes to the codebase, including a massive internal refactoring to allow handling of surface parcellations, new methods for interpolating missing data, and more!

Important: as this is our first official minor release, note that the changes in this release are *not backwards compatible* with previous versions.

- [ENH] Adds interareal similarity thresholding (#184), @rmarkello
- [REF] Allow negative tolerance for surfaces (#183), @rmarkello
- [REF] Updates reports for *missing* param (#182), @rmarkello
- [DOC] Updates master refs to main (#181), @rmarkello

- [ENH] Adds interpolation options for missing data (#180), @rmarkello
- [REF] Updates mouse data cache location (#179), @rmarkello
- [FIX] Bugfix for lr_mirror parameter (#178), @rmarkello
- [ENH] Adds reporting of processing methods (#177), @rmarkello
- [FIX] Nan hemisphere designation (#176), @rmarkello
- [FIX] Addresses bug when *tolerance=0* (#175), @rmarkello
- [TEST] Add py3.8 to azure tests (#174), @rmarkello
- [REF] Constrains centroid matching by atlas_info (#173), @rmarkello
- [REF] Updates CLI to match main workflow (#172), @rmarkello
- [REF] Modifies *lr_mirror* to accept single hemisphere mirroring (#171), @rmarkello
- [ENH] Add *norm_structures* parameter (#170), @rmarkello
- [ENH] Adds ability to handle surface parcellations (#169), @rmarkello
- [TEST] Parametrize *test_resid_dist* (#164), @4lovi4

7.2.4 0.0.8 (January 29, 2021)

Largely internal codebase changes, but an important bug fix for compatibility with newer versions of `pandas`.

Huge thanks to @4lovi4 for all their contributions!

- [FIX] Read_csv error when using pandas >= 1.2.0 (#167), @rmarkello
- [TEST] Change Azure VM image to fix SSL (#165), @rmarkello
- Adds Python 3.8 testing support to CircleCI (#163), @4lovi4
- issue #73 unquote url (#162), @4lovi4
- issue #80 doc string fix (#161), @4lovi4
- issue #139 readme fix (#160), @4lovi4

7.2.5 0.0.7 (October 15, 2020)

Only two changes—but relatively impactful ones—including the addition of a new workflow-style function (`abagen.get_samples_in_mask()`) to get preprocessed tissue samples without aggregating in specific brain regions and the ability to provide parcellations in donor-native space (rather than MNI space).

Documentation has been updated to reflect all new features!

- [ENH] Adds handling for donor-specific atlases (#156), @rmarkello
- [ENH] Allows users to get microarray samples in mask (#155), @rmarkello

7.2.6 0.0.6 (August 17, 2020)

Minor bug fixes, including:

- [FIX] Allow None input to CLI norm opts (#153), @rmarkello
- [MNT] Pin minimum pandas version to 0.25 (#151), @rmarkello
- [DOC] Adds Fulcher norm ref (#149), @rmarkello
- [FIX] Always remap annotation dataframe cols (#147), @rmarkello
- [FIX] Allow 'common' for donor_probes method (#146), @rmarkello

7.2.7 0.0.5 (March 24, 2020)

This release includes quite a bit of new functionality, including:

- Several new gene/sample normalization methods,
- A fetcher / new loader functions for RNAseq data (`abagen.fetch_rnaseq()`),
- The ability to use RNAseq data for selecting probes (`probe_selection='rnaseq`), and
- A new `donor_probes` parameter for `abagen.get_expression_data()` to control how probe selection is performed

Check out the documentation for more details!

- [ENH] Adds new mechanisms for probe selection (#145), @rmarkello
- [ENH,REF] Adds RNAseq probe selection method (#144), @rmarkello
- [TEST] Fix Azure maybe? (#143), @rmarkello
- [ENH] Adds FreeSurfer data fetcher (#142), @rmarkello
- [ENH] Adds fetchers / loaders for RNAseq data (#140), @rmarkello
- [TEST] Adds Windows testing with Azure (#141), @rmarkello
- [FIX,ENH] Error in get_expression_data, gene stability calculation (#136), @rmarkello
- [ENH,REF] New norms, utils, renamed modules (#135), @rmarkello

7.2.8 0.0.4 (February 26, 2020)

A release with a small bugfixes (#134) and a small fix-fix (#133)!

- [FIX] Coerce atlas dataobj to arr before indexing (#134), @rmarkello
- [REF] Drop probes with invalid/missing Entrez ID (#133), @rmarkello

7.2.9 0.0.3 (November 26, 2019)

A new release with some added features but primarily a good bit of re-arranging in the primary `abagen.get_expression_data()` workflow. Notable changes include:

- New parameters `region_agg`, `agg_metric`, `sample_norm`, and `gene_norm` (the latter of which supplants `donor_norm`), controlling how microarray samples are normalized and aggregated across donors;
- Large reductions in memory usage (#130), such that the primary workflow should only use ~2GB of RAM at its peak; and,
- Migration to CircleCI for all testing purposes!

Special thanks to @rhannema, @Ellen8780, @gifuni, and @VinceBaz for their contributions.

- [REF] Adds option to suppress norm warnings (#132), @rmarkello
- [ENH,REF] Adds new *region_agg* parameter (#131), @rmarkello
- [REF] Massive reduction in memory usage (#130), @rmarkello
- [FIX] Solves pandas bug with max_variance method (#128), @gifuni
- [STY] Fixes trailing whitespace (#129), @rmarkello
- [FIX] Fixes strange TypeError for pc_loading (#1), @rmarkello
- [REF] Ensures integer DataFrame when *return_count=True* (#127), @rhannema
- [ENH] Add fetcher for donor info (#126), @Ellen8780
- [REF,ENH] Modifies and adds normalization procedures (#119), @rmarkello
- [TEST] Updating CircleCI build (#122), @rmarkello
- [TEST] TravisCI –> CircleCI (#121), @rmarkello
- [REF] Removes *.get_data()* nibabel calls (#120), @rmarkello
- [FIX] Specify *engine='python'* in pandas queries (#117), @VinceBaz

7.2.10 0.0.2 (September 19, 2019)

This release comes with a **lot** of new changes, including:

- Several new arguments for `abagen.get_expression_data()`, including new probe selection methods, donor normalization techniques, and hemispheric mirroring of tissue samples;
- A command-line version of the primary workflow accessible via the `abagen` command;
- Improved data handling, using $HOME/abagen-data as the default storage location for data from the AHBA;
- New functionality for fetching raw AHBA donor MRI scans;
- Zenodo integration to make it easy to cite `abagen`; and,
- Massive documentation overhauls, with a dramatically updated user guide and API reference!

Special thanks to Golia Shafiei (@gshafiei), Ying-Qiu Zheng (@yingqiuz), James Frierson (@JamesFrierson1), and Arda Kosar (@abkosar) for their contributions.

- [MNT] Finishes Zenodo integration (#109), @rmarkello
- [MNT] Adds framework for Zenodo integration (#108), @rmarkello
- [REF] Fixes one-donor bug in get_expression_data() (#107), @rmarkello

- [REF] Identifies hippocampus as subcortex in Allen ontology (#106), @rmarkello
- [DOC] Updates documentation + contributing (#105), @rmarkello
- [REF] Mirroring before probe filtering (#101), @rmarkello
- [FIX] Installation not correctly bundling package data (#102), @rmarkello
- [MNT] Docs / package structure updates in prep for 0.2.0 (#95), @rmarkello
- [REF] abagen.io functions don't copy dataframes by default (#94), @rmarkello
- [FIX] Fixes broken include directive in API reference (#91), @rmarkello
- [ENH] Adds parameter for normalizing donor microarray expression values (#90), @rmarkello
- [ENH] Adds option to mirror samples across L/R hemispheres (#87), @rmarkello
- [ENH] Adds CLI for *abagen.get_expression_data* functionality (#82), @rmarkello
- [ENH] Adds ability to fetch raw AHBA MRIs (#85), @rmarkello
- [ENH] Adds ability to query gene groups (#83), @rmarkello
- [MNT,REF] Updates install, versioning, dependencies (#84), @rmarkello
- [REF] Adds brainstem to abagen.process ontology (#81), @rmarkello
- [DOC] Updates API documentation (#76), @rmarkello
- [REF,ENH] Adds new abagen.probes module (#67), @rmarkello
- [REF] Changes data directory locator for abagen data (#66), @rmarkello
- [FIX] Fixes doctest in abagen.mouse (#65), @rmarkello
- [REF] Removes .get_values() references (#64), @rmarkello
- [DOC] Adds logging to workflow functions (#61), @rmarkello
- Fixed abagen.mouse column ordering (#62), @abkosar
- [DOC] Update refs and http (#60), @rmarkello
- [REF] Use cached alleninf coordinates only (#59), @rmarkello
- [FIX] Removes RuntimeWarning in example code (#58), @rmarkello
- Updated README to include Allen Institute citations and disclaimers (#57), @JamesFrierson1
- [FIX] Catches AttributeError w/pandas fastparquet (#41), @rmarkello
- [REF] Updates get_expression_data() (#38), @rmarkello
- [TEST] Fixes tests (#34), @rmarkello
- Add mouse features (#32), @yingqiuz
- [TEST] Fix pytest version and update travis (#33), @rmarkello
- [TEST] Update travis testing (#31), @rmarkello
- [FIX] More fixes for atlas numbering (#30), @rmarkello
- [FIX] Allow non-sequential atlas numbering (#29), @rmarkello
- [ENH] Adds input check for remove_distance() (#28), @rmarkello
- [ENH] Allow label exclusion in *remove_distance()* (#27), @rmarkello
- [REF] Changes *remove_distance()* inputs (#26), @rmarkello

- [ENH] Add function for aggregating donors (#25), @rmarkello
- [ENH] Adds reannotated probe information (#24), @rmarkello
- [ENH] Adds *abagen.correct* for postprocessing (#20), @rmarkello
- [TEST] Removes pytest capturing (#23), @rmarkello
- [TEST] Calculates coverage only for extras (#22), @rmarkello
- [DOC] Updates doc-strings for primary functions (#19), @rmarkello
- [TEST] Add early test to reduce timeouts (#21), @rmarkello
- [FIX] Adds updated MNI coordinates file as backup (#17), @rmarkello
- [DOC] Updates default tolerance (#16), @gshafiei

7.2.11 0.0.1 (September 7, 2018)

Initial release of `abagen`, a toolbox for working with the Allen Brain Atlas human genetics data.

- [DOC] Updates various documentation (#15), @rmarkello
- [DOC] Adds LICENSE reference for alleninf (#14), @rmarkello
- [DOC] Updates README links and example usage (#13), @rmarkello
- [TEST] Updates tests of *abagen.get_expression_data()* (#12), @rmarkello
- [DOC] Adds Sphinx documentation (#11), @rmarkello
- [FIX] Resolves dataframe formatting issue (#10), @rmarkello
- [ENH] Adds DK atlas fetcher and updates README.md (#8), @rmarkello
- [REF] Cleaning up unused code (#7), @rmarkello
- [ENH] MAJOR refactoring of IO and processing (#4), @rmarkello
- [TEST] Adds .travis.yml and initial tests (#3), @rmarkello
- [STY] Stylistic updates to abagen.datasets (#2), @rmarkello

7.3 Command-line usage

You can use many of the primary workflows in `abagen` from the command line.

7.3.1 The `abagen` command

Assigns microarray expression data to ROIs defined in the specified *atlas*

This command aims to provide a workflow for generating pre-processed microarray expression data from the Allen Human Brain Atlas for arbitrary atlas designations. First, some basic filtering of genetic probes is performed, including:

1. Intensity-based filtering of microarray probes to remove probes that do not exceed a certain level of background noise (specified via the *–ibf_threshold* parameter),
2. Selection of a single, representative probe (or collapsing across probes) for each gene, specified via the *–probe_selection* parameter (and influenced by the *–donor_probes* parameter), and

3. Optional mirroring of the tissue samples across the left/right hemisphere boundary, as specified via the *–lr_mirror* parameter (turned off by default).

Tissue samples are then matched to parcels in the defined *atlas* for each donor. If *–atlas_info* is provided then this matching is constrained by both hemisphere and tissue class designation (e.g., cortical samples from the left hemisphere are only matched to ROIs in the left cortex, subcortical samples from the right hemisphere are only matched to ROIs in the left subcortex); see the *atlas_info* parameter description for more information.

Matching of microarray samples to parcels in *atlas* is done via a multi-step process:

1. Determine if the sample falls directly within a parcel,
2. Check to see if there are nearby parcels by slowly expanding the search space to include nearby voxels, up to a specified distance (specified via the *–tolerance* parameter),
3. If there are multiple nearby parcels, the sample is assigned to the closest parcel, as determined by the parcel centroid.

If at any step a sample can be assigned to a parcel the matching process is terminated. When the provided atlas is not volumetric (i.e., is surface-based) the samples are simply matched to the nearest vertex, and *–tolerance* is used as a standard deviation threshold. More control over the sample matching can be obtained by setting the *–missing* parameter.

Once all samples have been matched to parcels for all supplied donors, the microarray expression data are optionally normalized via the provided *–sample_norm* and *–gene_norm* functions (which are influenced by the *–norm_matched* and *–norm_structures* parameters) before being aggregated across donors via the supplied *–region_agg* and *–agg_metric* parameters.

```
usage: abagen [-h] [--version] [-v] [--atlas_info PATH]
              [--donors DONOR_ID [DONOR_ID ...]] [--data_dir PATH]
              [--n_proc N_PROC] [--ibf_threshold THRESHOLD]
              [--probe_selection METHOD] [--lr_mirror METHOD]
              [--sim_threshold THRESHOLD] [--missing METHOD] [--tol TOLERANCE]
              [--sample_norm METHOD] [--gene_norm METHOD] [--norm_all]
              [--norm_structures] [--region_agg METHOD] [--agg_metric METHOD]
              [--no-reannotated] [--no-corrected-mni] [--stdout]
              [--output-file PATH] [--save-counts] [--save-donors]
              atlas [atlas ...]
```

Positional Arguments

atlas	A NIFTI image in MNI152 space or two GIFTI images in fsaverage5 space, where each parcel is identified by a unique integer ID.

Named Arguments

--version	Show program version and exit.
-v, --verbose	Increase verbosity of status messages to display during workflow.

Options to specify information about the atlas used

--atlas_info, --atlas-info Filepath to CSV file containing information about *atlas*. The CSV file must have at least columns ["id", "hemisphere", "structure"] which contain information mapping the atlas IDs to hemispheres (i.e, "L", "R", or "B") and broad structural groups (i.e., "cortex", "subcortex/brainstem", "cerebellum"). If provided, this will constrain matching of tissue samples to regions in *atlas*. If the supplied *atlas* is a pair of GIFTI files with valid label tables this information will be intuited.

Options to specify which AHBA data to use during processing

--donors List of donors to use as sources of expression data. Specified IDs can be either donor numbers (i.e., 9861, 10021) or UIDs (i.e., H0351.2001). Can specify "all" to use all available donors. Default: "all"

--data_dir, --data-dir Directory where expression data should be downloaded to (if it does not already exist) / loaded from. If not specified this will check the environmental variable $ABAGEN_DATA, the $HOME/abagen-data directory, and the current working directory. If data does not already exist at one of those locations then it will be downloaded to the first of these location that exists and for which write access is enabled.

--n_proc, --n-proc Number of processors to use to download AHBA data. Can paralellize up to six times if all donors are requested. Default: 1

Options to specify processing options

--ibf_threshold, --ibf-threshold Threshold for intensity-based filtering of probes. This number should specify the ratio of samples, across all supplied donors, for which a probe must have signal above background noise in order to be retained. Default: 0.5

--probe_selection, --probe-selection Possible choices: average, corr_intensity, corr_variance, diff_stability, max_intensity, max_variance, mean, pc_loading, rnaseq

Selection method for subsetting (or collapsing across) probes that index the same gene. Must be one of {"average", "mean", "max_intensity", "max_variance", "pc_loading", "corr_variance", "corr_intensity", "diff_stability", "rnaseq"}. Default: "diff_stability"

--lr_mirror, --lr-mirror Possible choices: None, bidirectional, leftright, rightleft

Whether to mirror microarray expression samples across hemispheres to increase spatial coverage. Using "bidirectional" will mirror samples across both hemispheres, "leftright" will mirror samples in the left hemisphere to the right, and "rightleft" will mirror the right to the left. Default: None

--sim_threshold, --sim-threshold Threshold for inter-areal similarity filtering. Samples are correlated across probes and those samples with a total correlation less than the the provided threshold s.d. below the mean across samples are excluded from futheranalysis. If not specified no filtering is performed. Default: None

--missing Possible choices: None, centroids, interpolate

How to handle regions in *atlas* that are not assigned any tissue samples. If "centroids", any empty regions will be assigned the expression value of the nearest tissue sample (defined as the sample with the closest Euclidean distance to the parcel centroid). If "interpolate", expression values will be interpolated in the empty regions by assigning every node in the region the expression of the nearest sample and taking a weighted (inverse distance) average. If not specified empty regions will be returned with expression values of NaN. Default: None

--tol, --tolerance Distance (in mm) that a sample can be from a parcel for it to be matched to that parcel. If *atlas* is GIFTI files then this measure is a standard deviation threshold (i.e., samples greater than *tolerance* SDs away from the mean matched distance are ignored). Default: 2

--sample_norm, --sample-norm Possible choices: center, demean, minmax, mixed_sig, mixed_sigmoid, robust_sigmoid, rs, rsig, scaled_robust_sigmoid, scaled_rsig, scaled_sig, scaled_sig_qnt, scaled_sigmoid, scaled_sigmoid_quantiles, sig, sigmoid, srs, zscore, None, None

Method by which to normalize microarray expression values for each sample prior to collapsing into regions in *atlas*. Expression values are normalized separately for each sample and donor across genes. If None is specified then no normalization is performed. Default: "srs"

--gene_norm, --gene-norm Possible choices: center, demean, minmax, mixed_sig, mixed_sigmoid, robust_sigmoid, rs, rsig, scaled_robust_sigmoid, scaled_rsig, scaled_sig, scaled_sig_qnt, scaled_sigmoid, scaled_sigmoid_quantiles, sig, sigmoid, srs, zscore, None, None

Method by which to normalize microarray expression values for each donor prior to collapsing across donors. Expression values are normalized separately for each gene for each donor across all expression samples. If None is specified then no normalization is performed. Default: "srs"

--norm_all, --norm-all Whether to perform gene normalization (*gene_norm*) across all available samples instead of only across samples that were matched to regions in *atlas*. If *atlas* is very small (i.e., only a few regions of interest) using *–norm_all* is suggested.

--norm_structures, --norm-structures Whether to perform gene normalization (*gene_norm*) within structural classes (i.e., "cortex", "subcortex/brainstem", "cerebellum") instead of across all available samples.

--region_agg, --region-agg Possible choices: donors, samples

When multiple samples are identified as belonging to a region in *atlas* this determines how they are aggegated. If 'samples', expression data from all samples for all donors assigned to a given region are combined. If 'donors', expression values for all samples assigned to a given region are combined independently for each donor before being combined across donors. See *agg_metric* for mechanism by which samples are combined. Default: 'donors'

--agg_metric, --agg-metric Possible choices: mean, median

Mechanism by which to (1) reduce expression data of multiple samples in the same *atlas* region, and (2) reduce donor-level expression data into a single "group" expression dataframe. Must be one of {"mean", "median"}. Default: "mean"

Options to modify the AHBA data used

--no-reannotated, --no_reannotated Whether to use the original probe information from the AHBA dataset instead of the reannotated probe information from Arnatkeviciūtė et al., 2019. Using reannotated probe information discards probes that could not be reliably matched to genes. Default: False (i.e., use reannotations)

--no-corrected-mni, --no_corrected_mni Whether to use the original MNI coordinates provided with the AHBA data instead of the "corrected" MNI coordinates shipped with the *alleninf* package when matching tissue samples to anatomical regions. Default: False (i.e., use corrected coordinates)

Options to modify how data are output

--stdout Generated region x gene dataframes will be printed to stdout for piping to other things. You should REALLY consider just using –output-file instead and working with the generated CSV file(s). Incompatible with *–save-counts* and *–save-donors* (i.e., this will override those options). Default: False

--output-file, --output_file Path to desired output file. The generated region x gene dataframe will be saved here. Default: $PWD/abagen_expression.csv

--save-counts, --save_counts Whether to save dataframe containing number of samples from each donor that were assigned to each region in *atlas*. If specified, will be saved to the path specified by *output-file*, appending "counts" to the end of the filename. Default: False

--save-donors, --save_donors Whether to save donor-level expression dataframes instead of aggregating expression across donors with provided *agg_metric*. If specified, dataframes will be saved to path specified by *output-file*, appending donor IDs to the end of the filename. Default: False

7.4 User guide

`abagen` aims to provide a reproducible pipeline for processing and preparing microarray expression data provided by the Allen Human Brain Atlas (AHBA) for research analyses.

This user guide steps through the basics of fetching microarray expression data from the AHBA, defining an atlas or parcellation for wrangling that data, and actually parcellating the data into a more usable, analyis-ready format. If you still have questions after going through this guide you can refer to the *Reference API* or ask a question on GitHub.

7.4.1 The Allen Human Brain Atlas dataset

Fetching the AHBA data

In order to use `abagen`, you'll need to download the AHBA microarray data. You can download it with the following command:

```
>>> import abagen
>>> files = abagen.fetch_microarray(donors='all', verbose=0)
```

Note: Downloading the entire dataset (about 4GB) can take a long time depending on your internet connection speed! If you don't want to download all the donors you can provide the subject IDs of the donors you want as a list (e.g., `['9861', '10021']`) instead of passing `'all'`.

This command will download data from the specified donors into a folder called `microarray` in the `$HOME/abagen-data` directory. If you have already downloaded the data you can provide the `data_dir` argument to specify where the files have been stored:

```
>>> files = abagen.fetch_microarray(donors=['12876', '15496'], data_dir='/path/to/my/
↪data/')
```

Alternatively, `abagen` will check the directory specified by the environmental variable `$ABAGEN_DATA` and use that as the download location if the dataset does not already exist there.

If you provide a path to `data_dir` (or specify a path with `$ABAGEN_DATA`) the directory specified should have the following structure:

```
/path/to/my/data/
├── normalized_microarray_donor10021/
│   ├── MicroarrayExpression.csv
│   ├── Ontology.csv
│   ├── PACall.csv
│   ├── Probes.csv
│   └── SampleAnnot.csv
├── normalized_microarray_donor12876/
├── normalized_microarray_donor14380/
├── normalized_microarray_donor15496/
├── normalized_microarray_donor15697/
└── normalized_microarray_donor9861/
```

(Note the directory does not have to be named `microarray` for this to work.)

Loading the AHBA data

The `files` object returned by *`abagen.fetch_microarray()`* is a nested dictionary with filepaths to the five different file types in the AHBA microarray dataset. The keys are the donor IDs:

```
>>> print(files.keys())
dict_keys(['9861', '10021', '12876', '14380', '15496', '15697'])
```

And the values for each entry are a sub-dictionary of the downloaded files:

```
>>> print(sorted(files['9861']))
['annotation', 'microarray', 'ontology', 'pacall', 'probes']
```

You can load the data in these files using the *`abagen.io`* functions. There are IO functions for each of the five file types; you can get more information on the functions and the data contained in each file type by looking at the *Reference API*. Notably, all IO functions return `pandas.DataFrame` objects for ease-of-use.

For example, you can load the annotation file for the first donor with:

```
>>> data = files['9861']
>>> annotation = abagen.io.read_annotation(data['annotation'])
```

(continues on next page)

(continued from previous page)

```
>>> print(annotation)
           structure_id  slab_num  well_id  ... mni_x mni_y mni_z
sample_id                                   ...
1                  4077        22      594  ...   5.9 -27.7  49.7
2                  4323        11     2985  ...  29.2  17.0  -2.9
3                  4323        18     2801  ...  28.2 -22.8  16.8
...                 ...       ...      ...  ...   ...   ...   ...
944                4758        67     1074  ...   7.9 -72.3 -40.6
945                4760        67     1058  ...   8.3 -57.4 -59.0
946                4761        67     1145  ...   9.6 -46.7 -47.6

[946 rows x 13 columns]
```

And you can do the same for, e.g., the probe file with:

```
>>> probes = abagen.io.read_probes(data['probes'])
>>> print(probes)
                      probe_name  gene_id   gene_symbol                                ␣
↪ gene_name  entrez_id chromosome
probe_id
1058685              A_23_P20713      729           C8G  complement component 8, gamma␣
↪polypeptide        733          9
1058684   CUST_15185_PI416261804      731            C9                    complement␣
↪component 9        735          5
1058683             A_32_P203917      731            C9                    complement␣
↪component 9        735          5
...                          ...      ...           ...                                ␣
↪        ...        ...        ...
1071209             A_32_P885445  1012197  A_32_P885445    AGILENT probe A_32_P885445␣
↪(non-RefSeq)       <NA>        NaN
1071210               A_32_P9207  1012198    A_32_P9207      AGILENT probe A_32_P9207␣
↪(non-RefSeq)       <NA>        NaN
1071211              A_32_P94122  1012199   A_32_P94122     AGILENT probe A_32_P94122␣
↪(non-RefSeq)       <NA>        NaN

[58692 rows x 6 columns]
```

The other IO functions work similarly for the remaining filetypes.

7.4.2 Defining a parcellation

Acceptable parcellations

In order to process the microarray expression data from AHBA you'll need a parcellation (or "atlas"). Here, we define a parcellation (atlas) as either (1) a NIFTI image in MNI space, or (2) a tuple of GIFTI images in fsaverage space (and with fsaverage5 resolution!). In both cases, parcels in the atlas should be denoted by unique integer IDs (distinct across hemispheres). The primary workflows in abagen are designed to readily accept any parcellations / atlases in this format; however, if you want to use a different format please refer to *Non-standard parcellations*.

For demonstration purposes, abagen has a copy of the Desikan-Killiany atlas that you can use. Here, we load the volumetric atlas by default:

```
>>> import abagen
>>> atlas = abagen.fetch_desikan_killiany()
```

The returned object `atlas` is a dictionary with two keys: `image`, which is filepath to a NIFTI image containing the atlas data, and `info`, which is a filepath to a CSV file containing extra information about the parcellation:

```
>>> print(atlas['image'])
/.../data/atlas-desikankilliany.nii.gz
>>> print(atlas['info'])
/.../data/atlas-desikankilliany.csv
```

You can load the surface version of the atlas by providing the `surface` parameter:

```
>>> atlas = abagen.fetch_desikan_killiany(surface=True)
>>> print(atlas['image'])
('/.../data/atlas-desikankilliany-lh.label.gii.gz', '/.../data/atlas-desikankilliany-rh.
↪label.gii.gz')
```

Providing additional parcellation info

While only the image (i.e., NIFTI or GIFTIs) is required for processing the microarray data, the CSV file with information on the parcellation scheme can also be very useful. In particular, `abagen` can use the CSV file to constrain the matching of tissue samples to anatomical regions in the atlas image.

Note: If you are using a surface atlas and your GIFTI files have valid label tables then `abagen` will automatically create a pandas.DataFrame with all the relevant information described below. However, you can always provide a separate CSV file if you are unsure and this will override any label tables present in the GIFTI files.

If you want to supply your own CSV file with information about an atlas you must ensure it has (at least) the following columns:

1. `id`: an integer ID corresponding to the labels in the `atlas` image
2. `hemisphere`: a left/right/bilateral hemispheric designation (i.e., ‘L’, ‘R’, or ‘B’)
3. `structure`: a broad structural class designation (i.e., one of ‘cortex’, ‘subcortex/brainstem’, ‘cerebellum’, ‘white matter’, or ‘other’)

For example, a valid CSV might look like this:

```
>>> import pandas as pd
>>> atlas_info = pd.read_csv(atlas['info'])
>>> print(atlas_info)
    id                    label hemisphere            structure
0    1                 bankssts          L               cortex
1    2  caudalanteriorcingulate          L               cortex
2    3      caudalmiddlefrontal          L               cortex
..  ..                      ...        ...                  ...
80  81              hippocampus          R  subcortex/brainstem
81  82                 amygdala          R  subcortex/brainstem
82  83                brainstem          B  subcortex/brainstem

[83 rows x 4 columns]
```

Notice that extra columns (i.e., `label`) are okay as long as the three required columns are present! If you want to confirm your file is formatted correctly you can use `abagen.images.check_atlas()`:

```
>>> from abagen import images
>>> atlas = abagen.fetch_desikan_killiany()
>>> atlas = images.check_atlas(atlas['image'], atlas['info']);
```

If something is amiss with the file this function will raise an error and try to give some information about what you should check for.

Important: You might be asking: **"why should I provide this extra information for my parcellation?"** Providing this CSV file will ensure that microarray samples designated as belonging to a given hemisphere/structure by the AHBA ontology are not matched to regions in the `atlas` image with different hemispheric/structural designations. That is, if the AHBA ontology specifies that a tissue sample comes from the left hemisphere subcortex, it will only ever be matched to regions in `atlas` belonging to the left hemisphere subcortex.

While this seems trivial, it is **very important** because there are numerous tissue samples which occur on the boundaries of hemispheres and structural classes (i.e., cortex/subcortex). In many instances, these samples won't fall directly within a region of the `atlas`, at which point `abagen` will attempt to match them to nearby regions. Without the hemisphere/structure information provided by this CSV file there is a high likelihood of misassigning samples, leading to biased or skewed expression data.

Individualized parcellations

Instead of providing a single parcellation image that will be used for all donors, you can instead provide a parcellation image for each donor in the space of their "raw" (or native) T1w image. `abagen` ships with versions of the Desikan-Killiany parcellation defined in donor-native space:

```
>>> atlas = abagen.fetch_desikan_killiany(native=True)
>>> print(atlas['image'].keys())
dict_keys(['9861', '10021', '12876', '14380', '15496', '15697'])
>>> print(atlas['image']['9861'])
/.../data/native_dk/9861/atlas-desikankilliany.nii.gz
```

Note here that `atlas['image']` is a dictionary, where the keys are donor IDs and the corresponding values are paths to the parcellation for each donor. The primary workflows in `abagen` that accept a single atlas (i.e., *`abagen.get_expression_data()`* and *`abagen.get_samples_in_mask()`*) will also accept a dictionary of this format.

We also provide donor-specific surface atlases (derived from the FreeSurfer outputs that can be fetched with `abagen.datasets.fetch_freesurfer()`). These atlases are also shipped with `abagen` and can be loaded with:

```
>>> atlas = abagen.fetch_desikan_killiany(native=True, surface=True)
>>> print(atlas['image'].keys())
dict_keys(['9861', '10021', '12876', '14380', '15496', '15697'])
>>> print(atlas['image']['9861'])
('/.../9861/atlas-desikankilliany-lh.label.gii.gz', '/.../9861/atlas-desikankilliany-rh.
↪label.gii.gz')
```

Note that if you are using your own donor-specific surface atlases they must, by default, be based on the geometry of the FreeSurfer surfaces provided with `abagen.datasets.fetch_freesurfer()`. If you wish to use surface atlases based on different geometry please refer to *Non-standard parcellations*, below.

Finally, when in doubt we recommend simply using a standard-space, group-level atlas; however, we are actively investigating whether native-space atlases provide any measurable benefits to the `abagen` workflows.

Note: The donor-native volumetric versions of the DK parcellation shipped with abagen were generated by Arnatkevičiūte et al., 2018, *NeuroImage*, and are provided under the CC BY 4.0 license. The donor-native surface versions of the DK parcellation were generated by Romero-Garcia et al., 2017, *NeuroImage*, and are also provided under the CC BY 4.0 license.

Non-standard parcellations

If you'd like to use a non-standard atlas in the primary abagen workflows that may be possible—with some caveats. That is, the constraining factor here is the coordinates of the tissue samples from the AHBA: they are available in (1) the native space of each donor's MRI, or (2) MNI152 space, and we strongly encourage you to use one of these options (rather than e.g., attempting to register the coordinates to a new space). If you provide a group-level atlas the toolbox will default to using the MNI152 coordinates; if you provide donor-specific atlases then the tooblox will use the native coordinates. Thus, by default, abagen prefers you use one of the atlas conformations described above.

However, if you have an atlas in a different space or resolution you can (potentially) use it in the primary abagen workflows. To do this you will need to create a `abagen.AtlasTree` object. All atlases provided are internally coerced to *AtlasTree* instances, which is then used to assign microarray tissue samples to parcels in the atlas.

Take, for example, a surface atlas in fsaverage6 resolution (by default, surface atlases are assumed to be fsaverage5 resolution). In this case, you simply need to supply the relevant geometry files for the atlas and specify the space of the atlas:

```
>>> from abagen import images
>>> atlas = ('/.../fsaverage6-lh.label.gii', '/.../fsaverage6-rh.label.gii')
>>> surf = ('/.../fsaverage6-lh.surf.gii', '/.../fsaverage6-lh.surf.gii')
>>> atlas = images.check_atlas(atlas, geometry=surf, space='fsaverage6')
```

The same procedure can be used for an atlas using fsLR geometry:

```
>>> from abagen import images
>>> atlas = ('/.../fslr32k-lh.label.gii', '/.../fslr32k-rh.label.gii')
>>> surf = ('/.../fslr32k-lh.surf.gii', '/.../fslr32k-lh.surf.gii')
>>> atlas = images.check_atlas(atlas, geometry=surf, space='fslr')
```

7.4.3 Parcellating expression data

Basic usage

Once you've downloaded the microarray data and selected your parcellation you can process the data. This should be as simple as:

```
>>> expression = abagen.get_expression_data(atlas['image'])
```

If you are using a volumetric image it is highly recommended you provide the additional information on your parcellation, which can be done with:

```
>>> expression = abagen.get_expression_data(atlas['image'], atlas['info'])
```

Note: By default this function will use data from *all* the donors! Wrangling all this raw microarray data can be quite time-consuming, so if you'd like to speed up this step make sure you've performed the *IO installation*. Alternatively, if you don't want to use all the donors when testing or debugging the code, you can provide the subject IDs of the donors you want (e.g., `donors=['9861', '10021']`).

The *abagen.get_expression_data()* function will print out some information about what's happening as it goes. However, briefly the function:

1. Fetches the microarray data from AHBA (if this has not already been done). Refer to parameters `data_dir` and `donors` for more info.
2. Updates the MNI coordinates of all the tissue samples from AHBA using the coordinates from the `alleninf` package. This occurs by default; refer to parameter `corrected_mni` for more info.
3. Mirrors samples across hemispheres to increase spatial coverage. This does not occur by default; refer to parameter `lr_mirror` for more info (or see *Duplicating samples with the lr_mirror parameter*).
4. Reannotates microarray probe-to-gene mappings with information from Arnatkevičiūtė et al., 2019, NeuroImage. This occurs by default; refer to parameter `reannotated` for more info.
5. Performs a similarity-based filtering of tissue samples, removing those samples whose expression across probes is poorly correlated with other samples. This does not occur by default; refer to parameter `sim_threshold` for more info.
6. Performs intensity-based filtering of probes to remove those that do not exceed background noise. This occurs by default with a threshold of 0.5 (i.e., probes must exceed background noise in 50% of all tissue samples); refer to parameter `ibf_threshold` for more info.
7. Selects a representative probe amongst those probes indexing the same gene. This occurs by default by selecting the probe with the highest differential stability amongst donors; refer to parameter `probe_selection` for more info (or see *Probe selection options*).
8. Matches tissue samples to regions in the user-specified `atlas`. Refer to parameters `atlas`, `atlas_info`, `missing`, and `tolerance` for more info (or see *Filling in data with the missing parameter*).
9. Normalizes expression values for each sample across genes for each donor. This occurs by default using a scaled robust sigmoid normalization function; refere to parameter `sample_norm` for more info.
10. Normalizes expression values for each gene across samples for each donor. This occurs by default using a scaled robust sigmoid normalization function; refer to parameter `gene_norm` for more info.
11. Aggregates samples within regions in the user-specified `atlas` based on matches made in Step 7. By default, samples are averaged separately for each donor and then averaged across donors. Refer to parameters `region_agg`, `agg_metric`, and `return_donors` for more info.

You can investigate all these parameters and options for modifying how the `expression` array is generated by looking at the *Reference API*.

The parcellated expression DataFrame

The `expression` object returned by *abagen.get_expression_data()* is a `pandas.DataFrame`, where rows correspond to region labels as defined in the atlas image, columns correspond to genes, and entry values are microarray expression data normalized and aggregated across donors:

```
>>> print(expression)
gene_symbol      A1BG  A1BG-AS1       A2M  ...       ZYX     ZZEF1      ZZZ3
label                                      ...
1            0.498266  0.664570  0.395276  ...  0.675843  0.555539  0.487572
2            0.649068  0.578997  0.496142  ...  0.483165  0.382653  0.504041
3            0.530613  0.623289  0.516300  ...  0.732930  0.359707  0.450664
```

(continues on next page)

(continued from previous page)

```
...          ...       ...       ...  ...       ...       ...       ...
81          0.388748  0.277961  0.474202  ...  0.279683  0.480953  0.405504
82          0.825836  0.602271  0.334143  ...  0.195722  0.447894  0.746475
83          0.384593  0.203654  0.746060  ...  0.379274  0.706803  0.509437

[83 rows x 15633 columns]
```

By default the data are normalized using a scaled robust sigmoid function such that expression values for a given gene will range from 0-1, where 0 indicates the region with the lowest expression of that gene and 1 indicates the region with highest.

Since the generated DataFrame is an aggregate (default: average) of multiple donors it is possible (likely) that a given region may not have any expression values *exactly* equal to 0 or 1.

Getting dense expression data

Unfortunately, due to how tissue samples were collected from the donor brains it is possible that some regions in an atlas may not be represented by any expression data. In the above example, two of the rows are missing data:

```
>>> print(expression.loc[[72, 73]])
gene_symbol  A1BG  A1BG-AS1  A2M  ...  ZYX  ZZEF1  ZZZ3
label                             ...
72            NaN       NaN  NaN  ...  NaN    NaN   NaN
73            NaN       NaN  NaN  ...  NaN    NaN   NaN

[2 rows x 15633 columns]
```

These regions, corresponding to the right frontal pole (label 72) and right temporal pole (label 73) in the Desikan-Killiany atlas, were not matched to any tissue samples; this is likely due to the fact that only two of the six donors have tissue samples taken from the right hemisphere.

If you require a *dense* matrix—that is, you need expression values for **every** region in your `atlas`—there are a few parameters that you can consider tuning to try and achieve this.

Filling in data with the `missing` parameter

By default, the *abagen.get_expression_data()* function will attempt to be as precise as possible in matching microarray samples with brain regions. It takes the following steps to do this for each tissue sample:

1. Determine if the sample falls directly within a region of `atlas`.
2. Check to see if the sample is close to any regions by slowly expanding the search space (in 1mm increments) to include nearby voxels up to a specified distance threshold (specified via the `tolerance` parameter).
3. If there are multiple nearby regions, determine which region is closer by calculating the center-of-mass of the abutting regions.

If at any step a sample can be assigned to a region in `atlas` the sample is assigned to that region and the matching procedure is terminated. However, as we saw, regions with no assigned samples from any donor are simply left as NaN.

If you would like to force all regions to be assigned at least one sample you can set the `missing` parameter. This parameter accepts three options: `None` (default), `"centroids"`, and `"interpolate"`. By setting this parameter the workflow will go through the normal procedure as documented above and then, once all samples are matched, check for any empty regions and assign them expression values based on the specified method.

When using the 'centroid' method the empty regions in the atlas will be assigned the expression values of the tissue sample falling closest to the centroid of that region. Note that this procedure is only performed when _all_ donors are missing data in a given region. In this case, a weighted average of the matched samples are taken across donors, where weights are calculated as the inverse distance between the tissue sample matched to the parcel centroid for each donor.

When using the 'interpolate' method, expression values will be interpolated in the empty regions by assigning every node in the region the expression of the nearest tissue sample. The weighted (inverse distance) average of the densely-interpolated map will be taken and used to represent parcellated expression values for the region. Note that, unlike in the centroid matching procedure described above, this interpolation is done independently for every donor, irrespective of whether other donors have tissue samples that fall within a given region.

Thus, setting the `missing` parameter when calling *abagen.get_expression_data()* will **always** return a dense expression matrix (at the expense of some anatomical precision):

```
# first, check with ``missing='centroids'``
>>> exp_centroids = abagen.get_expression_data(atlas['image'], atlas['info'],
...                                               missing='centroids')
>>> print(exp_centroids.loc[[72, 73]])
gene_symbol       A1BG   A1BG-AS1       A2M  ...       ZYX     ZZEF1      ZZZ3
label                                        ...
72            0.574699  0.750184  0.246746  ...  0.656938  0.193677  0.647785
73            0.725151  0.652906  0.528831  ...  0.478334  0.501293  0.483642

[2 rows x 15633 columns]

# then, check with ``missing='interpolate'``
>>> exp_interpolate = abagen.get_expression_data(atlas['image'], atlas['info'],
...                                                 missing='interpolate')
>>> print(exp_interpolate.loc[[72, 73]])
gene_symbol       A1BG   A1BG-AS1       A2M  ...       ZYX     ZZEF1      ZZZ3
label                                   ...
72            0.532308  0.710846  0.299322  ...  0.675837  0.301105  0.586290
73            0.736345  0.663072  0.497092  ...  0.507378  0.467046  0.531494

[2 rows x 15633 columns]
```

Warning: Refer to the documentation for *normalization* for additional information on how other settings interact with the `missing` parameter.

Duplicating samples with the `lr_mirror` parameter

If your parcellation is sufficiently low-resolution it is likely that most regions in the left hemisphere (for which all six donors have tissue samples) will be matched to at least one sample, whereas regions in the right hemisphere may come up short.

To remedy this you can try modifying the `lr_mirror` parameter when calling *abagen.get_expression_data()*. This parameter accepts four options: `None` (default), `"bidirectional"`, `"leftright"`, and `"rightleft"`. As the name suggests, the `lr_mirror` options control whether tissue samples are mirrored across the left/right hemisphere axis. By supplying the 'bidirectional' options, all samples in the left hemisphere are duplicated and mirrored onto the right hemisphre, and vice-versa for right to left. The other options ('leftright' and 'rightleft) will mirror only one hemisphere (i.e., 'leftright' will mirror samples in the left onto the right hemisphere).

Unlike the `missing` parameter this will *not guarantee* that all regions are matched to a sample, but it will increase the likelihood that this happens:

```
>>> exp_mirror = abagen.get_expression_data(atlas['image'], atlas['info'],
...                                         lr_mirror='bidirectional')
>>> print(exp_mirror.loc[[72, 73]])
gene_symbol      A1BG   A1BG-AS1       A2M  ...       ZYX     ZZEF1      ZZZ3
label                                  ...
72           0.832617  0.648154  0.425707  ...  0.580406  0.439378  0.799856
73           0.682180  0.569551  0.627497  ...  0.430146  0.302926  0.425995

[2 rows x 15633 columns]
```

Note that since this effectively duplicates the number of tissue samples the function runtime will increase somewhat. Also, importantly, setting the `lr_mirror` parameter will change the expression values of **all** of the regions in the generated matrix–not just the regions that are missing data. It is worth considering which (if either!) of these options best suits your intended analysis.

7.4.4 Probe selection options

The probes used to measure microarray expression levels in the AHBA data are often redundant; that is, there are frequently several probes indexing the same gene. Since the output of the *abagen.get_expression_data()* workflow is a region by gene dataframe, at some point we need to transition from indexing probe expression levels to indexing gene expression levels. Effectively, this means we need to select from or condense the redundant probes for each gene; however, there are a number of ways to do that.

Currently, `abagen` supports eight options for this probe to gene conversion. All the options have been used at various points throughout the published record, so while there is no "right" choice we do encourage using the default option (*differential stability*) due to recent work by Arnatkevičiūte et al., 2019 showing that it provides the highest fidelity to RNA sequencing data.

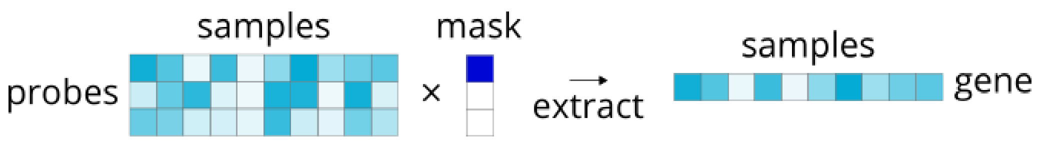
Available methods for `probe_selection` fall into two primary families:

1. *Selecting a representative probe*, and
2. *Collapsing across probes*

We describe all the methods within these families here. Methods can be implemented by passing the `probe_selection` argument to the *abagen.get_expression_data()* function. For a selection of references to published works that have used these different methods please see the documentation of `abagen.probes_.collapse_probes()`.

Selecting a representative probe

The first group of methods aim to **select** a single probe from each redundant group. This involves generating some sort of selection criteria and masking the original probe by sample expression matrix to extract only the chosen probe, which will be used to represent the associated gene's microarray expression values:

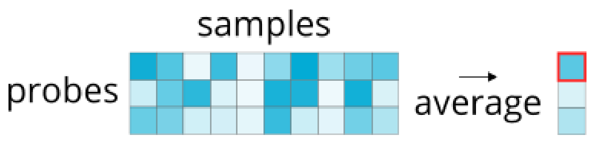


The only difference between methods in this group is the criteria used to select which probe to retain. The descriptions below explain how the mask in the above diagram is generated for each available option; in each diagram the red outline on the generated vector indicates which entry will be used to mask the original matrix. The extraction procedure (i.e., applying the mask to the original probe by sample expression matrix) is identical for all these methods.

Max intensity

```
>>> abagen.get_expression_data(atlas['image'], probe_selection='max_intensity')
```

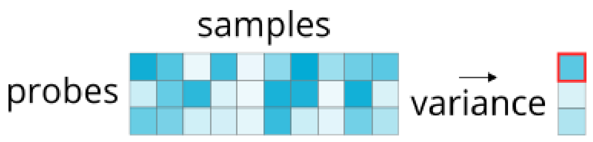
Selects the probe with the highest average expression across all samples (where samples are concatenated across donors).



Max variance

```
>>> abagen.get_expression_data(atlas['image'], probe_selection='max_variance')
```

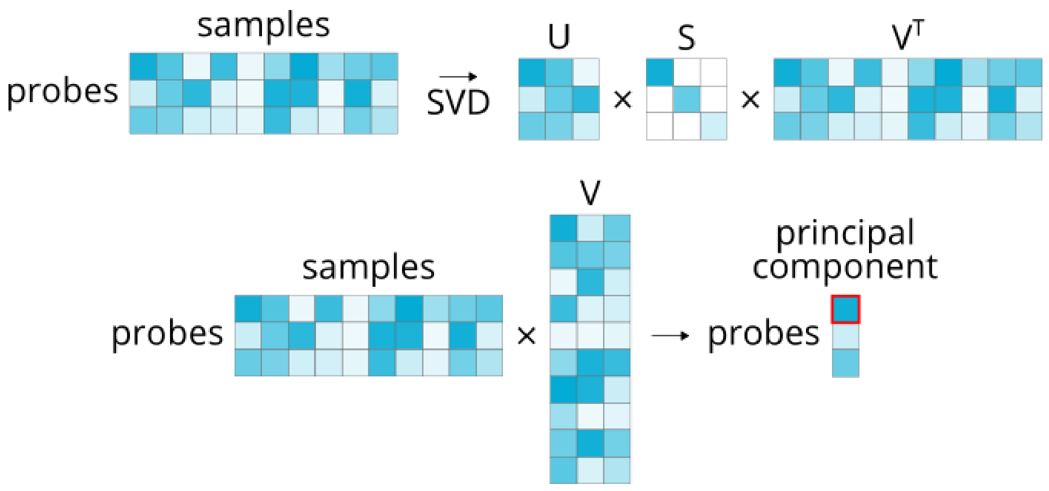
Selects the probe with the highest variance in expression across all samples (where samples are concatenated across donors).



Principal component loading

```
>>> abagen.get_expression_data(atlas['image'], probe_selection='pc_loading')
```

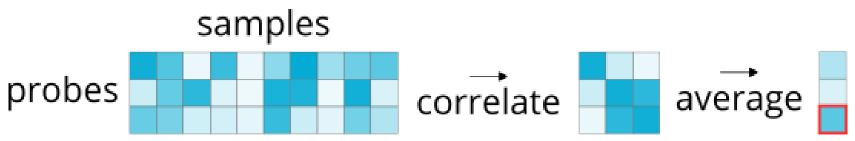
Selects the probe with the highest loading on the first principal component derived from the probe microarray expression across all samples (where samples are concatenated across donors).



Correlation

```
>>> abagen.get_expression_data(atlas['image'], probe_selection='corr_intensity')
>>> abagen.get_expression_data(atlas['image'], probe_selection='corr_variance')
```

When there are more than two probes indexing the same gene, selects the probe with the highest average correlation to other probes across all samples (where samples are concatenated across donors).

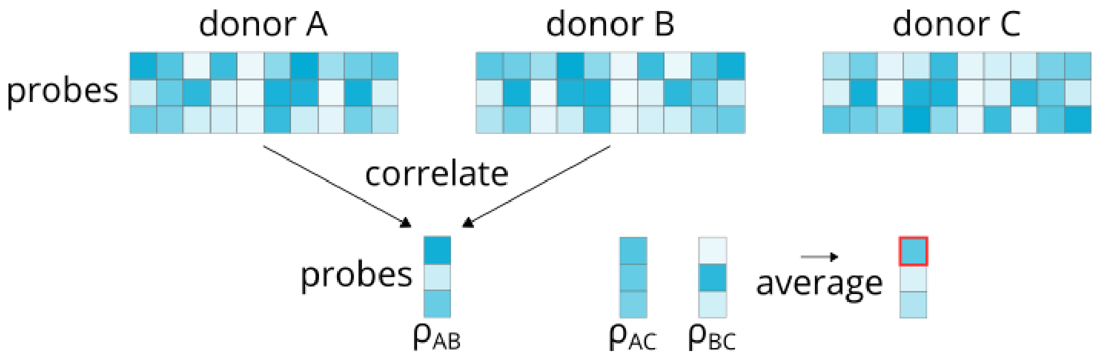


When there are exactly two probes the correlation procedure cannot be used, and so you can fall back to either the *Max intensity* (`corr_intensity`) or the *Max variance* (`corr_variance`) criteria.

Differential stability

```
>>> abagen.get_expression_data(atlas['image'], probe_selection='diff_stability')
```

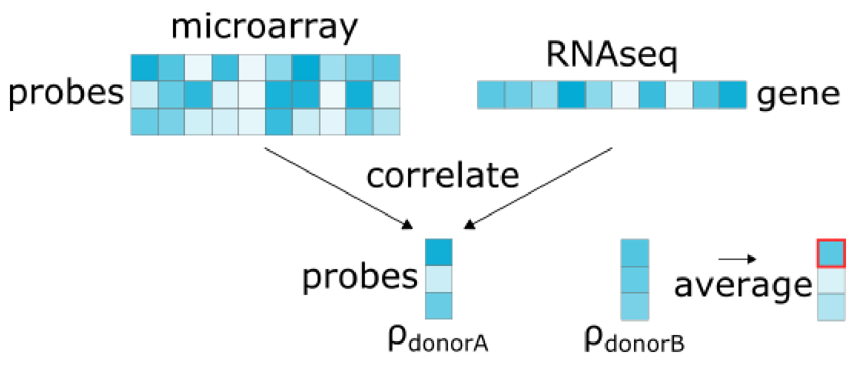
Computes the Spearman correlation of microarray expression values for each probe across brain regions for every **pair** of donors. Correlations are averaged and the probe with the highest correlation is retained.



RNAseq

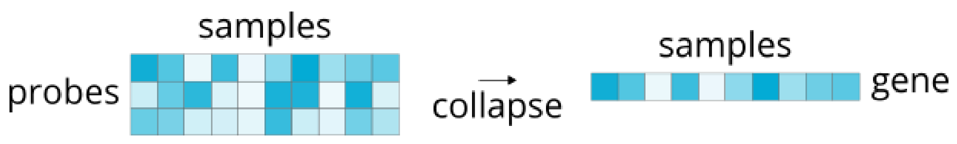
```
>>> abagen.get_expression_data(atlas['image'], probe_selection='rnaseq')
```

Computes the Spearman correlation of microarray expression values for each probe across brain regions with RNAseq data for the corresponding gene. As only two donors have RNAseq data (donors #9861 and 10021), this method only computes the correlations for these two donors. Correlations are averaged across the two donors and the probe with the highest correlation for each gene is retained.



Collapsing across probes

In contrast to selecting a single representative probe for each gene and discarding the others, we can instead opt to use all available probes and **collapse** them into a unified representation of the associated gene:

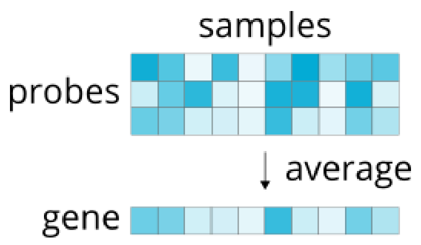


Currently only one method supports this operation.

Average

```
>>> abagen.get_expression_data(atlas['image'], probe_selection='average')
```

Takes the average expression values for all probes indexing the same gene.



Providing `'mean'` instead of `'average'` will return identical results.

7.4.5 Donor aggregation in probe selection

Unless otherwise specified in the description of that method, probe selection is performed using data aggregated across samples from *all* donors. However, this may not be desired: the probe that most reliably indexes a gene in one donor may differ from the probe that does so in another donor.

To allow for this possibility, we describe three options for modifying how probe selection is performed across donors in detail below. These methods can be implemented by passing the `donor_probes` argument to the *`abagen.get_expression_data()`* function.

Aggregate selection across donors

```
>>> abagen.get_expression_data(atlas['image'], donor_probes='aggregate')
```

The default option, this will aggregate tissue samples from all donors and apply the chosen `probe_selection` method to this single probe x sample matrix. The probe chosen to represent each gene will be identical across all donors.

Independent selection for donors

```
>>> abagen.get_expression_data(atlas['image'], donor_probes='independent')
```

Performs the chosen `probe_selection` method independently for each donor. The probe chosen to represent each gene *may* be different across donors.

Note: this option cannot be used when the specified `probe_selection` is one of: *'diff_stability'*, *'rnaseq'*, or *'average'*.

Most common selection across donors

```
>>> abagen.get_expression_data(atlas['image'], donor_probes='common')
```

Performs the chosen `probe_selection` method independently for each donor and then uses the most commonly-selected probe to represent each gene. The probe chosen to represent each gene will be identical across all donors.

Note: this option cannot be used when the specified `probe_selection` is one of: *'diff_stability'*, *'rnaseq'*, or *'average'*.

7.4.6 Data normalization options

The microarray expression data provided by the AHBA has been subjected to some normalization procedures designed to mitigate potential differences in expression values between donors due to "batch effects." Despite these procedures, there are still some notable differences between donors present in the downloadable data.

By default, *`abagen.get_expression_data()`* aggregates expression data across donors (though this can be prevented via the `return_donors` parameter). Prior to aggregation, the function performs a within-donor normalization procedure to attempt to mitigate donor-specific effects; however, there are a number of ways to achieve this.

Currently, `abagen` supports nine options for normalizing data:

1. *Centering*,
2. *Z-score*,
3. *Min-max*,
4. *Sigmoid*,
5. *Scaled sigmoid*,
6. *Scaled sigmoid quantiles*,
7. *Robust sigmoid*,
8. *Scaled robust sigmoid*, and
9. *Mixed sigmoid*

Most of the options have been used at various points throughout the published record, so while there is no "right" choice we do encourage using the default option (*scaled robust sigmoid*) due to recent work by Arnatkevičiūte et al., 2019 showing that it is—as the name might suggest—robust to outlier effects commonly observed in microarray data.

We describe all the methods in detail here; these can be implemented by passing the `sample_norm` and `gene_norm` keyword arguments to `abagen.get_expression_data()`. For a selection of references to published works that have used these different methods please see the documentation of `abagen.normalize_expression()`.

`sample_norm` vs `gene_norm`

Microarray expression data can be normalized in two directions:

1. Each sample can be normalized across all genes, or
2. Each gene can be normalized across all samples

These different forms of normalization are controlled by two parameters in the `abagen.get_expression_data()` function: `sample_norm` and `gene_norm`. Note that normalization of each sample across all genes occurs before normalization of each gene across all samples.

Both parameters can accept the same arguments (detailed below), and both are turned on by default.

Normalization methods

Centering

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='center')
```

Microarray values are centered with:

$$x_{norm} = x_y - \bar{x}$$

where $\bar{x}$ is the mean of the microarray expression values.

Z-score

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='zscore')
```

Microarray values are normalized using a basic z-score function:

$$x_{norm} = \frac{x_y - \bar{x}}{\sigma_x}$$

where $\bar{x}$ is the mean and σ_x is the sample standard deviation of the microarray expression values.

Min-max

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='minmax')
```

Microarray values are rescaled to the unit interval with:

$$x_{norm} = \frac{x_y - \min(x)}{\max(x) - \min(x)}$$

Sigmoid

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='sigmoid')
```

Microarray values are normalized using a general sigmoid function:

$$x_y = \frac{1}{1 + \exp\left(\frac{-(x_y - \bar{x})}{\sigma_x}\right)}$$

where $\bar{x}$ is the mean and σ_x is the sample standard deviation of the microarray expression values.

Scaled sigmoid

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='scaled_sigmoid')
```

Microarray values are processed with the *sigmoid* function and then rescaled to the unit interval with the *min-max* function.

Scaled sigmoid quantiles

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='scaled_sigmoid_
↪quantiles')
```

Input data are clipped to the 5th and 95th percentiles before being processed with the *scaled sigmoid* transform. The clipped distribution is only used for calculation of $\bar{x}$ and σ_x; the full (i.e., unclipped) distribution is processed through the transformation.

Robust sigmoid

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='robust_sigmoid')
```

Microarray values are normalized using a robust sigmoid function:

$$x_y = \frac{1}{1 + \exp\left(\frac{-(x_y - \langle x \rangle)}{\text{IQR}_x}\right)}$$

where $\langle x \rangle$ is the median and IQR_x is the normalized interquartile range of the microarray expression values given as:

$$erf\text{IQR}_x = \frac{Q_3 - Q1}{2 \cdot \sqrt{2} \cdot^{-1}\left(\frac{1}{2}\right)} \approx \frac{Q_3 - Q_1}{1.35}$$

Scaled robust sigmoid

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='scaled_robust_sigmoid
↪')
```

Microarray values are processed with the *robust sigmoid* function and then rescaled to the unit interval with the *min-max* function.

Mixed sigmoid

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm='mixed_sigmoid')
```

Microarray values are processed with the *scaled sigmoid* function when their interquartile range is 0; otherwise, they are processed with the *scaled robust sigmoid* function.

No normalization

```
>>> abagen.get_expression_data(atlas['image'], {sample,gene}_norm=None)
```

Providing `None` to the `sample_norm` and `gene_norm` parameters will prevent any normalization procedure from being performed on the data. Use this with caution!

Note: Some of the more advanced methods described on this page were initially proposed in:

> Fulcher, B.F., Little, M.A., & Jones, N.S. (2013). Highly comparative time-series analysis: the empirical structure of time series and their methods. *Journal of the Royal Society Interface*, 10(83), 20130048.

If using one of these methods please consider citing this paper in your work!

Applicable methods: *robust sigmoid*, *scaled robust sigmoid*, *mixed sigmoid*

Normalizing only matched samples

While sample normalization is _always_ performed across all genes, you can control which samples are used when performing gene normalization. By default, only those samples matched to regions in the provided atlas are used in the normalization process (controllable via the `norm_matched` parameter):

```
>>> abagen.get_expression_data(atlas['image'], norm_matched=True)
```

However, when a smaller atlas is provided with only a few regions, normalizing over just those samples matched to the atlas can be less desirable. To make it so that all available samples are used instead of only those matched, set `norm_matched` to `False`:

```
>>> abagen.get_expression_data(atlas['image'], norm_matched=False)
```

Warning: Given the preponderence of parameters in *`abagen.get_expression_data()`* it is perhaps unsurprising that they will interact with one another. However, it is worth pointing out that `norm_matched` will interact with the `missing` parameter in a relatively surprising manner (hence why we feel the need to make this note). This is due to the order in which sample-to-region matching, normalization, and "missing" regions are handled: when `norm_matched` is set to `True` all samples not matched to regions are removed prior to normalization. As such, if the `missing` parameter is set, the program is only able to fill in missing regions with samples that had already been assigned to other regions. If, instead, `norm_matched=False` and the `missing` parameter is set, the program can use the full range of samples to fill in missing regions. For this reason, we suggest using `norm_matched=False` when also setting the `missing` parameter; however, we do not impose a restriction on this.

Normalizing within structural classes

There are known differences in microarray expression between broad structural designations (e.g. cortex, subcortex/brainstem, cerebellum). As such, it may occasionally be desirable to constrain normalization such that the procedure is performed separately for each structural designation. This process can be controlled via the `norm_structures` parameter:

```
>>> abagen.get_expression_data(atlas['image'], norm_structures=True)
```

By default, this parameter is set to `False` and normalization uses all available samples. Note that changing this parameter will _dramatically_ modify the returned expression information, so use with caution. For obvious reasons this parameter will interact heavily with the `norm_matched` parameter described above.

7.4.7 Sample aggregation options

The primary goal of *`abagen.get_expression_data()`* is to allow users to aggregate the ~3,700 disparate tissue samples from the Allen Human Brain Atlas into regions of interest defined by an atlas or parcellation file. However, there exist several options for exactly *how* to aggregate samples within each region of the specified atlas.

These options are controlled via two parameters to *`abagen.get_expression_data()`*: `region_agg` and `agg_metric`. We discuss both parameters and the different options available to each below.

The *region_agg* parameter

This parameter determines how samples are aggregated together to generate the expression values for a region. It can take two values: *'donors'* or *'samples'*.

If set to *'donors'*, expression values for all samples assigned to a region are aggregated independently for each donor and *then* aggregated across donors. If set to *'samples'*, expression values for all samples for all donors assigned to a region are aggregated simultaneously.

The *agg_metric* parameter

This parameter determines the actual metric used for aggregating samples into regional expression values. It can be set to any callable function (as long as that function accepts the keyword `axis` argument), but generally either *'mean'* (the default) or *'median'* will suffice.

7.4.8 Using a binary mask

Basic usage

Sometimes, you're not interested in aggregating microarray expression samples within regions of an atlas—you want the actual, sample-level data instead. In this case, we provides the *`abagen.get_samples_in_mask()`* function.

To demonstrate how this works, we'll first make a brainmask for the left parahippocampal gyrus using the region definition from the Desikan-Killiany atlas:

```
>>> import nibabel as nib
>>> import pandas as pd
>>> atlas = abagen.fetch_desikan_killiany()
>>> dk = nib.load(atlas['image'])
>>> info = pd.read_csv(atlas['info'])
```

(continues on next page)

(continued from previous page)

```
>>> phg = int(info.query('label == "parahippocampal" & hemisphere == "L"')['id'])
>>> img = dk.__class__(dk.dataobj[:] == phg, dk.affine, dk.header)
```

We can then use this mask to obtain all the microarray samples that fall within its boundaries:

```
>>> expression, coords = abagen.get_samples_in_mask(mask=img)
```

abagen.get_samples_in_mask() returns two objects: (1) the the samples x gene expression matrix (`exp`), and (2) an array of MNI coordinates for those samples (`coords`). Because this is using *abagen.get_expression_data()* under the hood, the returned expression data have been preprocessed (i.e., filtered, normalized) according to that workflow. As such, you can provide all the same parameters and keyword arguments to *abagen.get_samples_in_mask()* as you can to *abagen.get_expression_data()* (with the exception of `atlas` which is superseded by `mask` and `region_agg`/`agg_metric` which will be ignored). Refer to the *API documentation*) for more details!

Since the returned expression dataframe is a samples x gene matrix (rather than regions x gene), the index of the dataframe corresponds to the unique well ID of the relevant sample (rather than the atlas region):

```
>>> print(expression)
gene_symbol       A1BG  A1BG-AS1       A2M  ...       ZYX     ZZEF1      ZZZ3
well_id                                     ...
2850          0.654914  0.234039  0.283280  ...  0.020379  0.228080  0.000000
998           0.428705  0.375819  0.457741  ...  0.254195  0.315383  0.502122
990           0.400673  0.409852  0.561666  ...  0.270064  0.397740  0.522261
...                ...       ...       ...  ...       ...       ...       ...
159226055     0.418706  0.751837  0.087808  ...  0.651541  0.410095  0.462773
159226117     0.533079  0.773214  0.265615  ...  0.441826  0.389615  0.455249
158158343     0.362038  0.553050  0.314730  ...  0.346605  0.261426  0.337738

[43 rows x 15633 columns]
```

This allows you to match up the samples with additional data provided by the AHBA (e.g., ontological information) as desired.

Get ALL the samples

If you want all of the available processed samples rather than only those within a given mask you can call the function without providing an explicit mask (this is the default when no `mask` parameter is passed):

```
>>> expression, coords = abagen.get_samples_in_mask(mask=None)
```

This will return all samples (after dropping those where the listed MNI coordinates don't match the listed hemisphere designation, etc.).

7.4.9 Generating reporting methods

There are a lot of options and parameters to choose from when processing data with `abagen.get_expression_data()` and while we've attempted to select reasonable defaults, we don't want to limit your options—you are free to pick and choose any combination of inputs to process the AHBA data! That said, we also wanted to make it easy for you to report *exactly* what was done to the data based on the parameters you choose, so to that end we have added the `return_report` parameter to `abagen.get_expression_data()`.

When `return_report=True`, the workflow will return an extra output. That is, in addition to the default regional microarray expression dataframe, a string will be returned that describes, in detail, all the processing that was done to the AHBA in the process of generating the expression matrix. We have tried to write this in such a way that you can simply copy-and-paste the provided text into the methods section of a paper, though you are of course free to edit it as you see fit (though if you feel edits are necessary please let us know and we can modify the generation more permanently!).

Example report

A report can be generated with:

```
>>> expression, report = abagen.get_expression_data(atlas['image'], atlas['info'],
...                                                 return_report=True)
```

Alternatively, you can use the `abagen.reporting` module to generate a report directly without having to re-run the entire pipeline. (Note that the `Report` class accepts (nearly) all the same parameters as the `get_expression_data()` workflow.)

```
>>> from abagen import reporting
>>> generator = reporting.Report(atlas['image'], atlas['info'])
>>> report = generator.gen_report()
```

The returned `report` (with default parameters) will look something like this example:

> *Regional microarry expression data were obtained from 6 post-mortem brains (1 female, ages 24.0–57.0, 42.50 +/- 13.38) provided by the Allen Human Brain Atlas (AHBA, https://human.brain-map.org; [H2012N]). Data were processed with the abagen toolbox (version X.Y; https://github.com/rmarkello/abagen) using a 83-region volumetric atlas in MNI space.*
>
> *First, microarray probes were reannotated using data provided by [A2019N]; probes not matched to a valid Entrez ID were discarded. Next, probes were filtered based on their expression intensity relative to background noise [Q2002N], such that probes with intensity less than the background in >=50.00% of samples across donors were discarded. When multiple probes indexed the expression of the same gene, we selected and used the probe with the most consistent pattern of regional variation across donors (i.e., differential stability; [H2015N]), calculated with:*
>
> $$\Delta_S(p) = \frac{1}{\binom{N}{2}} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \rho[B_i(p), B_j(p)]$$
>
> *where ρ is Spearman's rank correlation of the expression of a single probe, p, across regions in two donor brains B_i and B_j, and N is the total number of donors. Here, regions correspond to the structural designations provided in the ontology from the AHBA.*
>
> *The MNI coordinates of tissue samples were updated to those generated via non-linear registration using the Advanced Normalization Tools (ANTs; https://github.com/chrisfilo/alleninf). Samples were assigned to brain regions in the provided atlas if their MNI coordinates were within 2 mm of a given parcel. To reduce the potential for misassignment, sample-to-region matching was constrained by hemisphere and gross structural divisions (i.e., cortex, subcortex/brainstem, and cerebellum, such that e.g., a sample in*

the left cortex could only be assigned to an atlas parcel in the left cortex; [A2019N]). All tissue samples not assigned to a brain region in the provided atlas were discarded.

Inter-subject variation was addressed by normalizing tissue sample expression values across genes using a robust sigmoid function [F2013J]:

$$x_{norm} = \frac{1}{1+\exp(-\frac{(x-\langle x \rangle)}{\text{IQR}_x})}$$

where $\langle x \rangle$ *is the median and* IQR_x *is the normalized interquartile range of the expression of a single tissue sample across genes. Normalized expression values were then rescaled to the unit interval:*

$$x_{scaled} = \frac{x_{norm} - \min(x_{norm})}{\max(x_{norm}) - \min(x_{norm})}$$

Gene expression values were then normalized across tissue samples using an identical procedure. Samples assigned to the same brain region were averaged separately for each donor and then across donors.

REFERENCES ---------- [A2019N]: Arnatkeviciūtė, A., Fulcher, B. D., & Fornito, A. (2019). A practical guide to linking brain-wide gene expression and neuroimaging data. Neuroimage, 189, 353-367. [F2013J]: Fulcher, B. D., Little, M. A., & Jones, N. S. (2013). Highly comparative time-series analysis: the empirical structure of time series and their methods. Journal of the Royal Society Interface, 10(83), 20130048. [H2012N]: Hawrylycz, M. J., Lein, E. S., Guillozet-Bongaarts, A. L., Shen, E. H., Ng, L., Miller, J. A., … & Jones, A. R. (2012). An anatomically comprehensive atlas of the adult human brain transcriptome. Nature, 489(7416), 391-399. [H2015N]: Hawrylycz, M., Miller, J. A., Menon, V., Feng, D., Dolbeare, T., Guillozet-Bongaarts, A. L., … & Lein, E. (2015). Canonical genetic signatures of the adult human brain. Nature Neuroscience, 18(12), 1832.

[Q2002N]: Quackenbush, J. (2002). Microarray data normalization and transformation. Nature Genetics, 32(4), 496-501.

Note that due to text formatting limitations in Python, relevant equations used for e.g., normalizing the expression data will be provided in LaTeX format (i.e., surrounded by *$$* characters and with TeX math commands).

Important: Please note that we explicitly release all text in the `abagen.reporting` module (used to generate the above-referenced reports) under a CC0 license such that it can be used in manuscripts without modification.

7.5 Getting involved

Please refer to our contributing guidelines for a practical walkthrough on how to get involved and start contributing to `abagen`.

7.6 Citing abagen

Note: We strongly encourage you to use the *automatically-generated methods reports* built into `abagen`, which will provide a list of citations based on your selected processing options!

We're thrilled you've found `abagen` useful in your work! Please cite the following manuscripts when referencing your use of the toolbox:

1. Markello, RD, Arnatkeviciūtė, A, Poline, J-B, Fulcher, BD, Fornito, A, & Misic, B. (2021). Standardizing workflows in imaging transcriptomics with the abagen toolbox. Biorxiv. doi:10.1101/2021.07.08.451635

2. Arnatkevičiūtė, A, Fulcher, BD, & Fornito, A. (2019). A practical guide to linking brain-wide gene expression and neuroimaging data. NeuroImage, 189, 353-367. doi:10.1016/j.neuroimage.2019.01.011
3. Hawrylycz, MJ, Lein, ES, Guillozet-Bongaarts, AL, Shen, EH, Ng, L, Miller, JA, . . . , & Jones, AR. (2012). An anatomically comprehensive atlas of the adult human brain transcriptome. Nature, 489(7416), 391–399. doi:10.1038/nature11405

Additionally, to cite the specific version of the toolbox used in your analyses you can use the following Zenodo reference:

Note that this will always point to the most recent `abagen` release; for older releases please refer to the Zenodo listing.

For more information about why citing software is important please refer to this article from the Software Sustainability Institute.

7.7 Reference API

List of modules

7.7.1 `abagen.allen` - Primary workflows

Functions for mapping AHBA microarray dataset to atlases and and parcellations

abagen.get_expression_data(atlas[, . . .])	Assigns microarray expression data to ROIs defined in *atlas*
abagen.get_samples_in_mask([mask])	Returns preprocessed microarray expression data for samples in *mask*
abagen.get_interpolated_map(genes, mask[, . . .])	Generates dense (i.e., interpolated) expression maps for *genes*

abagen.get_expression_data

`abagen.get_expression_data(`*atlas*, *atlas_info=None*, ***, *ibf_threshold=0.5*, *probe_selection='diff_stability'*, *donor_probes='aggregate'*, *sim_threshold=None*, *lr_mirror=None*, *exact=None*, *missing=None*, *tolerance=2*, *sample_norm='srs'*, *gene_norm='srs'*, *norm_matched=True*, *norm_structures=False*, *region_agg='donors'*, *agg_metric='mean'*, *corrected_mni=True*, *reannotated=True*, *return_counts=False*, *return_donors=False*, *return_report=False*, *donors='all'*, *data_dir=None*, *verbose=0*, *n_proc=1*`)`

Assigns microarray expression data to ROIs defined in *atlas*

This function aims to provide a workflow for generating pre-processed, microarray expression data from the Allen Human Brain Atlas ([A2]) for abitrary *atlas* designations. First, some basic filtering of genetic probes is performed, including:

1. Intensity-based filtering of microarray probes to remove probes that do not exceed a certain level of background noise (specified via the *ibf_threshold* parameter),
2. Selection of a single, representative probe (or collapsing across probes) for each gene, specified via the *probe_selection* parameter (and influenced by the *donor_probes* parameter), and
3. Optional mirroring of the tissue samples across the left/right hemisphere boundary, as specified via the *lr_mirror* parameter (turned off by default).

Tissue samples are then matched to parcels in the defined *atlas* for each donor. If *atlas_info* is provided then this matching is constrained by both hemisphere and tissue class designation (e.g., cortical samples from the left hemisphere are only matched to ROIs in the left cortex, subcortical samples from the right hemisphere are only matched to ROIs in the left subcortex); see the *atlas_info* parameter description for more information.

Matching of microarray samples to parcels in *atlas* is done via a multi- step process:

1. Determine if the sample falls directly within a parcel,
2. Check to see if there are nearby parcels by slowly expanding the search space to include nearby voxels, up to a specified distance (specified via the *tolerance* parameter),
3. If there are multiple nearby parcels, the sample is assigned to the closest parcel, as determined by the parcel centroid.

If at any step a sample can be assigned to a parcel the matching process is terminated. When the provided atlas is not volumetric (i.e., surface-based) the samples are simply matched to the nearest vertex, and *tolerance* is used as a standard deviation threshold. More control over the sample matching can be obtained by setting the *exact* parameter; see the parameter description for more information.

Once all samples have been matched to parcels for all supplied donors, the microarray expression data are optionally normalized via the provided *sample_norm* and *gene_norm* functions (which are influenced by the *norm_matched* and *norm_structures* parameters) before being aggregated across donors via the supplied *region_agg* and *agg_metric* parameters.

Parameters

- **atlas** (*niimg-like object or dict*) – A parcellation image in MNI space or a tuple of GIFTI images in fsaverage5 space, where each parcel is identified by a unique integer ID. Alternatively, a dictionary where keys are donor IDs and values are parcellation images (or surfaces) in the native space of each donor.
- **atlas_info** (*os.PathLike or pandas.DataFrame, optional*) – Filepath to or preloaded dataframe containing information about *atlas*. Must have at least columns 'id', 'hemisphere', and 'structure' containing information mapping atlas IDs to hemisphere (i.e, "L", "R", "B") and broad structural class (i.e., "cortex", "subcortex/brainstem", "cerebellum"). If provided, this will constrain matching of tissue samples to regions in *atlas*. If *atlas* is a

tuple of GIFTI images with valid label tables this will be intuited from the data. Default: None

- **ibf_threshold** (*[0, 1] float, optional*) – Threshold for intensity-based filtering. This number specifies the ratio of samples, across all supplied donors, for which a probe must have signal significantly greater than background noise in order to be retained. Default: 0.5
- **probe_selection** (*str, optional*) – Selection method for subsetting (or collapsing across) probes that index the same gene. Must be one of 'average', 'max_intensity', 'max_variance', 'pc_loading', 'corr_variance', 'corr_intensity', or 'diff_stability', 'rnaseq'; see Notes for more information on different options. Default: 'diff_stability'
- **donor_probes** (*{'aggregate', 'independent', 'common'}, optional*) – Whether specified *probe_selection* method should be performed with microarray data from all donors ('aggregate'), independently for each donor ('independent'), or based on the most common selected probe across donors ('common'). Not all combinations of *probe_selection* and *donor_probes* methods are viable. Default: 'aggregate'
- **sim_threshold** (*(0, inf) float, optional*) – Threshold for inter-areal similarity filtering. Samples are correlated across probes and those samples with a total correlation less than *sim_threshold* standard deviations below the mean across samples are excluded from futher analysis. If not specified no filtering is performed. Default: None
- **lr_mirror** (*{None, 'bidirectional', 'leftright', 'rightleft'}, optional*) – Whether to mirror microarray expression samples across hemispheres to increase spatial coverage. Using 'bidirectional' will mirror samples across both hemispheres, 'leftright' will mirror samples in the left hemisphere to the right, and 'rightleft' will mirror the right to the left. Default: None
- **missing** (*{'centroids', 'interpolate', None}, optional*) – How to handle regions in *atlas* that are not assigned any tissue samples. If 'centroids', any empty regions will be assigned the expression value of the nearest tissue sample (defined as the sample with the closest Euclidean distance to the parcel centroid). If 'interpolate', expression values will be interpolated in the empty regions by assigning every node in the region the expression of the nearest sample and taking a weighted (inverse distance) average. If not specified empty regions will be returned with expression values of NaN. Default: None
- **tolerance** (*int, optional*) – Distance (in mm) that a sample must be from a parcel for it to be matched to that parcel. If *atlas* is a tuple of surface files then this measure is a standard deviation threshold (i.e., samples greater than *tolerance* SDs away from the mean matched distance are ignored). Default: 2
- **sample_norm** (*{'rs', 'srs', 'minmax', 'center', 'zscore', None}, optional*) – Method by which to normalize microarray expression values for each sample. Expression values are normalized separately for each sample and donor across all genes; see Notes for more information on different methods. If None is specified then no normalization is performed. Default: 'srs'
- **gene_norm** (*{'rs', 'srs', 'minmax', 'center', 'zscore', None}, optional*) – Method by which to normalize microarray expression values for each donor. Expression values are normalized separately for each gene and donor across all samples; see Notes for more information on different methods. If None is specified then no normalization is performed. Default: 'srs'
- **norm_matched** (*bool, optional*) – Whether to perform gene normalization (*gene_norm*) across only those samples matched to regions in *atlas* instead of all available samples. If *atlas* is very small (i.e., only a few regions of interest), using *norm_matched=False* is suggested. Default: True

- **norm_structures** (*bool, optional*) – Whether to perform gene normalization (*gene_norm*) within structural classes (i.e., 'cortex', 'subcortex/brainstem', 'cerebellum') instead of across all available samples. Default: False
- **region_agg** (*{'samples', 'donors'}, optional*) – When multiple samples are identified as belonging to a region in *atlas* this determines how they are aggegated. If 'samples', expression data from all samples for all donors assigned to a given region are combined. If 'donors', expression values for all samples assigned to a given region are combined independently for each donor before being combined across donors. See *agg_metric* for mechanism by which samples are combined. Default: 'donors'
- **agg_metric** (*{'mean', 'median'} or callable, optional*) – Mechanism by which to reduce sample-level expression data into region- level expression (see *region_agg*). If a callable, should be able to accept an *N*-dimensional input and the *axis* keyword argument and return an *N-1*-dimensional output. Default: 'mean'
- **corrected_mni** (*bool, optional*) – Whether to use the "corrected" MNI coordinates shipped with the *alleninf* package instead of the coordinates provided with the AHBA data when matching tissue samples to anatomical regions. Default: True
- **reannotated** (*bool, optional*) – Whether to use reannotated probe information provided by [A1] instead of the default probe information from the AHBA dataset. Using reannotated information will discard probes that could not be reliably matched to genes. Default: True
- **return_counts** (*bool, optional*) – Whether to return dataframe containing information on how many samples were assigned to each parcel in *atlas* for each donor. Default: False
- **return_donors** (*bool, optional*) – Whether to return donor-level expression arrays instead of aggregating expression across donors with provided *agg_metric*. Default: False
- **return_report** (*bool, optional*) – Whether to return a string containing longform text describing the processing procedures used to generate the *expression* DataFrames returned by this function. Default: False
- **donors** (*list, optional*) – List of donors to use as sources of expression data. Can be either donor numbers or UID. If not specified will use all available donors. Note that donors '9861' and '10021' have samples from both left + right hemispheres; all other donors have samples from the left hemisphere only. Default: 'all'
- **data_dir** (*os.PathLike, optional*) – Directory where expression data should be downloaded (if it does not already exist) / loaded. If not specified will use the current directory. Default: None
- **verbose** (*int, optional*) – Specifies verbosity of status messages to display during workflow. Higher numbers increase verbosity of messages while zero suppresses all messages. Default: 1
- **n_proc** (*int, optional*) – Number of processors to use to download AHBA data. Can parallelize up to six times. Default: 1

Returns

- **expression** (*(R, G) pandas.DataFrame*) – Microarray expression for *R* regions in *atlas* for *G* genes, aggregated across donors, where the index corresponds to the unique integer IDs of *atlas* and the columns are gene names. If `return_donors=True` then this is a list of (R, G) dataframes, one for each donor.
- **counts** (*(R, D) pandas.DataFrame*) – Number of samples assigned to each of *R* regions in *atlas* for each of *D* donors (if multiple donors were specified); only returned if `return_counts=True`.

- **report** (*str*) – Methods describing processing procedures implemented to generate *expression*, suitable to be used in a manuscript Methods section. Only returned if `return_report=True`.

Notes

The following methods can be used for collapsing across probes when multiple probes are available for the same gene:

1. `probe_selection='average'`

Takes the average of expression data across all probes indexing the same gene. Providing 'mean' as the input method will return the same thing. This method can only be used when *donor_probes='aggregate'*.

2. `probe_selection='max_intensity'`

Selects the probe with the maximum average expression across samples from all donors.

3. `probe_selection='max_variance'`

Selects the probe with the maximum variance in expression across samples from all donors.

4. `probe_selection='pc_loading'`

Selects the probe with the maximum loading along the first principal component of a decomposition performed across samples from all donors.

5. `probe_selection='corr_intensity'`

Selects the probe with the maximum correlation to other probes from the same gene when >2 probes exist; otherwise, uses the same procedure as *max_intensity*.

6. `probe_selection='corr_variance'`

Selects the probe with the maximum correlation to other probes from the same gene when >2 probes exist; otherwise, uses the same procedure as *max_varance*.

7. `probe_selection='diff_stability'`

Selects the probe with the most consistent pattern of regional variation across donors (i.e., the highest average correlation across brain regions between all pairs of donors). This method can only be used when *donor_probes='aggregate'*.

8. `method='rnaseq'`

Selects probes with most consistent pattern of regional variation to RNAseq data (across the two donors with RNAseq data). This method can only be used when *donor_probes='aggregate'*.

Note that for incompatible combinations of *probe_selection* and *donor_probes* (as detailed above), the *probe_selection choice will take precedence. For example, providing ``probe_selection='diff_stability'``* and `donor_probes='independent'` will cause *donor_probes* to be reset to *'aggregate'*.

The following methods can be used for normalizing microarray expression values prior to aggregating:

1. `{sample,gene}_norm=='rs'`

Uses a robust sigmoid function as in [A3] to normalize values

2. `{sample,gene}_norm='srs'`

Same as 'rs' but scales output to the unit normal (i.e., range 0-1)

3. `{sample,gene}_norm='minmax'`

Scales data to the unit normal (i.e., range 0-1)

4. `{sample,gene}_norm='center'`

Removes the mean of expression values

5. `{sample,gene}_norm='zscore'`

Applies a basic z-score (subtract mean, divide by standard deviation); uses degrees of freedom equal to one for standard deviation

References

abagen.get_samples_in_mask

`abagen.`**`get_samples_in_mask`**(*mask=None*, ***kwargs*)

Returns preprocessed microarray expression data for samples in *mask*

Uses the same processing workflow as `abagen.get_expression_data()` but instead of aggregating samples within regions simply returns sample-level expression data for all samples that fall within boundaries of *mask*.

Parameters

- **mask** (*niimg-like object or dict, optional*) – A mask image in MNI space or a tuple of GIFTI images in fsaverage5 space (where 0 is the background). Alternatively, a dictionary where keys are donor IDs and values are mask images (or surfaces) in the native space of each donor. If not supplied, all available samples will be returned. Default: None
- **kwargs** (*key-value pairs*) – All key-value pairs from `abagen.get_expression_data()` except for: *atlas*, *atlas_info*, *region_agg*, and *agg_metric*, which will be ignored. If *atlas* is supplied instead of *mask* then *atlas* will be used instead as a modified binary image. If both *atlas* and *mask* are supplied then *mask* will be used

Returns

- **expression** (*(S, G) pandas.DataFrame*) – Microarray expression for *S* samples for *G* genes, aggregated across donors, where the columns are gene names
- **coords** (*(S,) numpy.ndarray*) – MNI coordinates of samples in *expression*. Even if donor-specific masks are provided MNI coordinates will be returned to ensure comparability between subjects

abagen.get_interpolated_map

`abagen.`**`get_interpolated_map`**(*genes*, *mask*, *n_neighbors=10*, ***kwargs*)

Generates dense (i.e., interpolated) expression maps for *genes*

Uses k-nearest neighbors regression to estimate values at every voxel or vertex in the supplied *mask*. Note that by default *lr_mirror* is set to 'bidirectional' and *norm_matched* is set to False, unless explicitly specified.

Parameters

- **genes** (*(G,) list-of-str*) – List of gene acronyms for which dense maps are desired
- **mask** (*niimg-like object, optional*) – A mask image in MNI space or a tuple of GIFTI images in fsaverage5 space (where 0 is the background)
- **n_neighbors** (*int, optional*) – Number of neighboring tissue samples to use when interpolating data in dense map. Default: 10

- **kwargs** (*key-value pairs*) – All key-value pairs from `abagen.get_expression_data()` except for: *atlas*, *atlas_info*, *region_agg*, and *agg_metric*, which will be ignored.

Returns **dense** – Dictionary where keys are *genes* and values are dense maps in space of provided *mask*

Return type (G,) dict

7.7.2 `abagen.datasets` - Fetching AHBA datasets

Functions for fetching data relevant to the Allen Brain Atlas human microarray dataset

abagen.fetch_desikan_killiany([native, surface])	Fetches Desikan-Killiany atlas shipped with *abagen*
abagen.fetch_donor_info()	Returns dataframe with donor demographic information
abagen.fetch_freesurfer([data_dir, donors, ...])	Downloads FreeSurfer reconstructions of the Allen Human Brain Atlas MRIs
abagen.fetch_gene_group(group)	Return list of gene acronyms belonging to provided *group*
abagen.fetch_microarray([data_dir, donors, ...])	Downloads the Allen Human Brain Atlas microarray expression dataset
abagen.fetch_raw_mri([data_dir, donors, ...])	Downloads the “raw” Allen Human Brain Atlas T1w/T2w MRI images
abagen.fetch_rnaseq([data_dir, donors, ...])	Downloads RNA-sequencing data from the Allen Human Brain Atlas
abagen.datasets.fetch_fsaverage5([load])	Fetches and optionally loads fsaverage5 surface
abagen.datasets.fetch_fsnative(donors[, ...])	Fetches and optionally loads fsnative surface of *donor*

abagen.fetch_desikan_killiany

abagen.**fetch_desikan_killiany**(*native=False*, *surface=False*, **args*, ***kwargs*)

Fetches Desikan-Killiany atlas shipped with *abagen*

Parameters

- **native** (*bool, optional*) – Whether to return individualized atlases in donor native space. Default: False
- **surface** (*bool, optional*) – Whether to return surface instead of volumetric parcellation. This option is currently incompatible with `native=True`; instead, refer to `abagen.datasets.fetch_freesurfer()` for donor-specific surface atlases. Default: False

Returns **atlas** – Dictionary with keys [‘image’, ‘info’] pointing to atlas image and information files. If `native` then ‘image’ is a dictionary where keys are donor IDs and values are image paths. If `surface` then ‘image’ is a tuple of GIFTI files (.label.gii.gz)

Return type dict

References

Desikan, R. S., Ségonne, F., Fischl, B., Quinn, B. T., Dickerson, B. C., Blacker, D., … & Albert, M. S. (2006). An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest. Neuroimage, 31(3), 968-980.

Examples

```
>>> import abagen
>>> atlas = abagen.fetch_desikan_killiany()
>>> print(atlas['image'])
/.../abagen/data/atlas-desikankilliany.nii.gz
>>> print(atlas['info'])
/.../abagen/data/atlas-desikankilliany.csv
```

When fetching native-space atlases, *atlas['image']* will be a dictionary where the keys are donor IDs and the values are paths to the donor-specific atlases:

```
>>> atlas = abagen.fetch_desikan_killiany(native=True)
>>> print(atlas['image'].keys())
dict_keys(['9861', '10021', '12876', '14380', '15496', '15697'])
>>> print(atlas['image']['9861'])
/.../abagen/data/native_dk/9861/atlas-desikankilliany.nii.gz
```

abagen.fetch_donor_info

abagen.**fetch_donor_info**()

Returns dataframe with donor demographic information

Returns info – With columns ['donor', 'age', 'sex', 'ethnicity', 'medical_conditions', 'post_mortem_interval_hours'] detailing basic demographic info about donors

Return type pandas.DataFrame

abagen.fetch_freesurfer

abagen.**fetch_freesurfer**(*data_dir=None*, *donors=None*, *resume=True*, *verbose=1*)

Downloads FreeSurfer reconstructions of the Allen Human Brain Atlas MRIs

Parameters

- **data_dir** (*str, optional*) – Directory where data should be downloaded and unpacked. Default: $HOME/ abagen-data
- **donors** (*list, optional*) – List of donors to download; can be either donor number or UID. Can also specify 'all' to download all available donors. Default: 12876
- **resume** (*bool, optional*) – Whether to resume download of a partly-downloaded file. Default: True
- **verbose** (*int, optional*) – Verbosity level (0 means no message). Default: 1

Returns freesurfer – Dictionary where keys are donor IDs and values are paths to FreeSurfer directories for requested *donors*

Return type dict

References

Romero-Garcia, R., Whitaker, K., Vasa, F., Seidlitz, J., Shinn, M., Fonagy, P., Jones, P., et al. (2017). Data supporting NSPN publication "Structural covariance networks are coupled to expression of genes enriched in supragranular layers of the human cortex " [Dataset]. https://doi.org/10.17863/CAM.11392

abagen.fetch_gene_group

abagen.**fetch_gene_group**(*group*)

Return list of gene acronyms belonging to provided *group*

Groups are defined as in [DS1]

Parameters **group** (*{'brain', 'neuron', 'oligodendrocyte', 'synaptome', 'layers'}*) – Desired gene group

Returns **genes** – List of gene acronyms

Return type list of str

References

abagen.fetch_microarray

abagen.**fetch_microarray**(*data_dir=None*, *donors=None*, *resume=True*, *verbose=1*, *convert=True*, *n_proc=1*)

Downloads the Allen Human Brain Atlas microarray expression dataset

Parameters

- **data_dir** (*str, optional*) – Directory where data should be downloaded and unpacked. Default: $HOME/ abagen-data
- **donors** (*list, optional*) – List of donors to download; can be either donor number or UID. Can also specify 'all' to download all available donors. Default: 12876
- **resume** (*bool, optional*) – Whether to resume download of a partly-downloaded file. Default: True
- **verbose** (*int, optional*) – Verbosity level (0 means no message). Default: 1
- **convert** (*bool, optional*) – Whether to convert downloaded CSV files into parquet format for faster loading in the future; only available if `fastparquet` and `python- snappy` are installed. Default: True
- **n_proc** (*int, optional*) – Number of processes to parallelize download if multiple donors are specified. Default: 1

Returns **data** – Two-level nested dictionary, where top-level keys are donor IDs and second-level keys are ['microarray', 'ontology', 'pacall', 'probes', 'annotation'], where corresponding values are lists of filepaths to downloaded CSV files.

Return type dict

References

Hawrylycz, M. J., Lein, E. S., Guillozet-Bongaarts, A. L., Shen, E. H., Ng, L., Miller, J. A., … & Abajian, C. (2012). An anatomically comprehensive atlas of the adult human brain transcriptome. Nature, 489(7416), 391.

abagen.fetch_raw_mri

`abagen.fetch_raw_mri`(*data_dir=None*, *donors=None*, *resume=True*, *verbose=1*)
Downloads the "raw" Allen Human Brain Atlas T1w/T2w MRI images

Parameters

- **data_dir** (*str, optional*) – Directory where data should be downloaded and unpacked. Default: $HOME/ abagen-data
- **donors** (*list, optional*) – List of donors to download; can be either donor number or UID. Can also specify 'all' to download all available donors. Default: 12876
- **resume** (*bool, optional*) – Whether to resume download of a partly-downloaded file. Default: True
- **verbose** (*int, optional*) – Verbosity level (0 means no message). Default: 1

Returns **mris** – Two-level nested dictionary, where top-level keys are donor IDs and second-level keys are ['t1w', 't2w'], where corresponding values are lists of filepaths to downloaded Nifti files

Return type dict

abagen.fetch_rnaseq

`abagen.fetch_rnaseq`(*data_dir=None*, *donors=None*, *resume=True*, *verbose=1*)
Downloads RNA-sequencing data from the Allen Human Brain Atlas

Parameters

- **data_dir** (*str, optional*) – Directory where data should be downloaded and unpacked. Default: current directory
- **donors** (*list, optional*) – List of donors to download; can be either donor number or UID. Can also specify 'all' to download all available donors (two). Default: 9861
- **resume** (*bool, optional*) – Whether to resume download of a partly-downloaded file. Default: True
- **verbose** (*int, optional*) – Verbosity level (0 means no message). Default: 1

Returns **data** – Two-level nested dictionary, where top-level keys are donor IDs and second-level keys are ['counts', 'tpm', 'ontology', 'genes', 'annotation'], where corresponding values are lists of filepaths to downloaded CSV files.

Return type dict

References

Hawrylycz, M. J., Lein, E. S., Guillozet-Bongaarts, A. L., Shen, E. H., Ng, L., Miller, J. A., … & Abajian, C. (2012). An anatomically comprehensive atlas of the adult human brain transcriptome. Nature, 489(7416), 391.

abagen.datasets.fetch_fsaverage5

`abagen.datasets.`**`fetch_fsaverage5`**(*load=True*)

Fetches and optionally loads fsaverage5 surface

Parameters **`load`** (`bool, optional`) – Whether to pre-load files. Default: True

Returns **brain** – If *load* is True, a namedtuple where each entry in the tuple is a hemisphere, represented as a namedtuple with fields ('vertices', 'faces'). If *load* is False, a namedtuple where entries are filepaths.

Return type namedtuple ('lh', 'rh')

abagen.datasets.fetch_fsnative

`abagen.datasets.`**`fetch_fsnative`**(*donors*, *surf='pial'*, *load=True*, *data_dir=None*, *resume=True*, *verbose=1*)

Fetches and optionally loads fsnative surface of *donor*

Parameters

- **`donors`** (`str or list-of-str`) – Donor(s) to download; can be either donor number or UID. Can also specify 'all' to download all available donors.
- **`surf`** (`{'orig', 'white', 'pial', 'inflated', 'sphere'}, optional`) – Which surface to load. Default: 'pial'
- **`load`** (`bool, optional`) – Whether to pre-load files. Default: True
- **`data_dir`** (`str, optional`) – Directory where data should be downloaded and unpacked. Default: $HOME/ abagen-data
- **`resume`** (`bool, optional`) – Whether to resume download of a partly-downloaded file. Default: True
- **`verbose`** (`int, optional`) – Verbosity level (0 means no message). Default: 1

Returns **brain** – If *load* is True, a namedtuple where each entry in the tuple is a hemisphere, represented as a namedtuple with fields ('vertices', 'faces'). If *load* is False, a namedtuple where entries are filepaths. If multiple donors are requested a dictionary is returned where keys are donor IDs.

Return type namedtuple ('lh', 'rh')

7.7.3 `abagen.images` - Image processing functions

abagen.leftify_atlas(atlas)	Zeroes out all ROIs in the right hemisphere of volumetric *atlas*
abagen.check_atlas(atlas[, atlas_info, ...])	Checks that *atlas* is a valid atlas
abagen.annot_to_gifti(atlas)	Converts FreeSurfer-style annotation file *atlas* to in-memory GIFTI image
abagen.relabel_gifti(atlas[, background, offset])	Updates GIFTI images so label IDs are consecutive across hemispheres

abagen.leftify_atlas

abagen.**leftify_atlas**(*atlas*)

Zeroes out all ROIs in the right hemisphere of volumetric *atlas*

Assumes that positive X values indicate the right hemisphere (e.g., RAS+ orientation) and that the X-origin is in the middle of the brain

Parameters **atlas** (*str or niimg-like*) – Filepath to or in-memory loaded image

Returns **atlas** – Loaded image with right hemisphere zeroed out

Return type niimg-like

abagen.check_atlas

abagen.**check_atlas**(*atlas*, *atlas_info=None*, *geometry=None*, *space=None*, *donor=None*, *data_dir=None*)

Checks that *atlas* is a valid atlas

Parameters

- **atlas** (*niimg-like object or (2,) tuple-of-GIFTI*) – Parcellation image or surface, where voxels / vertices belonging to a given parcel are identified with a unique integer ID
- **atlas_info** (*{os.PathLike, pandas.DataFrame, None}, optional*) – Filepath or dataframe containing information about *atlas*. Must have at least columns ['id', 'hemisphere', 'structure'] containing information mapping *atlas* IDs to hemisphere (i.e., "L", "R", "B") and broad structural class (i.e.., "cortex", "subcortex/brainstem", "cerebellum", "white matter", or "other"). Default: None
- **geometry** (*(2,) tuple-of-GIFTI, optional*) – Surfaces files defining geometry of *atlas*, if *atlas* is a tuple of GIFTI images. Default: None
- **space** (*{'fsaverage', 'fsnative', 'fslr'}, optional*) – If *geometry* is supplied, what space files are in. Default: None
- **donor** (*str, optional*) – If specified, indicates which donor the specified *atlas* belongs to. Only relevant when *atlas* is surface-based, to ensure the correct geometry files are fetched. Default: None (i.e., group-level atlas)
- **data_dir** (*str, optional*) – Directory where donor-specific FreeSurfer data should be downloaded and unpacked. Only used if provided *donor* is not None. Default: $HOME/abagen-data

Returns **atlas** – AtlasTree object with information about *atlas* and functionality for labelling coordinates

Return type `abagen.AtlasTree`

abagen.annot_to_gifti

`abagen.annot_to_gifti(atlas)`

Converts FreeSurfer-style annotation file *atlas* to in-memory GIFTI image

Parameters **annot** (`os.PathLike`) – Surface annotation file (.annot)

Returns **gifti** – Converted gifti image

Return type nib.gifti.GiftiImage

abagen.relabel_gifti

`abagen.relabel_gifti(atlas, background=['unknown', 'corpuscallosum', 'Background+FreeSurfer_Defined_Medial_Wall', '???'], offset=None)`

Updates GIFTI images so label IDs are consecutive across hemispheres

Parameters

- **atlas** (*(2,) tuple-of-str*) – Surface label files in GIFTI format (lh.label.gii, rh.label.gii)
- **background** (*list-of-str, optional*) – If provided, a list of IDs in *atlas* that should be set to 0 (the presumptive background value). Other IDs will be shifted so they are consecutive (i.e., 0–N). Default: *abagen.images.BACKGROUND*
- **offset** (*int, optional*) – What the lowest value in *atlas[1]* should be not including background value. If not specified it will be purely consecutive from *atlas[0]*. Default: None

Returns **relabelled** – Re-labelled *atlas* files

Return type (2,) tuple-of-nib.gifti.GiftiImage

7.7.4 `abagen.correct` - Post-processing corrections

Functions for processing and correcting gene expression data

`abagen.remove_distance`(coexpression, atlas)	Corrects for distance-dependent correlation effects in *coexpression*
`abagen.keep_stable_genes`(expression[, …])	Removes genes in *expression* with differential stability < *threshold*
`abagen.normalize_expression`(expression[, …])	Performs normalization on *expression* data

abagen.remove_distance

`abagen.remove_distance(coexpression, atlas, atlas_info=None, labels=None)`

Corrects for distance-dependent correlation effects in *coexpression*

Regresses Euclidean distance between regions in *atlas* from correlated gene expression array *coexpression*. If *atlas_info* is provided different connection types (e.g., cortex-cortex, cortex-subcortex, subcortex- subcortex) will be residualized independently.

Parameters

- **coexpression** (*(R x R) array_like*) – Correlated gene expression array, where *R* is the number of regions, as generated with e.g., *numpy.corrcoef(expression)*.
- **atlas** (*niimg-like object*) – A parcellation image in MNI space, where each parcel is identified by a unique integer ID
- **atlas_info** (*str or pandas.DataFrame, optional*) – Filepath to or pre-loaded dataframe containing information about *atlas*. Must have at least columns 'id', 'hemisphere', and 'structure' containing information mapping atlas IDs to hemisphere (i.e, "L", "R") and broad structural class (i.e., "cortex", "subcortex", "cerebellum"). Default: None
- **labels** (*(N,) array_like, optional*) – If only a subset *N* of the ROIs in *atlas* were used to generate the *coexpression* array this array should specify which to consider. Not specifying this may cause a ValueError if *atlas* and *atlas_info* do not match. Default: None

Returns **residualized** – Provided *coexpression* data residualized against spatial distance between region pairs

Return type (R, R) numpy.ndarray

abagen.keep_stable_genes

`abagen.`**`keep_stable_genes`**`(`*expression*, *threshold=0.9*, *percentile=True*, *rank=True*, *return_stability=False*`)`

Removes genes in *expression* with differential stability < *threshold*

Calculates the similarity of gene expression across brain regions for every pair of donors in *expression*. Similarity is averaged across donor pairs and genes whose mean similarity falls below *threshold* are removed.

Parameters

- **expression** (*list of (R, G) pandas.DataFrame*) – Where each entry is the microarray expression of *R* regions across *G* genes for a given donor
- **threshold** (*[0, 1] float, optional*) – Minimum required average similarity (e.g, correlation) across donors for a gene to be retained. Default: 0.1
- **percentile** (*bool, optional*) – Whether to treat *threshold* as a percentile instead of an absolute cutoff. For example, *threshold=0.9* and *percentile=True* would retain only those genes with a differential stability in the top 10% of all genes, whereas *percentile=False* would retain only those genes with differential stability > 0.9. Default: True
- **rank** (*bool, optional*) – Whether to calculate similarity as Spearman correlation instead of Pearson correlation. Default: True
- **return_stability** (*bool, optional*) – Whether to return stability estimates for each gene in addition to expression data. Default: False

Returns

- **expression** (*list of (R, Gr) pandas.DataFrame*) – Microarray expression for *R* regions across *Gr* genes, where *Gr* is the number of retained genes
- **stability** (*(G,) numpy.ndarray*) – Stability (average correlation) of each gene across pairs of donors. Only returned if `return_stability=True`

abagen.normalize_expression

abagen.**normalize_expression**(*expression*, *norm='srs'*, *structures=None*, *ignore_warn=False*)

Performs normalization on *expression* data

Parameters

- **expression** (*list of (S, G) pandas.DataFrame*) – Microarray expression data to be normalized, where *S* is samples (or regions) and *G* is genes
- **norm** (*str, optional*) – Function with which to normalize expression data. See Notes for more information on options. Default: 'scaled_robust_sigmoid'
- **structures** (*list of (S,) pandas.DataFrame*) – Structural designations of *S* samples (or regions) in *expression*. Index of provided data frames should be identical to *expression* and must have at least column 'structure'. If provided, normalization will be performed separately for each distinct structural class. Default: None
- **ignore_warn** (*bool, optional*) – Whether to suppress potential warnings raised by normalization. Default: False

Returns **normalized** – Data from *expression* normalized separately for each gene

Return type list of (S, G) pandas.DataFrame

Notes

The following methods can be used for normalizing gene expression values for each donor (adapted from [PC2]):

1. `norm='center'`

Removes the mean of data in each column. Aliased to 'demean'

2. `norm='zscore'`

Applies a basic z-score (subtract mean, divide by standard deviation) to each column; uses degrees of freedom equal to one for standard deviation

3. `norm='minmax'`

Scales data in each column to the unit normal (i.e., range 0-1)

4. `norm='sigmoid'`

Applies a sigmoidal transform function to normalize data in each column. Aliased to 'sig'

5. `norm='scaled_sigmoid'`

Combines 'sigmoid' and 'minmax'. Aliased to 'scaled_sig'

6. `norm='scaled_sigmoid_quantiles'`

Caps input data at the 5th and 95th percentiles before performing the 'scaled_sigmoid' transform. Aliased to 'scaled_sig_qnt'

7. `norm='robust_sigmoid'`

Uses a robust sigmoid function ([PC1]) to normalize data in each column. Aliased to 'rs' and 'rsig'

8. `norm='scaled_robust_sigmoid'`

Combines 'robust_sigmoid' and 'minmax'. Aliased to 'srs' and 'scaled_rsig'

9. `norm='mixed_sigmoid'`

Uses 'scaled_sigmoid' transform for columns where the IQR is 0; otherwise, uses the 'scaled_robust_sigmoid' transform. Aliased to 'mixed_sig'

10. `norm='batch'`

Uses a linear model to remove donor effects from data. Differs from other methods in that all donors are simultaneously fit to the same model and data are residualized based on estimated betas. Linear model includes the intercept but it is not removed during residualization

References

7.7.5 `abagen.matching` - Functions for matching samples

Structures and functions used for matching samples to atlas

abagen.AtlasTree(atlas[, coords, triangles, ...])	Representation of a parcellation as a cKDtree for NN lookups

abagen.AtlasTree

class abagen.**AtlasTree**(*atlas*, *coords=None*, ***, *triangles=None*, *atlas_info=None*, *group_atlas=True*)

Representation of a parcellation as a cKDtree for NN lookups

Parameters

- **atlas** (*(N,) niimg-like object or array_like*) – Volumetric (niimg-like) or array of parcellation labels. If providing an array you must provide *coords* as well
- **coords** (*(N, D) array_like, optional*) – Coordinates representing points in *atlas*. If provided it is assumed that *atlas* is a surface representation (i.e., if *atlas* is volumetric simply provide a niimg-like object and the coordinates will be derived from the data). Default: None
- **triangles** (*(F, 3) array_like, optional*) – If *coords* are derived from a surface mesh, this array contains the indices of the nodes comprising the mesh triangles. Default: None
- **atlas_info** (*{os.PathLike, pandas.DataFrame, None}, optional*) – Filepath or dataframe containing information about *atlas*. Must have at least columns ['id', 'hemisphere', 'structure'] containing information mapping *atlas* IDs to hemisphere (i.e., "L" or "R") and broad structural class (i.e.., "cortex", "subcortex/brainstem", "cerebellum", "white matter", or "other"). Default: None
- **group_atlas** (*bool, optional*) – Whether the provided *atlas* is a group atlas (in MNI space) or a donor-level atlas (in native space). This will have an impact on how provided sample coordinates are handled. Default: True

property atlas

Returns values of provided atlas

property atlas_info

Returns atlas info dataframe, if it exists

property centroids

Return centroids of parcels in *self.atlas*

property coords

Returns coordinates of underlying cKDTree

`fill_label`(*annotation*, *label*, *return_dist=False*)
Assigns a sample in *annotation* to every node of *label* in atlas

Parameters

- **annotation** (`(S, 3) array_like`) – At a minimum, an array of XYZ coordinates must be provided. If a full annotation dataframe is provided, then information from the data frame (i.e., on hemisphere + structural assignments of tissue samples) is used to constrain matching of samples (if *self.atlas_info* is not None).
- **label** (`int`) – Which label in *self.atlas* should be filled
- **return_dist** (`bool, optional`) – Whether to also return distance to mapped samples

Returns

- **samples** (*(L,) np.ndarray*) – ID of sample mapped to all *L* nodes in *label* of atlas
- **distance** (*(L,) np.ndarray*) – Distances of matched samples to nodes in *label*. Only returned if *return_dist=True*

`property graph`
Returns graph of underlying parcellation

`label_samples`(*annotation*, *tolerance=2*)
Matches all samples in *annotation* to parcels in *self.atlas*

Attempts to place each sample provided in *annotation* into a parcel in *self.atlas*. If *self.volumetric* is True, this function tries to best match samples in *annotation* to parcels in *self.atlas* by:

1. Determining if the sample falls directly within a parcel,
2. Checking to see if there are nearby parcels by slowly expanding the search space to include nearby voxels, up to a specified distance (specified via the *tolerance* parameter),
3. Assigning the sample to the closest parcel if there are multiple nearby parcels, where closest is determined by the parcel centroid.

If at any step a sample can be assigned to a parcel the matching process is terminated. If there is still no parcel for a given sample after this process the sample is provided a label of 0.

On the other hand, if *self.volumetric* is False, then samples are simply matched to the nearest coordinate in *self.atlas*. Once matched, *tolerance* is treated as a standard deviation threshold. That is, all samples are matched to the nearest vertex, and then samples whose distance to the nearest vertex are more than *tolerance* s.d. above the mean distance for all samples are assigned a label of 0.

Parameters

- **annotation** (`(S, 3) array_like`) – At a minimum, an array of XYZ coordinates must be provided. If a full annotation dataframe is provided, then information from the data frame (i.e., on hemisphere + structural assignments of tissue samples) is used to constrain matching of regions.
- **tolerance** (`float, optional`) – Threshold for assigning samples to parcels. Default: 2

Returns **labels** – Dataframe with parcel labels for each of *S* samples

Return type (S, 1) pandas.DataFrame

`property labels`
Returns unique labels in atlas

`match_closest_centroids`(*annotation*, *return_dist=False*)
Matches samples in *annotation* to closest centroids in *self.atlas*

Parameters

- `annotation` (*(S, 3) array_like*) – At a minimum, an array of XYZ coordinates must be provided. If a full annotation dataframe is provided, then information from the data frame (i.e., on hemisphere + structural assignments of tissue samples) is used to constrain matching of regions (if *self.atlas_info* is not None).
- `return_dist` (*bool, optional*) – Whether to also return distance to matched centroids

Returns

- **labels** (*(S,) np.ndarray*) – ID of parcel with closest centroid to samples in *annotation*
- **distance** (*(S,) np.ndarray*) – Distances of matched centroid to samples in *annotation*. Only returned if *return_dist=True*

`property tree`
Returns cKDTree constructed from provided atlas and coordinates

`property triangles`
Returns triangles of underlying graph (if applicable)

`property volumetric`
Return whether *self.atlas* is derived from a volumetric image

`abagen.matching.get_centroids`(data, coordinates)	Finds centroids of *data* in *coordinates* space
`abagen.matching.closest_centroid`(coords, ...)	Returns index of *centroids* closest to *coords* (Euclidean distance)

abagen.matching.get_centroids

`abagen.matching.`**`get_centroids`**(*data*, *coordinates*, *labels=None*)
Finds centroids of *data* in *coordinates* space

Parameters

- `data` (*(N,) array_like*) – Data labelling all *N* points in *coordinates*
- `coordinates` (*(N, D) array_like*) – Coordinates of *data* array
- `labels` (*array_like, optional*) – List of values containing labels of which to find centroids. Default: all possible labels in *data*

Returns **centroids** – Where keys are labels and values are centroids

Return type dict

abagen.matching.closest_centroid

`abagen.matching.`**`closest_centroid`**(*coords*, *centroids*, *return_dist=False*)
Returns index of *centroids* closest to *coords* (Euclidean distance)

Parameters

- `coord` (*(S, 3) array_like*) – Coordinates of samples
- `centroids` (*(N, 3) array_like*) – Centroids of parcels
- `return_dist` (*bool, optional*) – Whether to also return distance of closest centroid

Returns

- **closest** (*(S,) np.ndarray*) – Indices of closest centroid in *centroids* to *coords*
- **distance** (*(S,) np.ndarray*) – Distances of closest centroid in *centroids* to *coords*. Only returned if *return_dist=True*

7.7.6 `abagen.reporting` - Functions for generating reports

Functions for generating workflow methods reports

Note: all text contained within this module is released under a CC-0 license.

`abagen.Report`(atlas[, atlas_info, . . .])	Generates report of methods for `abagen.get_expression_data()`

abagen.Report

class abagen.**Report**(*atlas*, *atlas_info=None*, ***, *ibf_threshold=0.5*, *probe_selection='diff_stability'*, *donor_probes='aggregate'*, *lr_mirror=None*, *missing=None*, *tolerance=2*, *sample_norm='srs'*, *gene_norm='srs'*, *norm_matched=True*, *norm_structures=False*, *region_agg='donors'*, *agg_metric='mean'*, *corrected_mni=True*, *reannotated=True*, *donors='all'*, *return_donors=False*, *data_dir=None*, *counts=None*, *n_probes=None*, *n_genes=None*)

Generates report of methods for `abagen.get_expression_data()`

Refer to the doc-string of the workflow for an overview of paramter options

gen_report()

Generates main text of report

7.7.7 `abagen.io` - Loading AHBA data files

Functions for loading the various files associated with the AHBA microarray and RNAseq datasets.

This also contains functionality for optionally converting the downloaded CSV files to parquet format, which provides much faster I/O access / quicker load times.

`abagen.io.read_annotation`(fname[, copy])	Loads SampleAnnot.csv file found at *fname*
`abagen.io.read_microarray`(fname[, copy, parquet])	Loads MicroarrayExpression.csv file found at *fname*
`abagen.io.read_ontology`(fname[, copy])	Loads Ontology.csv file found at *fname*
`abagen.io.read_pacall`(fname[, copy, parquet])	Loads PACall.csv file found at *fname*
`abagen.io.read_probes`(fname[, copy])	Loads Probes.csv file found at *fname*
`abagen.io.read_genes`(fname[, copy])	Loads Genes.csv file found at *fname*
`abagen.io.read_tpm`(fname[, copy])	Loads RNAseqTPM.csv file found at *fname*
`abagen.io.read_counts`(fname[, copy])	Loads RNAseqCounts.csv file found at *fname*

abagen.io.read_annotation

`abagen.io.`**`read_annotation`**(*fname*, *copy=False*)

Loads SampleAnnot.csv file found at *fname*

Sample annotation files contain metadata on all the tissue samples taken from a single donor brain, including the spatial location of the samples.

This information can be used to combine samples within the same anatomical region across donors.

Parameters

- **`fname`** (*`str`*) – Path to SampleAnnot.csv file
- **`copy`** (*`bool, optional`*) – Whether to return a copy if *fname* is a pre-loaded pandas.Dataframe. Default: False

Returns annotation – Dataframe containing structural information on *S* samples. The row index is the unique sample ID (integer, beginning with 1) which can be used to match data to the information obtained with e.g., *`read_microarray()`*.

Return type (S, 13) pandas.DataFrame

Notes

If the provided annotation file is from microarray expression data (obtained by, e.g., *abagen.fetch_microarray()*), then the returned DataFrame will have the following columns: 'structure_id', 'slab_num', 'well_id', 'slab_type', 'structure_acronym', 'structure_name', 'polygon_id', 'mri_voxel_x', 'mri_voxel_y', 'mri_voxel_z', 'mni_x', 'mni_y', 'mni_z'.

If the provided annotation file is from RNAseq data (obtained by, e.g., *abagen.fetch_rnaseq()*), then the returned DataFrame will have the following columns: 'RNAseq_sample_name', 'replicate_sample', 'sample_name', 'well_id', 'microarray_run_id', 'ontology_color', 'main_structure', 'sub_structure', 'structure_id', 'structure_acronym', 'hemisphere', 'brain', 'million_clusters', 'clip_percentage', 'RIN_RNA_squality', 'rnaseq_run_id', 'A.Pct', 'C.Pct', 'G.Pct', 'T.Pct', 'N.Pct'

abagen.io.read_microarray

`abagen.io.`**`read_microarray`**(*fname*, *copy=False*, *parquet=True*)

Loads MicroarrayExpression.csv file found at *fname*

Microarray files contain raw expression data for all the tissue samples taken from a single donor across all genetic probes.

Parameters

- **`fname`** (*`str`*) – Path to MicroarrayExpression.csv file
- **`copy`** (*`bool, optional`*) – Whether to return a copy if *fname* is a pre-loaded pandas.Dataframe. Default: False
- **`parquet`** (*`bool, optional`*) – Whether to load data from parquet file instead of CSV. If a parquet file does not already exist then one will be created for faster loading in the future. Only available if `fastparquet` and `python-snappy` module are installed. Default: True

Returns microarray – Dataframe containing microarray expression data, where *P* is probes and *S* is samples. The row index is the unique probe ID assigned during processing, which can be used to match data to the information obtained with *`read_probes()`*. The column index is the unique sample ID (integer, beginning at 0) which can be used to match data to the information obtained with *`read_annotation()`*.

Return type (P, S) pandas.DataFrame

abagen.io.read_ontology

`abagen.io.`**`read_ontology`**(*fname*, *copy=False*)

Loads Ontology.csv file found at *fname*

Ontology files contain information on the anatomical delineations used by the Allen Institute when obtaining samples from donor brains, and are used in their online Brain Viewer to colorize regions. These files should be the same for every donors.

This information can be used to ensure that microarray samples are appropriately matched to anatomical regions.

Parameters

- **fname** (*str*) – Path to Ontology.csv file
- **copy** (*bool, optional*) – Whether to return a copy if *fname* is a pre-loaded pandas.Dataframe. Default: False

Returns **ontology** – Dataframe containing ontology information for *R* anatomical regions used by the Allen Institute. Columns include: 'id', 'acronym', 'name', 'parent_structure_id', 'hemisphere', 'graph_order', 'structure_id_path', and 'color_hex_triplet'.

Return type (R, 8) pandas.DataFrame

abagen.io.read_pacall

`abagen.io.`**`read_pacall`**(*fname*, *copy=False*, *parquet=True*)

Loads PACall.csv file found at *fname*

PA files contain a present/absent flag indicating whether the corresponding probe's expression is above background noise. It is set to 1 when both of the following conditions are met:

1. The mean signal of the probe's expression is significantly different from the corresponding background, as assessed by a 2-sided t-test where p < 0.01, and
2. The difference between the background subtracted signal and the background is significant (> 2.6 * background standard deviation).

This information can be used to discard "noisy" probes that might not be contributing high-quality expression information.

Parameters

- **fname** (*str*) – Path to PACall.csv file
- **copy** (*bool, optional*) – Whether to return a copy if *fname* is a pre-loaded pandas.Dataframe. Default: False
- **parquet** (*bool, optional*) – Whether to load data from parquet file instead of CSV. If a parquet file does not already exist then one will be created for faster loading in the future. Only available if `fastparquet` and `python-snappy` module are installed. Default: True

Returns **pacall** – Dataframe containing a binary indicator determining whether expression information for each probe exceeded background noise in a given sample, where *P* is probes and *S* is samples. The row index is the unique probe ID assigned during processing, which can be used to match data to the information obtained with `read_probes()`. The column index is the unique sample ID (integer, beginning at 1) which can be used to match data to the information obtained with `read_annotation()`.

Return type (P, S) pandas.DataFrame

abagen.io.read_probes

`abagen.io.`**`read_probes`**`(`*fname*, *copy=False*`)`

Loads Probes.csv file found at *fname*

Probe files contain metadata on all genetic probes used in the AHBA data. These files should be the same for every donor.

This information can be used to e.g., query expression data for certain genes, collapse data across probes from the same gene, etc.

Parameters

- **fname** (`str`) – Path to Probes.csv file
- **copy** (`bool, optional`) – Whether to return a copy if *fname* is a pre-loaded pandas.Dataframe. Default: False

Returns probes – Dataframe containing information for *P* genetic probes. The row index is the unique probe ID assigned during processing, which can be used to match metadata to information obtained with `read_microarray()` and `read_pacall()`. Columns include 'probe_name', 'gene_id', 'gene_symbol', 'gene_name', 'entrez_id', and 'chromosome'.

Return type (P, 6) pandas.DataFrame

abagen.io.read_genes

`abagen.io.`**`read_genes`**`(`*fname*, *copy=False*`)`

Loads Genes.csv file found at *fname*

Genes files contain metadata on all genes used in the RNAseq AHBA data. These files should be the same for every donor.

Parameters

- **fname** (`str`) – Path to Genes.csv file
- **copy** (`bool, optional`) – Whether to return a copy if *fname* is a pre-loaded pandas.Dataframe. Default: False

Returns genes – Dataframe containing information for *G* unique genes. The row index is the unique gene symbol which can be used to match metadata to information obtained with `read_tpm()` and `read_counts()`. Columns include 'gene_id', 'entrez_id', 'chromosome', 'strand', 'number_of_transcripts', 'median_transcriptome_length', 'median_genome_length', 'median_number_of_exons', 'median_gene_start', and 'median_gene_end'

Return type (G, 11) pandas.DataFrame

abagen.io.read_tpm

abagen.io.**read_tpm**(*fname*, *copy=False*)

Loads RNAseqTPM.csv file found at *fname*

RNAseq TPM files contain TPM values for all the tissue samples taken from a single donor across all genes. TPM values are scaled fragment (read) counts derived using RSEM.

Parameters

- **fname** (*str*) – Path to RNAseqTPM.csv file
- **copy** (*bool, optional*) – Whether to return a copy if *fname* is a pre-loaded pandas.Dataframe. Default: False

Returns **tpm** – Dataframe containing RNAseq TPM expression data, where *G* is genes and *S* is samples. The row index is the unique gene symbol assigned during processing, which can be used to match data to the information obtained with *read_genes()*. The column index is the unique sample ID (integer, beginning at 0) which can be used to match data to the information obtained with *read_annotation()*.

Return type (G, S) pandas.DataFrame

abagen.io.read_counts

abagen.io.**read_counts**(*fname*, *copy=False*)

Loads RNAseqCounts.csv file found at *fname*

RNAseq count files contain fragment counts for all the tissue samples taken from a single donor across all genes. Fragment counts can be fractional, as ambiguous reads are distributed between relevant transcripts. For present / absent calling, a value of zero indicates no transcript was seen.

Parameters

- **fname** (*str*) – Path to RNAseqCounts.csv file
- **copy** (*bool, optional*) – Whether to return a copy if *fname* is a pre-loaded pandas.Dataframe. Default: False

Returns **tpm** – Dataframe containing RNAseq count expression data, where *G* is genes and *S* is samples. The row index is the unique gene symbol assigned during processing, which can be used to match data to the information obtained with *read_genes()*. The column index is the unique sample ID (integer, beginning at 0) which can be used to match data to the information obtained with *read_annotation()*.

Return type (G, S) pandas.DataFrame

7.7.8 abagen.mouse - Working with the Allen Mouse Brain Atlas

abagen.mouse.available_gene_info()	Lists available attributes for *abagen.mouse.get_gene_info()*
abagen.mouse.available_structure_info()	Lists available attributes for *abagen.mouse.get_structure_info()*
abagen.mouse.available_unionization_info()	Lists attributes for *abagen.mouse.get_unionization_from_gene()*
abagen.mouse.get_gene_info([id, acronym, …])	Queries Allen API for information about given gene

continues on next page

Table 9 – continued from previous page

abagen.mouse.get_structure_info([id, . . .])	Queries Allen API for information about given gene
abagen.mouse.get_structure_coordinates([id, . . .])	Finds xyz coordinates of provided structure(s) in *reference_space*
abagen.mouse.get_unionization_from_gene([. . .])	Gets unionization data for provided gene(s)
abagen.mouse.fetch_allenref_genes([. . .])	Loads all genes from Allen Reference database
abagen.mouse.fetch_allenref_structures([. . .])	Loads all anatomical structures in the Allen Reference Atlas
abagen.mouse.fetch_rubinov2015_structures([. . .])	Loads subset of anatomical structures in Allen Reference Atlas from [MI1]

abagen.mouse.available_gene_info

abagen.mouse.**available_gene_info**()

Lists available attributes for *abagen.mouse.get_gene_info()*

abagen.mouse.available_structure_info

abagen.mouse.**available_structure_info**()

Lists available attributes for *abagen.mouse.get_structure_info()*

abagen.mouse.available_unionization_info

abagen.mouse.**available_unionization_info**()

Lists attributes for *abagen.mouse.get_unionization_from_gene()*

abagen.mouse.get_gene_info

abagen.mouse.**get_gene_info**(*id=None*, *acronym=None*, *name=None*, *attributes=None*, *verbose=False*)

Queries Allen API for information about given gene

One of *id*, *acronym*, or *name* must be provided.

Parameters

- **id** (*int, optional*) – Numerical gene ID
- **acronym** (*str, optional*) – Short-form gene acronym (case sensitive)
- **name** (*str, optional*) – Full gene name (case sensitive)
- **attributes** (*str or list, optional*) – Which attributes / information to obtain for the provided gene. See *abagen.mouse.available_gene_info()* for list of available attributes to request. If not specified all available attributes will be returned. Default: None
- **verbose** (*bool, optional*) – Whether to print status messages. Default: False

Returns **info** – If *attributes* is a str, returns an int or str depending on specified attribute. If *attributes* is a list, return a dict where keys are attributes and values are str or int.

Return type pandas.DataFrame

Raises `ValueError` – The provided gene is invalid

Examples

Get gene ID and name corresponding to gene acronym 'Pdyn':

```
>>> from abagen import mouse
>>> mouse.get_gene_info(acronym='Pdyn',
...                     attributes=['id', 'name'])
            id          name
acronym
Pdyn     18376  prodynorphin
```

You can also supply multiple genes to the query:

```
>>> mouse.get_gene_info(acronym=['Ace', 'Cd99'],
...                     attributes=['id', 'name'])
             id                                              name
acronym
Ace       11210  angiotensin I converting enzyme (peptidyl-dipe...
Cd99     163028                                      CD99 antigen
```

abagen.mouse.get_structure_info

abagen.mouse.**get_structure_info**(*id=None*, *acronym=None*, *name=None*, *attributes=None*, *verbose=False*)

Queries Allen API for information about given gene

One of *structure_id*, *structure_acronym*, or *structure_name* must be provided.

Parameters

- **id** (*int, optional*) – Numerical structure ID
- **acronym** (*str, optional*) – Short-form structure acronym (case sensitive)
- **name** (*str, optional*) – Full structure name (case sensitive)
- **attributes** (*str or list, optional*) – Which attributes / information to obtain for the provided structure. See *abagen.mouse.available_structure_info()* for list of available attributes to request. If not specified all available attributes will be returned. Default: None
- **verbose** (*bool, optional*) – Whether to print status messages. Default: False

Returns **info** – Where columns are the requested attributes and index is the provided structural identifier type (e.g., 'id', 'acronym', 'name')

Return type pandas.DataFrame

Raises `ValueError` – The provided structure is invalid

Examples

Get the full names of structures 22 and 1018:

```
>>> from abagen import mouse
>>> mouse.get_structure_info(id=[22, 1018],
...                          attributes=['acronym', 'name'])
     acronym                                  name
id
22      PTLp  Posterior parietal association areas
1018    AUDv                  Ventral auditory area
```

abagen.mouse.get_structure_coordinates

`abagen.mouse.`**`get_structure_coordinates`**(*id=None*, *acronym=None*, *name=None*, *reference_space='sagittal'*, *verbose=False*)

Finds xyz coordinates of provided structure(s) in *reference_space*

Parameters

- **id** (*int, optional*) – Numerical structure ID
- **acronym** (*str, optional*) – Short-form structure acronym (case sensitive)
- **name** (*str, optional*) – Full structure name (case sensitive)
- **reference_space** (*{'sagittal', 'coronal'}, optional*) – Reference space from which to extract coordinates. Default: 'sagittal'
- **verbose** (*bool, optional*) – Whether to print status messages. Default: False

Returns **coords** – With columns ['structure_id', 'x', 'y', 'z']

Return type pandas.DataFrame

Examples

Get the coordinates of structure 1018:

```
>>> from abagen import mouse
>>> mouse.get_structure_coordinates(id=1018)
   structure_id     x     y     z
0          1018  7800  3400  1050
```

abagen.mouse.get_unionization_from_gene

`abagen.mouse.`**`get_unionization_from_gene`**(*id=None*, *acronym=None*, *name=None*, *slicing_direction='sagittal'*, *structures=None*, *attributes=None*, *average=True*, *verbose=False*)

Gets unionization data for provided gene(s)

One of *id*, *acronym*, or *name* must be provided.

Parameters

- **id** (*int, optional*) – Numerical gene ID

- **acronym** (*str, optional*) – Short-form gene acronym (case sensitive)
- **name** (*str, optional*) – Full gene name (case sensitive)
- **slicing_direction** (*{'sagittal', 'coronal'}, optional*) – Slicing direction of brain tissue
- **structures** (*list, optional*) – List of structures (id, acronym, or name) for which to get unionization information associated with provided *experiment_id*. If not specified uses structures documented in [MI1]. Specifying either the id or name is recommended as acronyms are not unique to structures. Default: None
- **attributes** (*str or list, optional*) – Which attributes / information to obtain for the provided gene. See `abagen.mouse.available_gene_info()` for list of available attributes to request. If not specified then only 'expression_density' will be returned. Specifying 'all' will return all information. Default: None
- **average** (*bool, optional*) – Whether to average across experiments if there are multiple experiments corresponding to any provided gene(s). Only experiments probing the same gene will be considered for averaging, and distinct structures will be retained. Default: True
- **verbose** (*bool, optional*) – Whether to print status messages. Default: False

Returns **unionization** – Where columns are unionization attributes and the index corresponds to strucuture and gene ids (if *experiments* is provided as a list with multiple genes). If *average=False*, *experiments* will also be a level in index

Return type pandas.DataFrame

Examples

```
>>> from abagen import mouse
>>> mouse.get_unionization_from_gene(acronym='Pdyn',
...                                  structures=[22, 31])
                      expression_density
gene_id structure_id
18376   22                      0.024840
        31                      0.017199
>>> mouse.get_unionization_from_gene(acronym=['Ace', 'Cd99'],
...                                  structures=[22, 31])
                      expression_density
gene_id structure_id
11210   22                      0.001283
        31                      0.001427
163028  22                      0.067537
        31                      0.056442
```

abagen.mouse.fetch_allenref_genes

`abagen.mouse.fetch_allenref_genes`(*entry_type=None*, *cache=True*, *data_dir=None*, *verbose=True*)

Loads all genes from Allen Reference database

Parameters

- **entry_type** (*{'id', 'acronym', 'name'}, optional*) – The type of gene identifier to load. Specifying 'id' returns a list of numerical gene IDs, 'acronym' returns a list of short-form gene acronyms, and 'name' returns full gene names. If not specified, returns a dataframe with all information. Default: None
- **cache** (*bool, optional*) – Whether to use cached gene information (if it exists). Setting to False will overwrite cache. Default: True
- **data_dir** (*str, optional*) – Directory where data should be downloaded and unpacked. Default: $HOME/ abagen-data
- **verbose** (*bool, optional*) – Whether to print status message. Default: True

Returns **genes** – Genes in Allen Reference database

Return type list or `pandas.DataFrame`

Notes

May require internet access to make query to the Allen API (which will take some time); after query is made once the results are cached.

abagen.mouse.fetch_allenref_structures

`abagen.mouse.fetch_allenref_structures`(*entry_type=None*, *cache=True*, *data_dir=None*, *verbose=True*)

Loads all anatomical structures in the Allen Reference Atlas

Parameters

- **entry_type** (*{'id', 'acronym', 'name'}, optional*) – The type of structural identifier to load. Specifying 'id' returns a list of numerical structure IDs, 'acronym' returns a list of short-form structure acronyms, and 'name' returns full structure names. If not specified, returns a dataframe with all information. Default: None
- **cache** (*bool, optional*) – Whether to use cached structure information (if it exists). Setting to False will overwrite cache. Default: True
- **data_dir** (*str, optional*) – Directory where data should be downloaded and unpacked. Default: $HOME/ abagen-data
- **verbose** (*bool, optional*) – Whether to print status message. Default: True

Returns **structures** – Anatomical structures in Allen Reference Atlas

Return type list or `pandas.DataFrame`

Notes

May require internet access to make query to the Allen API (which will take some time); after query is made once the results are cached.

abagen.mouse.fetch_rubinov2015_structures

`abagen.mouse.fetch_rubinov2015_structures`(*entry_type=None*)

Loads subset of anatomical structures in Allen Reference Atlas from [MI1]

Parameters **entry_type** (`{'id', 'acronym', 'name'}, optional`) – The type of structural identifier to load. Specifying 'id' returns a list of numerical structure IDs, 'acronym' returns a list of short-form structure acronyms, and 'name' returns full structure names. If not specified, returns a dataframe with all information. Default: None

Returns **structures** – Anatomical structures in Allen Reference Atlas from [MI1]

Return type list or `pandas.DataFrame`

References

BIBLIOGRAPHY

[A1] Arnatkeviciūtė, A., Fulcher, B. D., & Fornito, A. (2019). A practical guide to linking brain-wide gene expression and neuroimaging data. NeuroImage, 189, 353-367.

[A2] Hawrylycz, M.J. et al. (2012) An anatomically comprehensive atlas of the adult human transcriptome. Nature, 489, 391-399.

[A3] Fulcher, B. D., & Fornito, A. (2016). A transcriptional signature of hub connectivity in the mouse connectome. Proceedings of the National Academy of Sciences, 113(5), 1435-1440.

[DS1] Burt, J. B., Demirtaş, M., Eckner, W. J., Navejar, N. M., Ji, J. L., Martin, W. J., … & Murray, J. D. (2018). Hierarchy of transcriptomic specialization across human cortex captured by structural neuroimaging topography. Nature neuroscience, 21(9), 1251.

[PC1] Fulcher, B. D., & Fornito, A. (2016). A transcriptional signature of hub connectivity in the mouse connectome. Proceedings of the National Academy of Sciences, 113(5), 1435-1440.

[PC2] Fulcher, B. D., Little, M. A., & Jones, N. S. (2013). Highly comparative time-series analysis: the empirical structure of time series and their methods. Journal of the Royal Society Interface, 10(83), 20130048

[MI1] Rubinov, M., Ypma, R. J., Watson, C., & Bullmore, E. T. (2015). Wiring cost and topological participation of the mouse brain connectome. Proceedings of the National Academy of Sciences, 112(32), 10032-10037.

PYTHON MODULE INDEX

INDEX