Read the Docs Template Documentation

Read the Docs

May 14, 2018

About ConWhAt

ConWhAt is a tool for studying the effects of white matter damage on brain networks.

The code is hosted on github: https://github.com/JohnGriffiths/ConWhAt/

It is written in python and draws strongly on the powerful functionality provided by other neuroimaging analysis tools in the nipy ecosystem for manipulation and visualization of nifti images and tractography streamlines.

The primary intended mode of interaction is through jupyter notebooks and python / bash scripting. Notebooks versions of some of the following documentation pages can be viewed at https://github.com/JohnGriffiths/ConWhAt/doc/examples

(Please note: the software and atlases are still in beta, and future versions may include major front and/or backend changes)

CHAPTER 1

Overview

Classical and modern schemas for defining and characterizing neuroanatomical structures in white matter tissue can be categorized along two main dimensions *Ontology* and *Representation*. Each of these has two main flavours: tract-based/connectivity-based, and image-based/streamline-based. This perspective is a key part of the rationale for, and design of, the ConWhAt software and atlases. Read more about these concepts here.

For info about the design and construction of the ConWhAt volumetric and streamlinetric atlases, see here.

CHAPTER 2

Ontology & Representation

Classical and modern schemas for defining and characterizing neuroanatomical structures in white matter tissue can be categorized along two main dimensions *Ontology* and *Representation*. Each of these has two main flavours: tract-based/connectivity-based, and image-based/streamline-based.

2.1 Ontology

Conventional approaches to atlasing white matter structures follow a *tract-based* ontology: they assign locations in stereotaxic space to a relatively small number of gross white matter tracts from the classical neuroanatomy literature.

This has been an extremely successful program of research, particularly in relation to post-mortem dissections and MR image or tractography streamline segmentation methodologies, as it draws on some of the brain's most salient and consistent macroscopic structural features.

Unfortunately, however, tract-based ontologies aren't particularly well-suited to network-based descriptions of brain organization. The reason for this is that identifying that a given spatial location falls within one or other canonical white matter tract (e.g. the inferior longitudinal fasciculus) doesn't in itself say very much about the specific grey matter connectivity of that location. Although they consist of many hundreds of thousands of structural connections (axons), the white matter tracts per se are not descriptions of connectivity, but rather of large 3D geometric structures that can be located relative to certain anatomical landmarks.

The second flavour of white matter ontology, which is becoming increasingly prominent in modern modern neuroscientific research, is a *connectivity-based* one. The idea here is that rather than following the classical anatomical tract nomenclature, to label white matter voxels according to the grey matter regions that their constituent fibers interconnect. Combining this with the modern macro-connectomics tractography approach (whole-brain tractography, segmented using region pairs from a given grey matter parcellation), gives the *connectome-based white matter atlas* methodology, which is what ConWhAt (and other earlier tools, notably NeMo) is designed to support.

The benefit of this approach is that a scientist/clinician/citizen can take a set of (standard space) coordinates, or a nifti-format ROI mask such as a binary lesion map, and straightforwardly query which grey matter region pairs (i.e. connectome-edges) have fibers passing through those locations.

That information can then be used together with the used parcellation's *canonical connectome* (normative group-averaged connectivity matrix), to obtain a lesion-modified structural (macro) connectome. This can be done very

quickly with zero tractography data or analysis required, and as little as a list of numbers (voxel coordinates) as input.

An important point to emphasize is that the tract-based and connectivity-based ontologies are not diametrically opposed; in fact they should be regarded as highly complementary. This is why we have also included support in ConWhAt for standard tract-based atlas analyses.

2.2 Representation

Traditionally, anatomical atlases have (as the name suggests) existed as collections of more-or-less schematic two- or three-dimensional depictions, printed on the pages of a (generally quite large) book. Whilst this mode of representation is by no means uncommon, atlases in modern neuroimaging are generally understood to be digital data structures, which bear varying degrees of resemblance to their paper-based forebears.

In particular, representations of white matter anatomical data in neuroimaging come in two flavours: *image-based* (which we refer to as *volumetric*), and (tractography) *streamline-based* (which we refer to neologistically as *streamlinetric*). These two forms of representation are very different beasts, each with its own set of distinctive features and pros/cons (which is why we make a major effort to support both in ConWhAt)

For example: the basic units of volumetric representations are scalar-valued (voxel intensities), which when taken as a set allow for complex and rich encoding of 3D shapes in virtue of their arrangement on a regular 3D grid. In contrast, the basic units of streamlinetric representations are vector-valued; namely lists of coordinates in 3D space. Each individual streamline (unlike each individual voxel) therefore provides some holistic 3D shape information. The closest equivalent of voxel intensities for streamlines would be the presence of multiple overlapping streamlines; although this is much less compressed than scalar intensity values.

The definition and interpretation of 'damage' also turns out to be somewhat different for volumetric vs. streamlinetric representations. In the volumetric case, damage (defined as e.g. proportional overlap with a lesion) is evaluated independently for every voxel. In the streamlinetric case, damage is instead evaluated independently for every streamline, with the important corollary that evaluations at different spatial locations are not independent of each other. In short, if an upstream part of a streamline is considered to be damaged, then downstream parts are also considered to be damaged, even if they themselves are nowhere near the damaged area. Which is, of course, how one would expect real damage to axons to operate. Streamlinetric quantifications of damage are somewhat more difficult to work with than their volumetric equivalents, however.

There has been relatively little work done on direct comparisons of volumetric and streamlinetric characterizations of lesions, or indeed of white matter in general. ConWhAt is to our knowledge the first and only atlas-based tool that allows direct comparison between the two.

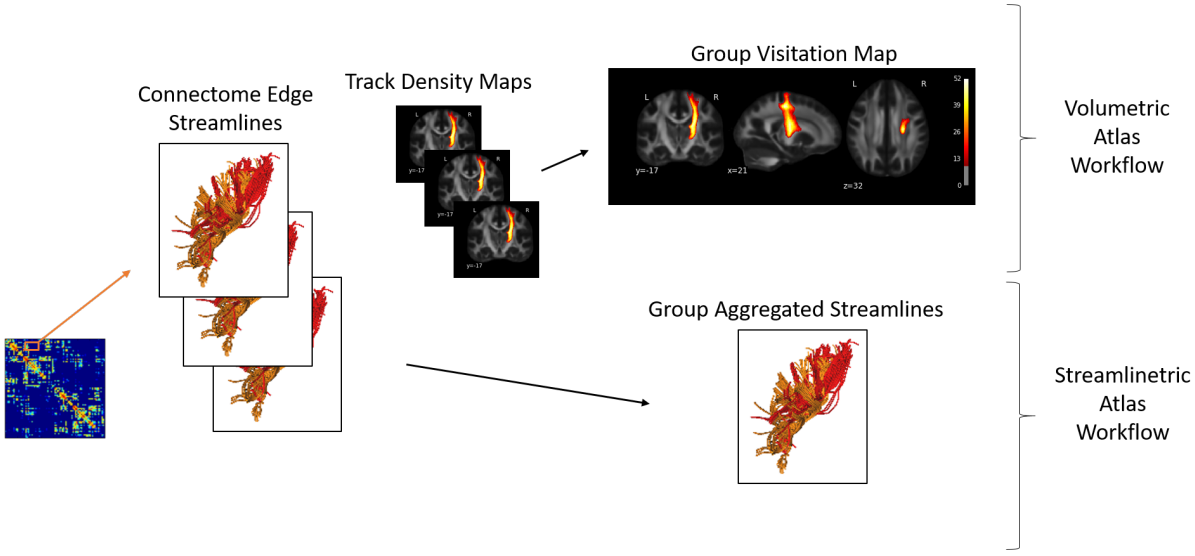
CHAPTER 3

ConWhAt Atlases

A central component of ConWhAt is the large set of connectome-based white matter atlases we have developed for use with the software. The atlas construction methodology is described in detail in Griffiths & McIntosh (in prep). Here we give a brief summary:

All of the ConWhAt atlases were constructed from on dipy deterministic whole-brain HARDI tractography reconstructions using the HCP WU-Minn corpus. Whole-brain streamline sets were segmented with region pairs using a broad set of brain parcellations, yielding anatomical connectivity matrices and connectome edge-labelled streamline sets. The streamlines are then entered into both volumetric and a streamlinetric atlas construction pipelines:

- Volumetric workfow: convert streamlines to track density images (visitation maps), spatially normalize, average
- Streamlinetric workflow: spatially normalize streamlines, concatenate, cluster



CHAPTER 4

Installation

4.1 Using latest github master source

Clone latest version from github

```
$ git clone https://github.com/JohnGriffiths/ConWhAt
```

Now go to the cloned folder and install manually

```
$ cd ConWhAt
$ python setup.py install
```

Alternatively, simply add the cloned path to your `pythonpath`.

4.2 Using pypi

(coming soon)

4.3 Using with docker

(coming soon)

- Install docker-engine (instructions here)
- Build the docker container

```
$ docker build -it ConWhAt <path to ConWhAt folder>
```

- Start Jupyter notebook server in the container

```
$ docker run -it -p 8888:8888 ConWhAt
```

CHAPTER 5

Downloading atlases

In order to use ConWhAt you will need to download the atlases you want to use in your analyses.

Be aware: some of these files are quite large!

The atlas files are hosted on the ConWhAt NITRC page.

There are currently three options for downloading the atlas data:

1. Navigate to the NITRC page using the above link and download atlases manually
2. Use the links in the table below
3. Use the provided data fetcher utilities (see below)

5.1 List of ConWhAt atlases

Here is a list of the currently and soon-to-be available ConWhAt atlases:

Name	Description	Available
CWL2k8Sc33Vol3dS100	Lausanne 2008 scale 33 parcellation volumetric, 100subjects	*Now* (link)
CWL2k8Sc60Vol3dS100	Lausanne 2008 scale 60 parcellation volumetric, 100subjects	*Now* (link)
CWL2k8Sc125Vol3dS100	Lausanne 2008 scale 125 parcellation volumetric, 100subjects	*Soon*
CWL2k8Sc250Vol3dS100	Lausanne 2008 scale 250 parcellation volumetric, 100subjects	*Soon*
CWL2k8Sc500Vol3dS100	Lausanne 2008 scale 500 parcellation volumetric, 100subjects	*Soon*
CWL2k8Sc33StreamS100	Lausanne 2008 scale 33 parcellation streamlinetric, 100subjects	*Soon*
CWL2k8Sc60StreamS100	Lausanne 2008 scale 60 parcellation streamlinetric, 100subjects	*Soon*
CWL2k8Sc125StreamS100	Lausanne 2008 scale 125 parcellation streamlinetric, 100subjects	*Soon*
CWL2k8Sc250StreamS100	Lausanne 2008 scale 250 parcellation streamlinetric, 100subjects	*Soon*
CWL2k8Sc500StreamS100	Lausanne 2008 scale 500,parcellation streamlinetric, 100subjects	*Soon*

5.2 Downloading using the data fetcher functions

Atlases can be downloaded directly from the NITRC repository using the *fetcher* utilities:

```
from conwhat.utils.fetchers import fetch_conwhat_atlas

res = fetchers.fetch_conwhat_atlas(atlas_name='CWL2k8Sc33Vol3d100s_v01',
                                   dataset_dir='/tmp',remove_existing=True)
```

5.3 Note on streamlinetric atlases

Important note: the streamlinetric atlases all make use of a common streamlines file, which is quite large and only needs to be downloaded once. Currently that file lives in the L2k8 scale 33 atlas folder. You need to create symlinks to that file in all other streamlinetric atlas folders that use that streamlines file. Downloading using the data fetcher (recommended) will set up these hyperlinks for you.

CHAPTER 6

Exploring the atlases

There are four different atlas types in ConWhat, corresponding to the 2 ontology types (Tract-based / Connectivity-Based) and 2 representation types (Volumetric / Streamlinetric).

(More on this schema here)

```
>>> # ConWhAt stuff
>>> from conwhat import VolConnAtlas,StreamConnAtlas,VolTractAtlas,StreamTractAtlas
>>> from conwhat.viz.volume import plot_vol_scatter

>>> # Neuroimaging stuff
>>> import nibabel as nib
>>> from nilearn.plotting import plot_stat_map,plot_surf_roi

>>> # Viz stuff
>>> %matplotlib inline
>>> from matplotlib import pyplot as plt
>>> import seaborn as sns

>>> # Generic stuff
>>> import glob, numpy as np, pandas as pd, networkx as nx
```

We'll start with the scale 33 lausanne 2008 volumetric connectivity-based atlas.

Define the atlas name and top-level directory location

```
>>> atlas_dir = '/scratch/hpc3230/Data/conwhat_atlases'
>>> atlas_name = 'CWL2k8Sc33Vol3d100s_v01'
```

Initialize the atlas class

```
>>> vca = VolConnAtlas(atlas_dir=atlas_dir + '/' + atlas_name,
                       atlas_name=atlas_name)

loading file mapping
```

(continues on next page)

(continued from previous page)

```
loading vol bbox
loading connectivity
```

This atlas object contains various pieces of general information

```
>>> vca.atlas_name

'CWL2k8Sc33Vol3d100s_v01'
```

```
>>> vca.atlas_dir

'/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01'
```

Information about each atlas entry is contained in the `vfms` attribute, which returns a pandas dataframe

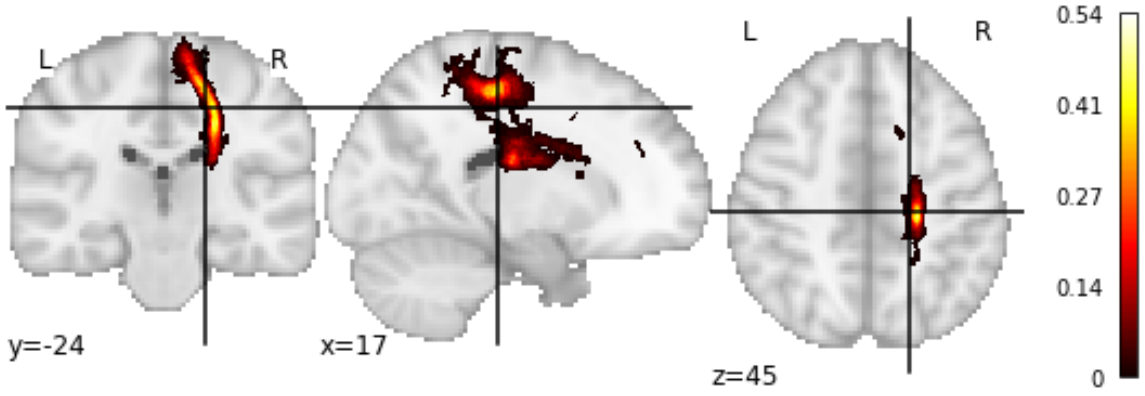
Additionally, connectivity-based atlases also contain a `networkx` graph object `vca.Gnx`, which contains information about each connectome edge

```
>>> vca.Gnx.edges[(10,35)]

{'attr_dict': {'4dvolind': nan,
  'fullname': 'L_paracentral_to_L_caudate',
  'idx': 1637,
  'name': '10_to_35',
  'nii_file': 'vismap_grp_11-36_norm.nii.gz',
  'nii_file_id': 1637,
  'weight': 50.240000000000002,
  'xmax': 92,
  'xmin': 61,
  'ymax': 167,
  'ymin': 75,
  'zmax': 92,
  'zmin': 62}}
```

Individual atlas entry nifti images can be grabbed like so

```
>>> img = vca.get_vol_from_vfm(1637)

getting atlas entry 1637: image file /scratch/hpc3230/Data/conwhat_atlases/
↪CWL2k8Sc33Vol3d100s_v01/vismap_grp_11-36_norm.nii.gz
```

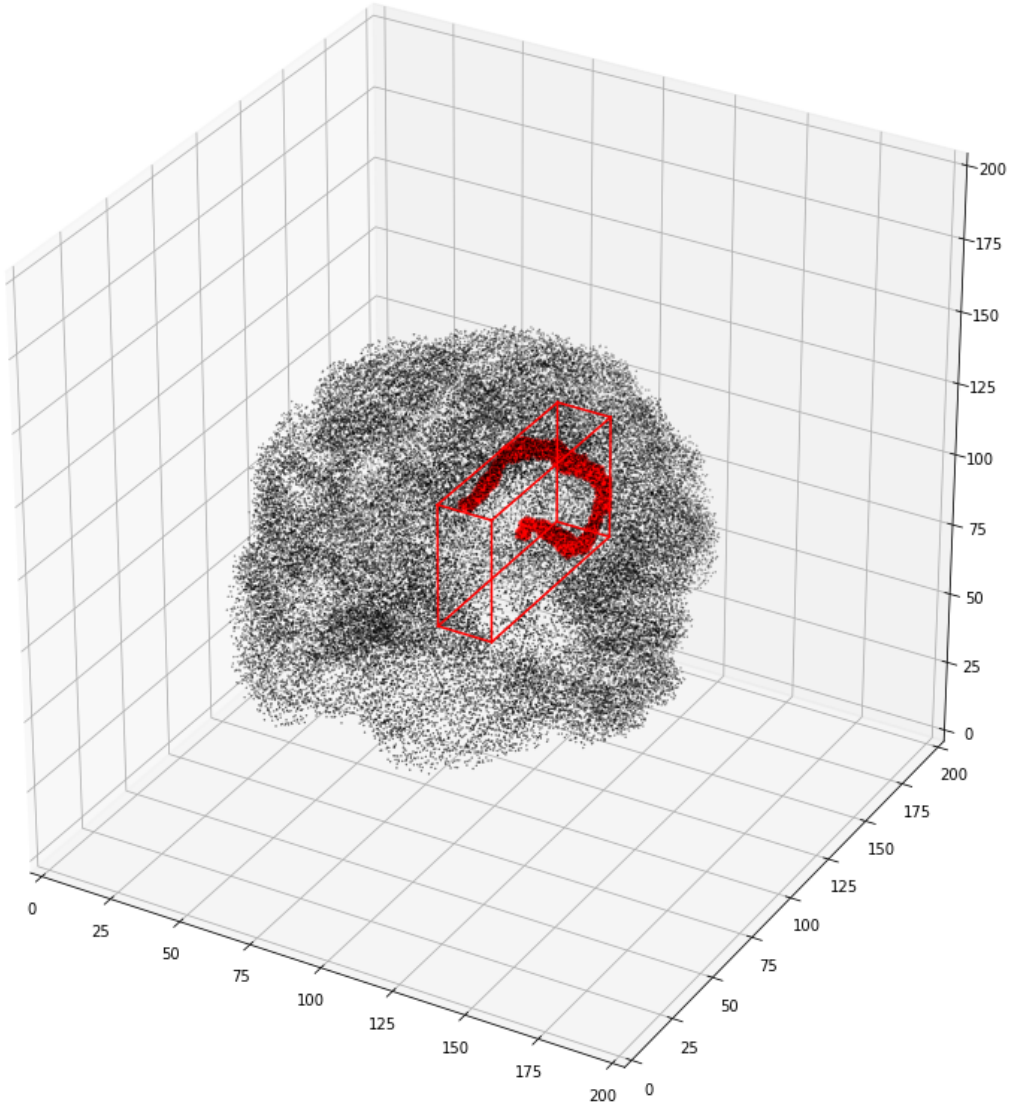
```
>>> plot_stat_map(img)
```



Or alternatively as a 3D scatter plot, along with the x,y,z bounding box

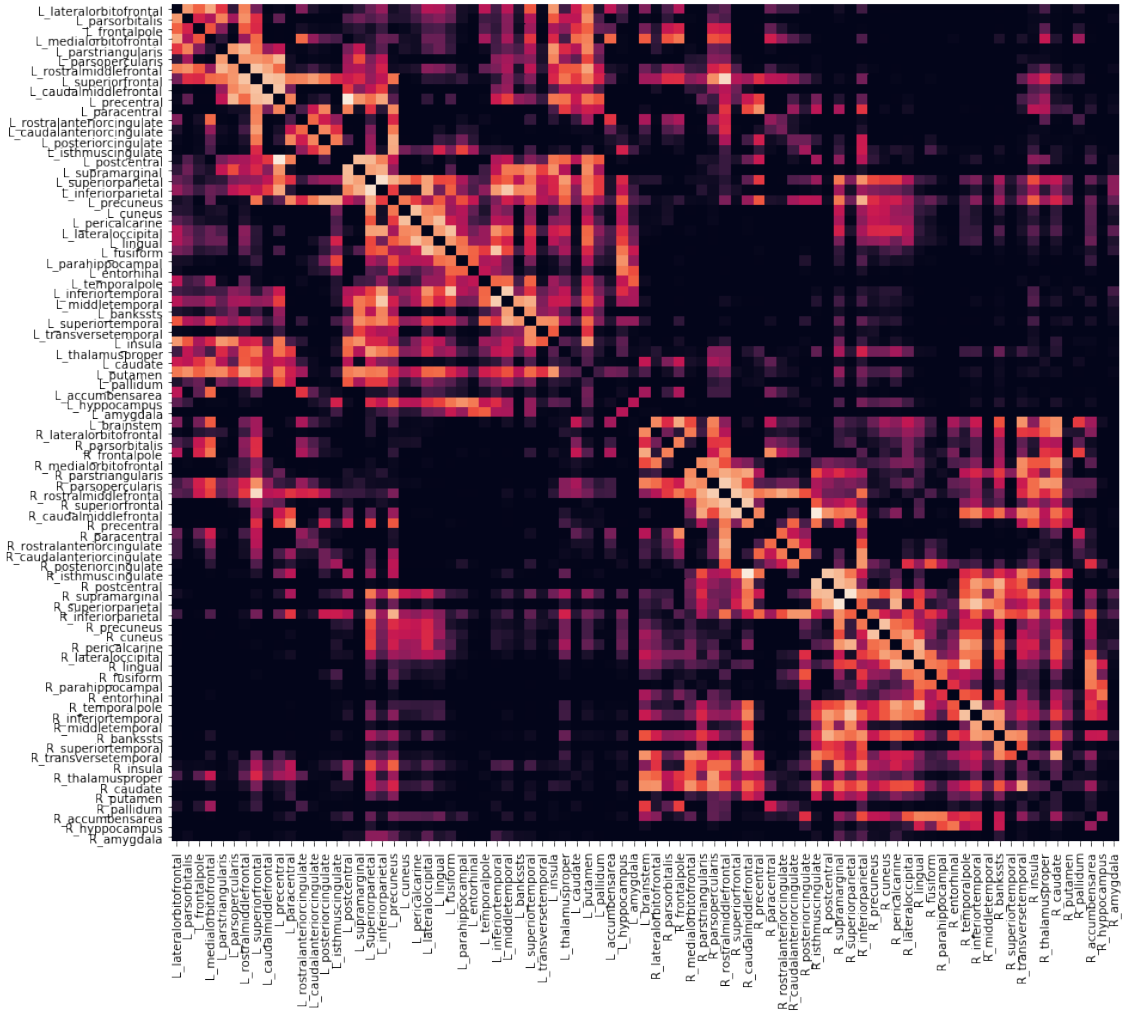
```
>>> vca.bbox.ix[1637]

xmin      61
xmax      92
ymin      75
ymax     167
zmin      62
zmax      92
Name: 1637, dtype: int64
```

```
>>> ax = plot_vol_scatter(vca.get_vol_from_vfm(1),c='r',bg_img='nilearn_destrieux',
>>>                       bg_params={'s': 0.1, 'c':'k'},figsize=(20, 15))
>>> ax.set_xlim([0,200]); ax.set_ylim([0,200]); ax.set_zlim([0,200]);

getting atlas entry 1: image file /scratch/hpc3230/Data/conwhat_atlases/
↪CWL2k8Sc33Vol3d100s_v01/vismap_grp_39-56_norm.nii.gz
```



We can also view the weights matrix like so:

```
>>> fig, ax = plt.subplots(figsize=(16,12))
>>> sns.heatmap(np.log1p(vca.weights),xticklabels=vca.region_labels,
>>>                yticklabels=vca.region_labels,ax=ax);
>>> plt.tight_layout()
```



The `vca` object also contains x,y,z bounding boxes for each structure

We also stored additional useful information about the ROIs in the associated parcellation, including cortical/subcortical labels

```
>>> vca.cortex

array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,
        0.,  0.,  0.,  0.])
```

. . . hemisphere labels

```
>>> vca.hemispheres

array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
```

(continues on next page)

(continued from previous page)

```
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.])
```

. . . and region mappings to freesurfer's fsaverage brain

```
>>> vca.region_mapping_fsav_lh

array([ 24.,  29.,  28., ...,  16.,   7.,   7.])
```

```
>>> vca.region_mapping_fsav_rh

array([ 24.,  29.,  22., ...,   9.,   9.,   9.])
```

which can be used for, e.g. plotting ROI data on a surface

```
>>> f = '/opt/freesurfer/freesurfer/subjects/fsaverage/surf/lh.inflated'
>>> vtx,tri = nib.freesurfer.read_geometry(f)
>>> plot_surf_roi([vtx,tri],vca.region_mapping_fsav_lh);
```

CHAPTER 7

Defining a Lesion

Conducting a lesion analysis in ConWhAt is extremely simple. All that is needed is a binary `.nii` format lesion mask, with ones indicating lesioned tissue, and zeros elsewhere.

> *(Note: we terms like 'lesion' and 'damage' throughout most of this documentation, as that is the most natural primary context for ConWhAt analyses. Remember however that all we are doing at the end of the day is doing a set of look-up operations between a list of standard space coordinates on the one hand (as defined by non-zero values in a ''.nii'' image), and the spatial locations of each 'connectome edge' - i.e. each entry in our anatomical connectivity matrix. One can envisave many alternative interpretations/applications of this procedure; for example to map the connectivity effects of magnetic field or current distributions from nonivasive brain stimulation). Still, for concreteness and simplicity, we stick with 'lesion', 'damage', etc. for the most part.)*

A common way to obtain a lesion map is to from a patient's T1-weighted MR image. Although this can be done manually, it is strongly recommended to use an automated lesion segmentation tools, followed by manual editing.

An alternative way is simply to define a lesion location using standard space coordinates, and build a 'lesion' mask *de-novo*. This is what we do in the following example. On the next page we do a ConWhAt connectome-based decomposition analysis on this 'synthetic' lesion mask.

```
>>> # ConWhAt stuff
>>> from conwhat import VolConnAtlas,StreamConnAtlas,VolTractAtlas,StreamTractAtlas
>>> from conwhat.viz.volume import plot_vol_and_rois_nilearn

>>> # Neuroimaging stuff
>>> import nibabel as nib
>>> from nilearn.plotting import plot_roi
>>> from nipy.labs.spatial_models.mroi import subdomain_from_balls
>>> from nipy.labs.spatial_models.discrete_domain import grid_domain_from_image

>>> # Viz stuff
>>> %matplotlib inline
>>> from matplotlib import pyplot as plt
```

(continues on next page)

(continued from previous page)

```
>>> # Generic stuff
>>> import numpy as np
```

Define some variables

```
>>> # Locate the standard space template image
>>> fsl_dir = '/global/software/fsl/5.0.10'
>>> t1_mni_file = fsl_dir + '/data/standard/MNI152_T1_1mm_brain.nii.gz'
>>> t1_mni_img = nib.load(t1_mni_file)

>>> # This is the output we will save to file and use in the next example
>>> lesion_file = 'synthetic_lesion_20mm_sphere_-46_-60_6.nii.gz'
```

Define the 'synthetic lesion' location and size using standard (MNI) space coordinates

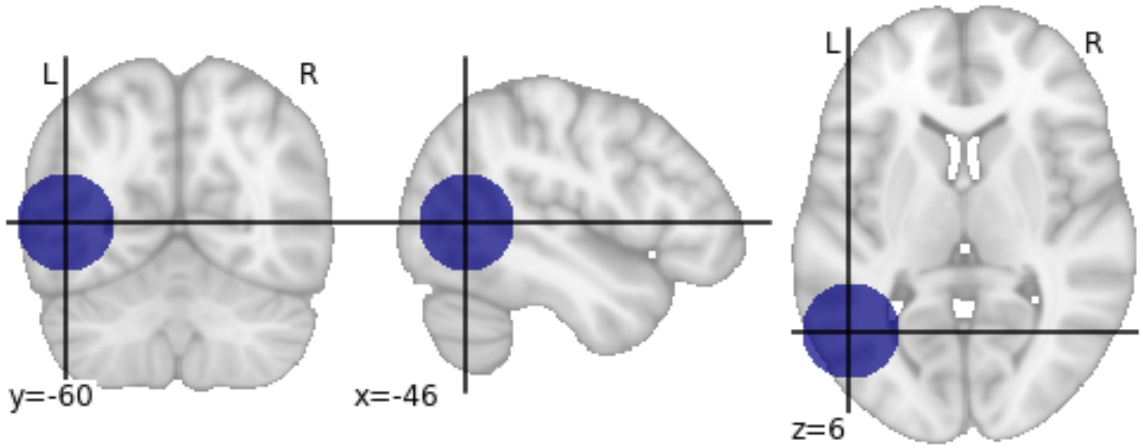
```
>>> com = [-46,-60,6] # com = centre of mass
>>> rad = 20             # radius
```

Create the ROI

```
>>> domain = grid_domain_from_image(t1_mni_img)
>>> lesion_img = subdomain_from_balls(domain,np.array([com]), np.array([rad])).to_
↪image()
```

Plot on brain slices

```
>>> plot_roi(lesion_img,bg_img=t1_mni_img,black_bg=False);
```



Save to file

```
>>> lesion_img.to_filename(lesion_file)
```

. . . now we move on to doing a lesion analysis with this file.

CHAPTER 8

Compute lesion overlap

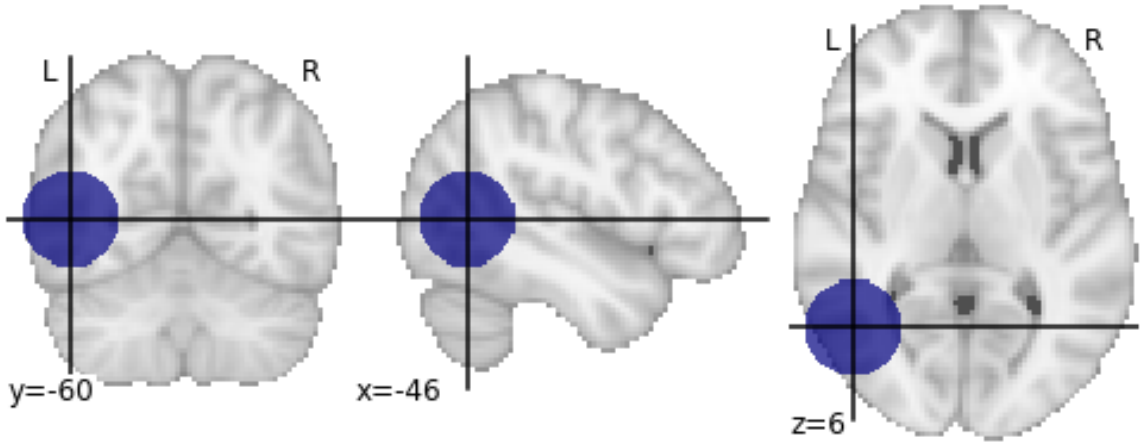
```
# ConWhAt stuff
from conwhat import VolConnAtlas,StreamConnAtlas,VolTractAtlas,StreamTractAtlas
from conwhat.viz.volume import plot_vol_scatter

# Neuroimaging stuff
import nibabel as nib
from nilearn.plotting import (plot_stat_map,plot_surf_roi,plot_roi,
                              plot_connectome,find_xyz_cut_coords)
from nilearn.image import resample_to_img

# Viz stuff
%matplotlib inline
from matplotlib import pyplot as plt
import seaborn as sns

# Generic stuff
import glob, numpy as np, pandas as pd, networkx as nx
from datetime import datetime
```

We now use the synthetic lesion constructed in the previous example in a ConWhAt lesion analysis.

```
lesion_file = 'synthetic_lesion_20mm_sphere_-46_-60_6.nii.gz' # we created this file␣
↪from scratch in the previous example
```

Take another quick look at this mask:

```
lesion_img = nib.load(lesion_file)
plot_roi(lesion_file);
```



Since our lesion mask does not (by construction) have a huge amount of spatial detail, it makes sense to use one of the lower-resolution atlas. As one might expect, computation time is considerably faster for lower-resolution atlases.

```
>>> cw_atlases_dir = '/global/scratch/hpc3230/Data/conwhat_atlases'  # change this␣
↪accordingly
>>> atlas_name = 'CWL2k8Sc33Vol3d100s_v01'
>>> atlas_dir = '%s/%s' %(cw_atlases_dir, atlas_name)
```

See the previous tutorial on 'exploring the conwhat atlases' for more info on how to examine the components of a given atlas in *ConWhAt*.

Initialize the atlas

```
>>> cw_vca = VolConnAtlas(atlas_dir=atlas_dir)
```

```
loading file mapping
loading vol bbox
loading connectivity
```

Choose which connections to evaluate.

This is normally an array of numbers indexing entries in `cw_vca.vfms`.

Pre-defining connection subsets is a useful way of speeding up large analyses, especially if one is only interested in connections between specific sets of regions.

As we are using a relatively small atlas, and our lesion is not too extensive, we can assess all connections.

```
>>> idxs = 'all' # alternatively, something like: range(1,100), indicates the first␣
↪100 cnxns (rows in .vmfs)
```

Now, compute lesion overlap statistics.

```
>>> jlc_dir = '/global/scratch/hpc3230/joblib_cache_dir' # this is the cache dir␣
↪where joblib writes temporary files
>>> lo_df,lo_nx = cw_vca.compute_hit_stats(lesion_file,idxs,n_jobs=4,joblib_cache_
↪dir=jlc_dir)
```

```
computing hit stats for roi synthetic_lesion_20mm_sphere_-46_-60_6.nii.gz
```

This takes about 20 minutes to run.

`vca.compute_hit_stats()` returns a `pandas` dataframe, `lo_df`, and a `networkx` object, `lo_nx`.

Both contain mostly the same information, which is sometimes more useful in one of these formats and sometimes in the other.

`lo_df` is a table, with rows corresponding to each connection, and columns for each of a wide set of statistical metrics for evaluating sensitivity and specificity of binary hit/miss data:

```
>>> lo_df.head()
```

Typically we will be mainly interested in two of these metric scores:

`TPR` - True positive (i.e. hit) rate: number of true positives, divided by number of true positives + number of false negatives

`corr_thrbin` - Pearson correlation between the lesion amge and the thresholded, binarized connectome edge image (group-level visitation map)

```
>>> lo_df[['TPR', 'corr_thrbin']].iloc[:10].T
```

We can obtain these numbers as a 'modification matrix' (connectivity matrix)

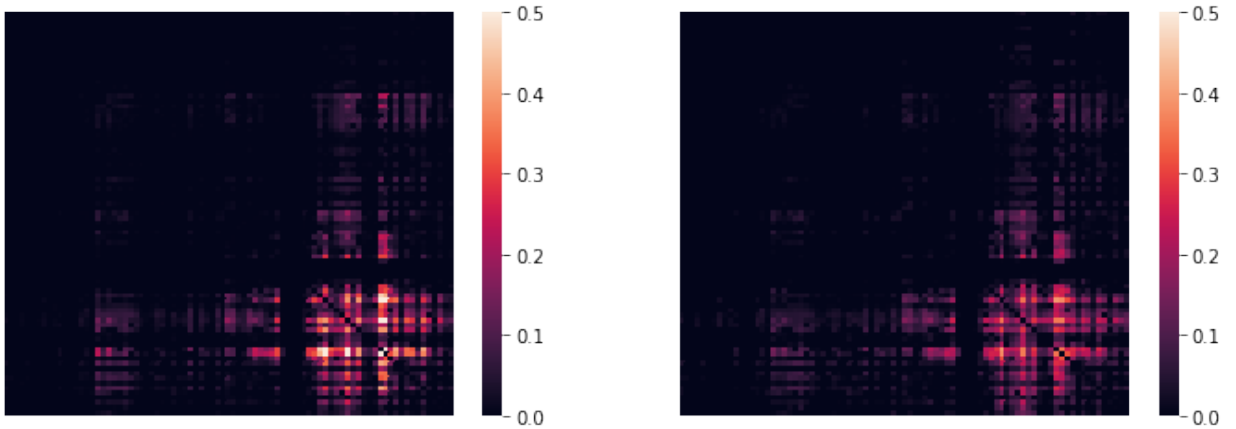
```
>>> tpr_adj = nx.to_pandas_adjacency(lo_nx,weight='TPR')
>>> cpr_adj = nx.to_pandas_adjacency(lo_nx,weight='corr_thrbin')
```

These two maps are, unsurprisingly, very similar:

```
>>> np.corrcoef(tpr_adj.values.ravel(), cpr_adj.values.ravel())
```

```
array([[1.        , 0.96271946],
       [0.96271946, 1.        ]])
```

```
>>> fig, ax = plt.subplots(ncols=2, figsize=(12,4))
>>> sns.heatmap(tpr_adj,xticklabels='',yticklabels='',vmin=0,vmax=0.5,ax=ax[0]);
>>> sns.heatmap(cpr_adj,xticklabels='',yticklabels='',vmin=0,vmax=0.5,ax=ax[1]);
```



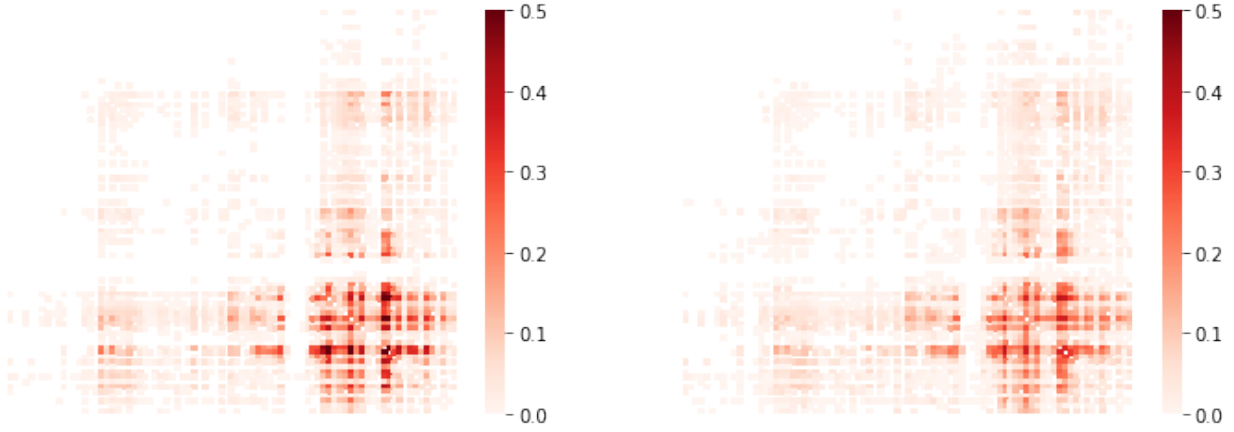
(. . . with an alternative color scheme. . .)

```
>>> fig, ax = plt.subplots(ncols=2, figsize=(12,4))
>>> sns.heatmap(tpr_adj, xticklabels='',yticklabels='',cmap='Reds',
>>>                    mask=tpr_adj.values==0,vmin=0,vmax=0.5,ax=ax[0]);
```

(continues on next page)

(continued from previous page)

```
>>> sns.heatmap(cpr_adj,xticklabels='',yticklabels='',cmap='Reds',
>>>                     mask=cpr_adj.values==0,vmin=0,vmax=0.5,ax=ax[1]);
```



We can list directly the most affected (greatest % overlap) connections,

```
>>> cw_vca.vfms.loc[lo_df.index].head()
```

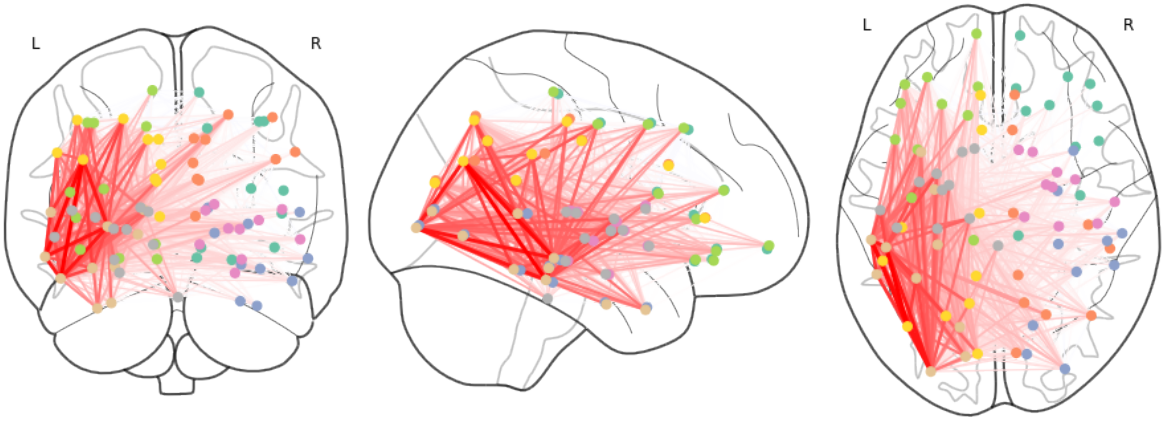
To plot the modification matrix information on a brain, we first need to some spatial locations to plot as nodes. For these, we calculate (an approprixation to) each atlas region's centriod location:

```
>>> parc_img = cw_vca.region_nii
>>> parc_dat = parc_img.get_data()
>>> parc_vals = np.unique(parc_dat)[1:]

>>> ccs = {roival: find_xyz_cut_coords(nib.Nifti1Image((dat==roival).astype(int),img.
↪affine),
>>>                                     activation_threshold=0) for roival in roivals}
>>> ccs_arr = np.array(ccs.values())
```

Now plotting on a glass brain:

```
>>> fig, ax = plt.subplots(figsize=(16,6))
>>> plot_connectome(tpr_adj.values,ccs_arr,axes=ax,edge_threshold=0.2,colorbar=True,
>>>                     edge_cmap='Reds',edge_vmin=0,edge_vmax=1.,
>>>                     node_color='lightgrey',node_kwargs={'alpha': 0.4});
>>> #edge_vmin=0,edge_vmax=1)
```

```
>>> fig, ax = plt.subplots(figsize=(16,6))
>>> plot_connectome(cpr_adj.values,ccs_arr,axes=ax)
```



The lines in this figure show network connections (drawn as a straight line between two nodes) whose atlas image volume have a non-zero level of overlap with the synthetic lesion volume. Transparency and colour intensity indicate the magnitude of overlap. Thus the thickest, brightest red lines correspond to tracts that pass directly through the centre of the synthetic lesion mask, and for whom the lesion overlaps with a substantial amount of their total volume. Light, thinner lines, extending to/from the contralateral hemisphere and frontal cortex, correspond to connections with a proportionally smaller degree of lesion load.

CHAPTER 9

Connectivity-based decomposition of white matter tracts

CHAPTER 10

Setting up TVB simulations from ConWhAt outputs