nighres Documentation

Release 1.0.0b9

Julia Huntenburg

May 22, 2018
# Getting started

1. Installing Nighres 3
2. Nighres usage examples 7
3. Brain 15
4. Cortex 21
5. Surface 25
6. Laminar 27
7. Data 29
8. Input/Output 31
9. Data handling and the niimg 35
10. Saving outputs 37
11. Levelsets 39
12. Overview 41
13. Setting up 43
14. Wrapping an existing CBS Tools class 45
15. Adding a new Python function 47
16. Writing examples 49
17. Adapting the docs 51
18. Making a Pull Request 53
Nighres is a Python package for processing of high-resolution neuroimaging data. It developed out of CBS High-Res Brain Processing Tools and aims to make those tools easier to install, use and extend.

**Warning:** Nighres is currently still in beta stage
CHAPTER 1

Installing Nighres

1.1 From PyPI

You can download the latest stable release of Nighres from PyPI.

Because parts of the package have to be built locally it is currently not possible to use pip install directly from PyPI. Instead, please download and unpack the tarball to Build Nighres. (Or use the Docker image)

1.2 From Github

You can also get the latest version from Github

```
git clone https://github.com/nighres/nighres
```

Or download and unpack the zip file from Github under Clone and download -> Download ZIP

1.3 Build Nighres

1. Make sure you have JCC installed. You need to both get the package via your package manager and install using pip to make it accessible to the Python interpreter, e.g:

```
sudo apt-get install jcc
pip install jcc
```

2. Navigate to the Nighres directory you downloaded (and unpacked) and run the build script:

```
./build.sh
```

The build might take a while because it pulls the original Java code from https://github.com/piloubazin/cbstools-public and builds the wrappers using JCC.
3. Install the Python package:

```
pip install .
```

### 1.4 Testing the installation

You can often catch installation problems by simply import Nighres in Python. Make sure to navigate out of the directory from which you installed to make sure Nighres has actually been installed correctly and can be accessed from any location.

```
python -c "import nighres"
```

If that works, you can try running one of the examples. You can find them inside the unpacked Nighres directory, in the subdirectory `examples`. Alternatively, you can also download the `examples` from the online documentation.

### 1.5 Dependencies

To build Nighres you need:

- JCC
- Java, a version that is 1.7 or higher

To run Nighres depends on:

- Numpy
- Nibabel

If not already available, these packages are automatically installed when you run pip install.

**Todo:** Check and include version dependencies

### 1.6 Docker

To quickly try out nighres in a preset, batteries-included environment, you can use the included Dockerfile, which includes Ubuntu 14 Trusty, openJDK-8, nighres, and Jupyter Notebook. The only thing you need to install is Docker, a lightweight container platform that runs on Linux, Windows and Mac OS X.

To build the Docker image, do the following:

```
git clone https://github.com/nighres/nighres
cd nighres
docker build . -t nighres
```

To run the Docker container:

```
docker run --rm -p 8888:8888 nighres
```

Now go with your browser to `https://localhost:8888` to start a notebook. You should be able to import nighres by entering:
import nighres

into the first cell of your notebook.

Usually you also want to have access to some data when you run nighres. You can grant the Docker container access to a data folder on your host OS by using the -v tag when you start the container:

docker run --rm -v /home/me/my_data:/data -p 8888:8888 nighres

Now, in your notebook you will be able to access your data on the path /data

## 1.7 Additional dependencies (optional)

### Plotting in the examples

- Nilearn and its dependencies, if Nilearn is not installed, plotting in the examples will be skipped and you can view the results in any other nifti viewer

### Building the documentation

- sphinx
- sphinx-gallery
- matplotlib
- sphinx-rtd-theme (pip install sphinx-rtd-theme)
- pillow (pip install pillow)

### Using the docker image

- Docker
2.1 Tissue classification from MP2RAGE data

This example shows how to obtain a tissue classification from MP2RAGE data by performing the following steps:

1. Downloading open MP2RAGE dataset using `nighres.data.download_7T_TRT()`
2. Remove the skull and create a brain mask using `nighres.brain.mp2rage_skullstripping()`
3. Atlas-guided tissue classification using MGDM `nighres.brain.mgdm_segmentation()`

### 2.1.1 Import and download

First we import `nighres` and the `os` module to set the output directory. Make sure to run this file in a directory you have write access to, or change the `out_dir` variable below.

```python
import nighres
import os

out_dir = os.path.join(os.getcwd(), 'nighres_examples/tissue_classification')
```

We also try to import Nilearn plotting functions. If Nilearn is not installed, plotting will be skipped.

```python
skip_plots = False
try:
    from nilearn import plotting
```

---

except ImportError:
    skip_plots = True
    print('Nilearn could not be imported, plotting will be skipped')

Now we download an example MP2RAGE dataset. It is the structural scan of the first subject, first session of the 7T Test-Retest dataset published by Gorgolewski et al (2015)².

dataset = nighres.data.download_7T_TRT(out_dir)

### 2.1.2 Skull stripping

First we perform skull stripping. Only the second inversion image is required to calculate the brain mask. But if we input the T1map and T1w image as well, they will be masked for us. We also save the outputs in the `out_dir` specified above and use a subject ID as the base file name.

```
skullstripping_results = nighres.brain.mp2rage_skullstripping(
    second_inversion=dataset['inv2'],
    t1_weighted=dataset['t1w'],
    t1_map=dataset['t1map'],
    save_data=True,
    file_name='sub001_sess1',
    output_dir=out_dir)
```

**Tip:** in Nighres functions that have several outputs return a dictionary storing the different outputs. You can find the keys in the docstring by typing `nighres.brain.mp2rage_skullstripping?` or list them with `skullstripping_results.keys()`.

To check if the skull stripping worked well we plot the brain mask on top of the original image. You can also open the images stored in `out_dir` in your favourite interactive viewer and scroll through the volume.

Like Nilearn, we use Nibabel SpatialImage objects to pass data internally. Therefore, we can directly plot the outputs using Nilearn plotting functions.

```
if not skip_plots:
    plotting.plot_roi(skullstripping_results['brain_mask'], dataset['t1w'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')
```

---

² Gorgolewski et al (2015). A high resolution 7-Tesla resting-state fMRI test-retest dataset with cognitive and physiological measures. DOI: 10.1038/sdata.2014.54
2.1.3 MGDM classification

Next, we use the masked data as input for tissue classification with the MGDM algorithm. MGDM works with a single contrast, but can be improved with additional contrasts. In this case we use the T1-weighted image as well as the quantitative T1map.

```python
mgdm_results = nighres.brain.mgdm_segmentation(
    contrast_image1=skullstripping_results['t1w_masked'],
    contrast_type1="Mp2rage7T",
    contrast_image2=skullstripping_results['t1map_masked'],
    contrast_type2="T1map7T",
    save_data=True, file_name="sub001_sess1",
    output_dir=out_dir)
```

Now we look at the topology-constrained segmentation MGDM created

```python
if not skip_plots:
    plotting.plot_img(mgdm_results['segmentation'],
                      vmin=1, vmax=50, cmap='cubehelix', colorbar=True,
                      annotate=False, draw_cross=False)
```
MGDM also creates an image which represents for each voxel the distance to its nearest border. It is useful to assess where partial volume effects may occur.

```python
if not skip_plots:
    plotting.plot_anat(mgdm_results['distance'], vmin=0, vmax=20, annotate=False, draw_cross=False, colorbar=True)
```

If the example is not run in a jupyter notebook, render the plots:

```python
if not skip_plots:
    plotting.show()
```

### 2.1.4 References

**Total running time of the script:** (0 minutes 0.000 seconds)

**Note:** Click [here](#) to download the full example code

#### 2.2 Cortical depth estimation from MGDM segmentation

This example shows how to obtain a cortical laminar depth representation from an MGDM segmentation result with the following steps:

1. Get a segmentation result from the `tissue_classification` example
2. Extract the cortex of the left hemisphere with `nighres.brain.extract_brain_region()`
3. Cortical reconstruction with CRUISE `nighres.cortex.cruise_cortex_extraction()`\(^1\)
4. Anatomical depth estimation through `nighres.laminar.volumetric_layering()`\(^2\)

Important note: this example assumes you have run the tissue classification example first (example_tissue_classification.py)


2.2.1 Import and point to previous example

First we import `nighres` and the `os` module to set the output directory. Make sure to run this file in a directory you have write access to, or change the `out_dir` variable below.

```python
import nighres
import os

in_dir = os.path.join(os.getcwd(), 'nighres_examples/tissue_classification')
out_dir = os.path.join(os.getcwd(), 'nighres_examples/cortical_depth_estimation')
```

We also try to import Nilearn plotting functions. If Nilearn is not installed, plotting will be skipped.

```python
skip_plots = False
try:
    from nilearn import plotting
except ImportError:
    skip_plots = True
    print('Nilearn could not be imported, plotting will be skipped')
```

Now we pull the MGDM results from previous example.

```python
segmentation = os.path.join(in_dir, 'sub001_sess1_mgdm_seg.nii.gz')
boundary_dist = os.path.join(in_dir, 'sub001_sess1_mgdm_dist.nii.gz')
max_labels = os.path.join(in_dir, 'sub001_sess1_mgdm_lbls.nii.gz')
max_probas = os.path.join(in_dir, 'sub001_sess1_mgdm_mems.nii.gz')
```

2.2.2 Region Extraction

Here we pull from the MGDM output the needed regions for cortical reconstruction: the GM cortex (`region`), the underlying WM (with filled subcortex and ventricles, `inside`) and the surrounding CSF (with masked regions, `background`).

```python
cortex = nighres.brain.extract_brain_region(segmentation=segmentation,
                                           levelset_boundary=boundary_dist,
                                           maximum_membership=max_probas,
                                           maximum_label=max_labels,
                                           extracted_region='left_cerebrum',
                                           save_data=True,
                                           file_name='sub001_sess1_left_cerebrum',
                                           output_dir=out_dir)
```

**Tip:** in Nighres functions that have several outputs return a dictionary storing the different outputs. You can find the keys in the docstring by typing `nighres.brain.mp2rage_extract_brain_region?` or list them with `cortex.keys()`.

To check if the extraction worked well we plot the GM and WM probabilities. You can also open the images stored in `out_dir` in your favourite interactive viewer and scroll through the volume.

Like Nilearn, we use Nibabel SpatialImage objects to pass data internally. Therefore, we can directly plot the outputs using Nilearn plotting functions.
2.2.3 CRUISE cortical reconstruction

Next, we use the extracted data as input for cortex reconstruction with the CRUISE algorithm. CRUISE works with the membership functions as a guide and the WM inside mask as a (topologically spherical) starting point to grow a refined GM/WM boundary and CSF/GM boundary.
Now we look at the topology-constrained segmentation CRUISE created

```python
if not skip_plots:
    plotting.plot_img(cruise['cortex'],
                      vmin=0, vmax=2, cmap='cubehelix', colorbar=True,
                      annotate=False, draw_cross=False)
```

### 2.2.4 Volumetric layering

Finally, we use the GM/WM boundary (GWB) and CSF/GM boundary (CGB) from CRUISE to compute cortical depth with a volume-preserving technique

```python
depth = nighres.laminar.volumetric_layering(
    inner_levelset=cruise['gwb'],
    outer_levelset=cruise['cgb'],
    n_layers=4,
    save_data=True,
    file_name="sub001_sess1_left_cerebrum",
    output_dir=out_dir)
```

Now we look at the laminar depth estimates

```python
if not skip_plots:
    plotting.plot_img(depth['depth'],
                      vmin=0, vmax=1, cmap='autumn', colorbar=True,
                      annotate=False, draw_cross=False)
```
If the example is not run in a jupyter notebook, render the plots:

```python
if not skip_plots:
    plotting.show()
```

### 2.2.5 References

**Total running time of the script:** (0 minutes 0.000 seconds)
3.1 mgdm_segmentation

nighres.brain.mgdm_segmentation(contrast_image1, contrast_type1, contrast_image2=None, contrast_type2=None, contrast_image3=None, contrast_type3=None, contrast_image4=None, contrast_type4=None, n_steps=5, max_iterations=800, topology='wcs', atlas_file=None, topology_lut_dir=None, adjust_intensity_priors=False, compute_posterior=False, diffuse_probabilities=False, save_data=False, output_dir=None, file_name=None)

MGDM segmentation

Estimates brain structures from an atlas for MRI data using a Multiple Object Geometric Deformable Model (MGDM)

Parameters

- **contrast_image1**(niimg) – First input image to perform segmentation on
- **contrast_type1**(str) – Contrast type of first input image, must be listed as a prior in used atlas(specified in atlas_file)
- **contrast_image2**(niimg, optional) – Additional input image to inform segmentation, must be in the same space as constrast_image1, requires contrast_type2
- **contrast_type2**(str, optional) – Contrast type of second input image, must be listed as a prior in used atlas (specified in atlas_file)
- **contrast_image3**(niimg, optional) – Additional input image to inform segmentation, must be in the same space as constrast_image1, requires contrast_type3
- **contrast_type3**(str, optional) – Contrast type of third input image, must be listed as a prior in used atlas (specified in atlas_file)
- **contrast_image4**(niimg, optional) – Additional input image to inform segmentation, must be in the same space as constrast_image1, requires contrast_type4
• **contrast_type4** *(str, optional)* – Contrast type of fourth input image, must be listed as a prior in used atlas (specified in **atlas_file**)

• **n_steps** *(int, optional)* – Number of steps for MGDM (default is 5, set to 0 for quick testing of registration of priors, which does not perform true segmentation)

• **max_iterations** *(int, optional)* – Maximum number of iterations per step for MGDM (default is 800, set to 1 for quick testing of registration of priors, which does not perform true segmentation)

• **topology** *(('wcs', 'no'), optional)* – Topology setting, choose ‘wcs’ (well-composed surfaces) for strongest topology constraint, ‘no’ for no topology constraint (default is ‘wcs’)

• **atlas_file** *(str, optional)* – Path to plain text atlas file (default is stored in DEFAULT_ATLAS) or atlas name to be searched in ATLAS_DIR

• **topology_lut_dir** *(str, optional)* – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)

• **adjust_intensity_priors** *(bool)* – Adjust intensity priors based on dataset (default is False)

• **compute_posterior** *(bool)* – Compute posterior probabilities for segmented structures (default is False)

• **diffuse_probabilities** *(bool)* – Regularize probability distribution with a non-linear diffusion scheme (default is False)

• **save_data** *(bool)* – Save output data to file (default is False)

• **output_dir** *(str, optional)* – Path to desired output directory, will be created if it doesn’t exist

• **file_name** *(str, optional)* – Desired base name for output files with file extension (suffixes will be added)

**Returns**

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

• **segmentation (niimg)**: Hard brain segmentation with topological constraints (if chosen) (_mgdm_seg)

• **labels (niimg)**: Maximum tissue probability labels (_mgdmLbls)

• **memberships (niimg)**: Maximum tissue probability values, 4D image where the first dimension shows each voxel’s highest probability to belong to a specific tissue, the second dimension shows the second highest probability to belong to another tissue etc. (_mgdm_mems)

• **distance (niimg)**: Minimum distance to a segmentation boundary (_mgdm_dist)

**Return type** *dict*

**Notes**

Original Java module by Pierre-Louis Bazin. Algorithm details can be found in


References

3.1.1 Examples using nighres.brain.mgdm_segmentation

• Tissue classification from MP2RAGE data

3.2 mp2rage_skullstripping

nighres.brain.mp2rage_skullstripping

Parameters

• second_inversion (niimg) – Second inversion image derived from MP2RAGE sequence
• t1_weighted (niimg) – T1-weighted image derived from MP2RAGE sequence (also referred to as “uniform” image) At least one of t1_weighted and t1_map is required
• t1_map (niimg) – Quantitative T1 map image derived from MP2RAGE sequence At least one of t1_weighted and t1_map is required
• skip_zero_values (bool) – Ignores voxels with zero value (default is True)
• topology_lut_dir (str, optional) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
• save_data (bool) – Save output data to file (default is False)
• output_dir (str, optional) – Path to desired output directory, will be created if it doesn’t exist
• file_name (str, optional) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

• brain_mask (niimg): Binary brain mask (_strip_mask)
• inv2_masked (niimg): Masked second inversion image (_strip_inv2)
• t1w_masked (niimg): Masked T1-weighted image (_strip_t1w)
• t1map_masked (niimg): Masked T1 map (_strip_t1map)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Details on the MP2RAGE sequence can be found in¹

¹ Marques et al. (2010). MP2RAGE, a self bias-field corrected sequence for improved segmentation and T1-mapping at high field. DOI: 10.1016/j.neuroimage.2009.10.002
References

3.2.1 Examples using nighres.brain.mp2rage_skullstripping

- Tissue classification from MP2RAGE data

3.3 extract_brain_region

nighres.brain.extract_brain_region(segmentation, levelset_boundary, maximum_membership, maximum_label, extracted_region, atlas_file=None, normalize_probabilities=False, partial_volume_distance=1.0, save_data=False, output_dir=None, file_name=None)

Extract Brain Region

Extracts masks, probability maps and levelset surfaces for specific brain regions and regions from a Multiple Object Geometric Deformable Model (MGDM) segmentation result.

Parameters

- segmentation (niimg) – Segmentation result from MGDM.
- levelset_boundary (niimg) – Levelset boundary from MGDM.
- maximum_membership (niimg) – 4D image of the maximum membership values from MGDM.
- maximum_label (niimg) – 4D image of the maximum labels from MGDM.
- atlas_file (str, optional) – Path to plain text atlas file (default is stored in DEFAULT_ATLAS). or atlas name to be searched in ATLAS_DIR
- extracted_region (values) – Region to be extracted from the MGDM segmentation.
- normalize_probabilities (bool) – Whether to normalize the output probabilities to sum to 1 (default is False).
- estimate_tissue_densities (bool) – Wheter to recompute partial volume densities from the probabilities (slow, default is False).
- partial_volume_distance (float) – Distance in mm to use for tissues densities, if recomputed (default is 1mm).
- save_data (bool) – Save output data to file (default is False)
- output_dir (str, optional) – Path to desired output directory, will be created if it doesn’t exist
- file_name (str, optional) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets, # stands for shorthand names of the different extracted regions, respectively: rcr, lcr, cr, cb, cbs, sub, an, fn)
- region_mask (niimg): Hard segmentation mask of the (GM) region of interest (_xmask_#gm)
- inside_mask (niimg): Hard segmentation mask of the (WM) inside of the region of interest (_xmask_#wm)
- background_mask (niimg): Hard segmentation mask of the (CSF) region background (_xmask_#bg)
- region_proba (niimg): Probability map of the (GM) region of interest (_xproba_#gm)
- inside_proba (niimg): Probability map of the (WM) inside of the region of interest (_xproba_#wm)
- background_proba (niimg): Probability map of the (CSF) region background (_xproba_#bg)
- region_lvl (niimg): Levelset surface of the (GM) region of interest (_xlvl_#gm)
- inside_lvl (niimg): Levelset surface of the (WM) inside of the region of interest (_xlvl_#wm)
- background_lvl (niimg): Levelset surface of the (CSF) region background (_xlvl_#bg)

Return type: dict

Notes

Original Java module by Pierre-Louis Bazin.

3.3.1 Examples using nighres.brain.extract_brain_region

- Cortical depth estimation from MGDM segmentation
4.1 cruise_cortex_extraction

```python
nighres.cortex.cruise_cortex_extraction(init_image, wm_image, gm_image, csf_image, vd_image=None, data_weight=0.4, regularization_weight=0.1, max_iterations=500, normalize_probabilities=False, correct_wm_pv=True, wm_dropoff_dist=1.0, topology='wcs', topology_lut_dir=None, save_data=False, output_dir=None, file_name=None)
```

CRUISE cortex extraction

Segments the cortex from a whole brain segmented data set with the CRUISE method (includes customized partial voluming corrections and the Anatomically-Consistent Enhancement (ACE) of sulcal fundi).

Note that the main input images are generated by the nighres module `nighres.brain.extract_brain_region()`.

**Parameters**

- `init_image(niimg)` – Initial white matter (WM) segmentation mask (binary mask>0 inside WM)
- `wm_image(niimg)` – Filled WM probability map (values in [0,1], including subcortical GM and ventricles)
- `gm_image(niimg)` – Cortical gray matter (GM) probability map (values in [0,1], highest inside the cortex)
- `csf_image(niimg)` – Sulcal cerebro-spinal fluid (CSF) and background probability map (values in [0,1], highest in CSF and masked regions)
- `vd_image(niimg, optional)` – Additional probability map of vessels and dura mater to be excluded
- `data_weight(float)` – Weighting of probability-based balloon forces in CRUISE (default 0.4, sum of {data_weight,regularization_weight} should be below or equal to 1)
- **regularization_weight** (*float*) – Weighting of curvature regularization forces in CRUISE (default 0.1, sum of \{data_weight,regularization_weight\} should be below or equal to 1)

- **max_iterations** (*int*) – Maximum number of iterations in CRUISE (default is 500)

- **normalize_probabilities** (*bool*) – Whether to normalize the wm, gm, and csf probabilities (default is False)

- **correct_wm_pv** (*bool*) – Whether to correct for WM partial voluming in gyral crowns (default is True)

- **wm_dropoff_dist** (*float*) – Distance parameter to lower WM probabilities away from current segmentation (default is 1.0 voxel)

- **topology** (*{'wcs', 'no'}*) – Topology setting, choose ‘wcs’ (well-composed surfaces) for strongest topology constraint, ‘no’ for no topology constraint (default is ‘wcs’)

- **topology_lut_dir** (*str*) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)

- **save_data** (*bool*) – Save output data to file (default is False)

- **output_dir** (*str*, optional) – Path to desired output directory, will be created if it doesn’t exist

- **file_name** (*str*, optional) – Desired base name for output files with file extension (suffixes will be added)

- **Returns**

  Dictionary collecting outputs under the following keys (suffix of output files in brackets)

  - **cortex** (*niimg*): Hard segmentation of the cortex with labels background=0, gm=1, and wm=2 (_cruise_cortex)

  - **gwb** (*niimg*): Gray-White matter Boundary (GWB) level set function (_cruise_gwb)

  - **cgb** (*niimg*): CSF-Gray matter Boundary (CGB) level set function (_cruise_cgb)

  - **avg** (*niimg*): Central level set function, obtained as geometric average of GWB and CGB (not the middle depth of the cortex, use volumetric_layering if you want accurate depth measures) (_cruise_avg)

  - **thickness** (*niimg*): Simple cortical thickness estimate: distance to the GWB and CGB surfaces, in mm (_cruise_thick)

  - **pwm** (*niimg*): Optimized WM probability, including partial volume and distant values correction (_cruise_pwm)

  - **pgm** (*niimg*): Optimized GM probability, including CSF sulcal ridges correction (_cruise_pgm)

  - **pcsf** (*niimg*): Optimized CSF probability, including sulcal ridges and vessel/dura correction (_cruise_pwm)

- **Return type** *dict*

- **Notes**

  Original algorithm by Xiao Han. Java module by Pierre-Louis Bazin. Algorithm details can be found in

References

4.1.1 Examples using nighres.cortex.cruise_cortex_extraction

- Cortical depth estimation from MGDM segmentation
Surface

5.1 probability_to_levelset

```
nighres.surface.probability_to_levelset(probability_image, save_data=False, output_dir=None, file_name=None)
```

Levelset from tissue classification

Creates a levelset surface representations from a probabilistic or deterministic tissue classification. The levelset indicates each voxel’s distance to the closest boundary. It takes negative values inside and positive values outside of the brain.

**Parameters**

- `probability_image` (*niimg*) – Tissue segmentation to be turned into levelset. Values should be in [0, 1], either a binary mask or defining the boundary at 0.5.
- `save_data` (*bool*) – Save output data to file (default is False)
- `output_dir` (*str*, optional) – Path to desired output directory, will be created if it doesn’t exist
- `file_name` (*str*, optional) – Desired base name for output files with file extension (suffixes will be added)

**Returns** Levelset representation of surface (output file suffix _levelset)

**Return type** *niimg*

**Notes**

Original Java module by Pierre-Louis Bazin
6.1 volumetric_layering

nighres.laminar.volumetric_layering(inner_levelset, outer_levelset, n_layers=4, topology_lut_dir=None, save_data=False, output_dir=None, file_name=None)

Equivolumetric layering of the cortical sheet.

Parameters

• inner_levelset (niimg) – Levelset representation of the inner surface, typically GM/WM surface
• outer_levelset (niimg) – Levelset representation of the outer surface, typically GM/CSF surface
• n_layers (int, optional) – Number of layers to be created (default is 10)
• topology_lut_dir (str, optional) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
• save_data (bool) – Save output data to file (default is False)
• output_dir (str, optional) – Path to desired output directory, will be created if it doesn’t exist
• file_name (str, optional) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

• depth (niimg): Continuous depth from 0 (inner surface) to 1 (outer surface) (_layering_depth)
• layers (niimg): Discrete layers from 1 (bordering inner surface) to n_layers (bordering outer surface) (_layering_layers)
• boundaries (niimg): Levelset representations of boundaries between all layers in 4D (_layering_boundaries)

**Return type**: dict

**Notes**

Original Java module by Miriam Waehnert, Pierre-Louis Bazin and Juliane Dinse. Algorithm details can be found in

**References**

6.1.1 Examples using `nighres.laminar.volumetric_layering`

• Cortical depth estimation from MGDM segmentation

6.2 profile_sampling

`nighres.laminar.profile_sampling(profile_surface_image, intensity_image, save_data=False, output_dir=None, file_name=None)`

Sampling data on multiple intracortical layers

**Parameters**

- **profile_surface_image (niimg)** – 4D image containing levelset representations of different intracortical surfaces on which data should be sampled
- **intensity_image (niimg)** – Image from which data should be sampled
- **save_data (bool)** – Save output data to file (default is False)
- **output_dir (str, optional)** – Path to desired output directory, will be created if it doesn’t exist
- **file_name (str, optional)** – Desired base name for output files with file extension (suffixes will be added)

**Returns** 4D profile image, where the 4th dimension represents the profile for each voxel (output file suffix _profiles)

**Return type**: niimg

**Notes**

Original Java module by Pierre-Louis Bazin and Juliane Dinse

---

7.1 download_data

def download_7T_TRT(data_dir, overwrite=False):
    Downloads the MP2RAGE data of subject 001, session 1 of the 7T Test-Retest dataset published by Gorgolewski et al (2015)\(^1\)

    Parameters
    - *data_dir* (*str*) – Writeable directory in which downloaded files should be stored. A subdirectory called ‘7T_TRT’ will be created in this location.
    - *overwrite* (*bool*) – Overwrite existing files in the same exact path (default is False)

    Returns
    Dictionary with keys pointing to the location of the downloaded files
    - *inv2* : path to second inversion image
    - *t1w* : path to T1-weighted (uniform) image
    - *t1map* : path to quantitative T1 image

    Return type  *dict*

    Notes

    The full dataset is available at [http://openscience.cbs.mpg.de/7t_trt/](http://openscience.cbs.mpg.de/7t_trt/)

---

References

7.1.1 Examples using `nighres.data.download_7T_TRT`

- Tissue classification from MP2RAGE data
8.1 io_volume

nighres.io.load_volume(volume)
Load volumetric data into a Nibabel SpatialImage

Parameters:
volume (niimg) – Volumetric data to be loaded, can be a path to a file that nibabel can load, or a Nibabel SpatialImage

Returns:
image

Return type: Nibabel SpatialImage

Notes

Originally created as part of Laminar Python [1].

References

nighres.io.save_volume(filename, volume, dtype='float32', overwrite_file=True)
Save volumetric data that is a Nibabel SpatialImage to a file

Parameters:

- filename (str) – Full path and filename under which volume should be saved. The extension determines the file format (must be supported by Nibabel)
- volume (Nibabel SpatialImage) – Volumetric data to be saved
- dtype (str, optional) – Datatype in which volumetric data should be stored (default is float32)
- overwrite_file (bool, optional) – Overwrite existing files (default is True)
Notes

Originally created as part of Laminar Python [1].

References

8.2 io_mesh

nighres.io.load_mesh_geometry(surf_mesh)
Load a mesh geometry into a dictionary with entries “coords” and “faces”

Parameters

surf_mesh – Mesh geometry to be loaded, can be a path to a file (currently supported formats are freesurfer geometry formats, gii and ASCII-coded vtk, ply or obj) or a dictionary with the keys “coords” and “faces”

Returns
Dictionary with a numpy array with key “coords” for a Numpy array of the x-y-z coordinates of the mesh vertices and key “faces” for a Numpy array of the the indices (into coords) of the mesh faces

Return type
dict

Notes

Originally created as part of Laminar Python [1].

References

nighres.io.save_mesh_geometry(filename, surf_dict)
Saves surface mesh geometry to file

Parameters

• filename (str) – Full path and filename under which surfaces data should be saved. The extension determines the file format. Currently supported are freesurfer geometry formats, gii and ASCII-coded vtk, obj, ply
• surf_dict (dict) – Surface mesh geometry to be saved. Dictionary with a numpy array with key “coords” for a Numpy array of the x-y-z coordinates of the mesh vertices and key “faces2 for a Numpy array of the the indices (into coords) of the mesh faces

Notes

Originally created as part of Laminar Python [1].

References

nighres.io.load_mesh_data(surf_data, gii_darray=None)
Loads mesh data into a Numpy array

Parameters
• **surf_data** – Mesh data to be loaded, can be a Numpy array or a path to a file. Currently supported formats are freesurfer data formats (mgz, curv, sulc, thickness, annot, label), nii, gii, ASCII-coded vtk and txt

• **gii_darray** (*int, optional*) – Index of gii data array to load (default is to load all)

**Returns** Numpy array containing the data

**Return type** np.ndarray

**Notes**

Originally created as part of Laminar Python [1]_

**References**

nighres.io.save_mesh_data(filename, surf_data)
Saves surface data that is a Numpy array to file

**Parameters**

• **filename** (*str*) – Full path and filename under which surfaces data should be saved. The extension determines the file format. Currently supported are freesurfer formats curv, thickness, sulc and ASCII-coded txt

• **surf_data** (*np.ndarray*) – Surface data to be saved

**Notes**

Originally created as part of Laminar Python [1]_

**References**
CHAPTER 9

Data handling and the niimg

Nighres represents data internally using NibabelSpatialImage objects which are referred to as niimg. Much of the input and output functionality has been adopted or inspired from Nilearn's conventions for data handling.

Todo: Explanation why this is useful and little example how it works, also mention dictionary outputs
Saving outputs

Each Nighres processing interface allows you to save the outputs by setting the `save_data` parameter to `True`. If this parameter is not specified, it defaults to `False` and the data is returned as a data object (see *Data handling and the niimg*) but not saved to disk.

If `save_data` is set to `True`, Nighres applies the following logic:

**Output directory**

1. If `output_dir` is specified, the data is saved there. In case `output_dir` doesn’t exist, it is created.
2. If `output_dir` is not specified, Nighres tries to use the location of an input file as the location for saving. This only works if the input is a file name and not a data object.
3. Otherwise, the data is saved in the current working directory.

**File names**

1. If `file_name` is specified, this name is used as a base to create the output names. A suffix is added to each output (you can see in the docstrings which suffix refers to which output). The extension of `file_name` specifies the format in which the output will be saved. If `file_name` has no extension, Nighres defaults to `nii.gz`.
2. If `file_name` is not specified, Nighres tries to use the name of an input file as a base name for saving. This only works if the input is indeed a file name and not a data object.
Levelsets

**Todo:** What is a levelset and why is Nighres using it, example of creating levelset using `nighres.surface.probability_to_levelset()`
You want to contribute to Nighres? That’s fantastic! A few words to get started:

**You don’t have to be an expert!**

Contributing to open source software is a great way of learning by doing, and others in the Nighres community will be happy to help you out if you have questions or get stuck.

**What to do**

Every contribution helps – whether it’s fixing a typo in the documentation or adding an entire new feature. Often it’s a good idea to start by fixing a smaller issue to get familiar with the process. If you don’t know where to start, you can look at the issue tracker.

**Git/Github**

To contribute you will need to know some basics about git and Github, which are not covered in this guide. There are lots of tutorials online (e.g. [https://www.atlassian.com/git/tutorials](https://www.atlassian.com/git/tutorials), especially “Getting started” and “Collaborating”)

**Questions?**

For specifics about code you want to contribute *Making a Pull Request* is a great way to start a discussion and get input. Otherwise, you can post a question on [neurostars.org](https://neurostars.org). Please use the tag **nighres** for your question.

**Ready to go?**

The next pages will take you through the steps you need from cloning the Nighres repository, to making a pull request.

---

**Note:** Help us improve this developer’s guide! If you find anything unclear or not covered, open an [issue on github](https://github.com/) or add what you were missing (see *Adapting the docs*).
CHAPTER 13

Setting up

1. Fork and clone the Nighres github repository

   You can find a good description here. Make sure that you set up your local clone to track both your fork (typically as origin) as well as the original Nighres repo (typically as upstream).

2. Make a new branch to work on

   ```
git checkout -b <branch_name>
   ```

   This is important. When you make a pull request later, we will likely ask you to rebase because the Nighres master might have moved on since you forked it. You need a clean master branch for that.

   Pick a descriptive branch name. For example, when you fix a bug in the function load_volume a good name is fix/load_volume. When you write a new interface called laminar_connectivity (that would be cool!) a good name is enh/laminar_connectivity.

   Make one branch for each new feature or fix. That way independent changes can be handled in different pull requests later.

3. Install in editable mode (optional)

   ```
pip install -e <path_to_nighres_directory>
   ```

   This way, when you import Nighres functions in Python, they will always come from the current version of the code in your Nighres directory. This is convenient for testing your code while developing.

   (Alternatively, you can stay inside your Nighres directory and import functions directly from there without installing at all)
4. Let the coding begin!

If you want to work on an existing function, you can most likely just make your changes, check if the Examples still run and the move on to Adapting the docs and Making a Pull Request.

If you want to add new functions be sure to check out our intructions on Wrapping an existing CBS Tools class or Adding a new Python function.

**Important:** Please adhere to PEP8 style conventions. This is easiest by using a Python linter which is available in most editors.
You want to wrap an existing CBS Tools module into Nighres? Great!

14.1 How does it work?

The CBS Tools modules have been first created for JIST. Most of them are however not directly dependent on the JIST and MIPAV libraries, which is why we can turn them into a JCC-compatible, lightweight library.

JCC requires that all data is passed as numbers, strings, and 1D arrays, so this is what the CBS Tools modules need to manipulate as input and output, and also why we created a second python layer for structured data handling. Once a CBS Tools module has been formatted to the adequate i/o structure, JCC wraps it and you can call it directly from python.

14.2 Standard procedures

Each JCC-wrapped CBS Tools module has three important functions: .setSomeParameter(), .getSomeResult(), and .execute(). You need to set the inputs with meaningful values, execute the module, and retrieve the results. Note that default parameters are always assumed when possible, but a good wrapper should expose all parameters.

14.3 General wrapper content

1 Start the Java Virtual Machine (JVM)

   cbstools.initVM(initialheap='6000m', maxheap='6000m')

   Note that the initial and maximum amount of allocated memory may be adjusted or set up as a global parameter.

2 Create an instance of the module

   mymodule = cbstools.SomeCoolModule()
3 Set all the parameters and data arrays

```python
my_module.setThisImportantParameter(some_value)
my_module.setInputImage(cbstools.JArray('float')(my_image_data.flatten('F')).astype(float))
```

4 Run the module

```python
my_module.execute()
```

5 Retrieve the outputs

```python
my_result_data = np.reshape(np.array(my_module.getCoolResultImage(), dtype=np.float32), dimensions, 'F')
```

Note that because you are passing simple 1D arrays, you need to keep a record of image dimensions, resolutions, headers, etc.

14.4 Convenience python layer

We strongly advise (at least for inclusion into Nighres) to handle the inputs and outputs of python structures as we do, i.e. loading images from files or passing them as NiftiImage, and allowing to save all outputs automatically.

We also adhere to a somewhat strict formatting of the outputs with two suffixes, first for the module name, and second for the specific output, e.g. “_mgdm_seg” for the segmentation obtained from the mgdm_brain_segmentation module.

14.5 What if my favorite CBS Tools module is not ready to wrap?

Because preparing modules for wrapping requires some reformatting, only a few modules so far are ready. You can accelerate the process in two ways: 1) send a request to the CBS Tools developers or 2) write a reformatted module for CBS Tools yourself and make a pull request. All the currently wrapped CBS Tools modules follow the same formatting procedure, so transforming a JIST module definition into a core module (which then can be called either from JIST or from Nighres) should be reasonably easy in most cases.
Great, you want to contribute a Python function or module to Nighres!

15.1 Before you begin

Please take a look at the code that is under development or discussion. If you see something related get in touch with the developer(s) directly to combine your efforts.

Take a moment to familiarise yourself with the documentation and functions, it is likely that some helper functions that you will want to use (e.g., I/O) have already been coded and tested.

15.2 Let’s get into it

1. Follow the steps described in Setting up

2. Decide where to put your code

If the functionality that you are adding falls within the scope of a pre-existing submodule, edit or create files within the submodule. If it is entirely new (e.g., statistics), create a new submodule with its own dedicated subdirectory.

3. Coding If you are creating a submodule from scratch, an easy way to start is to copy the __init__.py and initial import statements from an existing module.
Please code PEP8 compliant, best to use a Python linter in your editor.

Please keep within our documentation guidelines.

Leave plenty of comments in your code so that others can understand and possibly adapt your code later.

Use the standard I/O interfaces wherever possible but also feel free to add additional I/O functionality as necessary.

Test your code internally. We aim to add unit tests in the future, feel free to make a start on that.

4. Write an example showcasing your new function

5. To get your code into the Nighres master follow Adapting the docs and Making a Pull Request

6. Pat yourself on the back, you have now joined the Nighres developer community!

A word on dependencies

One of Python’s strengths are the many different packages with specific functionalities. We try to keep the dependencies for Nighres as slim as possible, so please keep that in mind while you are coding. For example, if you need to perform a correlation and would normally use scipy.stats.pearsonr maybe you can get by with numpy.corrcoef (numpy is already a dependency, scipy isn’t). If additional packages are required, they should be pip installable and potentially relevant to other functionality that may be coded in the future.
Adapting the docs

Nighres uses Sphinx which makes it easy to automatically pull the docstrings of functions into the online documentation. This means once a function has a good docstring, the work is essentially done. It also means that writing great docstrings is important.

17.1 If you changed an existing function

Make sure to check if your changes also require changes in the docstring. Adapting an existing docstring is usually straightforward. If you are curious how your changes will look in the online docs, you can try out Building the docs locally.

17.2 If you added a new function

Make sure to write a comprehensive docstring. We use NumPy/SciPy docstring conventions. But the easiest is to just copy another function’s docstring and adapt. Then you need to add a few things to the nighres/doc directory:

1. Go to the subdirectory which refers to the submodule you added a function to, e.g. nighres/doc/laminar/
2. Create a new file named after your new function, e.g. laminar_connectivity.rst
3. Add the name of this file in the index.rst file of the submodule
4. Copy the content of another function’s rst file inside of your new file
5. Adapt the title and the two mentions of the function name in your new file to match your new function
6. Submit the changes to the docs along with your PR

Again, you can check how the changes you made will look in the online documentation by Building the docs locally.
17.3 If you added a new submodule

This is going to be a rare case. But if indeed added a new submodule in your PR, say its called `nighres.fancypants`:

1. In the doc directory, add a new subdirectory for your new module, e.g. `nighres/doc/fancypants`
2. In that subdirectory make a file called `index.rst`. Best to just copy the `index.rst` from another submodule. Then adapt the title in the file to the name of your new submodule, and remove all function names in the toctree.
3. Follow steps 3 to 5 from If you added a new function to add all functions of your new submodule
4. Add your submodule in the `index.rst` file in the `nighres/doc` main directory under the “Modules and Functions” toctree. In our case we would add the line `fancypants/index`
5. Submit the changes to the docs along with your PR

17.4 More than docstrings

You can also make changes to parts of the documentation that are not function docstrings. For example to help improving this guide!

The easiest way is to browse to the page in the online documentation that you want to change and click on Edit on Github to find the location of the file you have to change. For example, if you want to make a change to this page, the link would send you to https://github.com/nighres/nighres/blob/master/doc/developers/docs.rst, which tells you that in your local copy of the repo you should edit the file `nighres/doc/developers/docs.rst`.

Sphinx uses reStructuredText (reST) syntax. For formatting take a look at this Sphinx Cheatsheet.

You can also add whole new parts to the online documentation. This might require learning a bit more about Sphinx.

You can also open an issue and get help.

When you are done, you can check your changes by Building the docs locally.

17.5 Building the docs locally

If you make changes to the documentation, they will only appear online at http://nighres.readthedocs.io/en/latest/ once your PR has been merged and a new release was made. You can, however, build the docs locally for a preview:

1. Make sure to have all additional dependencies installed that you need to build the documentation
2. Navigate into your local copy of nighres/doc
3. Type make clean and then make html
4. Open an Internet browser and point it to the local html pages that were just built for you, e.g. file:///home/nighres/doc/_build/html/index.html

You can make changes and repeat this process until you are satisfied.
If you are working on changes to the Nighres code, let others know by opening a Pull Request (PR).

You don’t have to wait until your code is perfect, opening a PR early is a great way of letting other members of the community know that you are working on this. That avoids redundant work and gets you input at an early stage.

To open a PR:

1. Go to your fork of Nighres in your Github account
2. In the Branch drop-down menu, switch to the branch that you have worked on
3. Click on New pull request. You will see the changes that will be included in your PR.
4. Enter a descriptive title and small comment why these changes are necessary. If you are addressing a specific issue, refer to the issue number.
5. Click Create pull request and wait for responses (by default you will be informed about activity on your PR per e-mail)

After your PR is reviewed, you might be asked to make some changes. This is normal, almost no PR is merged without revisions. Pushing changes to your own branch automatically updates the code in the PR.

When all comments are addressed we will likely ask you to rebase, and then merge your code into the Nighres repository.

Credit

Nighres is a community-developed project made possible by these contributors. The project was born and continues to evolve at brainhack.

We thank the Google Summer of Code 2017 and INCF as a mentoring organization, for supporting the initial development phase of Nighres. See also the development blog.
When using Nighres in your research, please make sure to cite the references mentioned in the documentation of the particular functions you use. We are also preparing a dedicated Nighres paper. For now, we suggest you cite:

- Bazin et al. (2014) A computational framework for ultra-high resolution cortical segmentation at 7Tesla. DOI: 10.1016/j.neuroimage.2013.03.077
- Huntenburg et al. (2017) Laminar Python: Tools for cortical depth-resolved analysis of high-resolution brain imaging data in Python. DOI: 10.3897/rio.3.e12346
Index

C
  cruise_cortex_extraction() (in module nighres.cortex), 21

D
  download_7T_TRT() (in module nighres.data), 29

E
  extract_brain_region() (in module nighres.brain), 18

L
  load_mesh_data() (in module nighres.io), 32
  load_mesh_geometry() (in module nighres.io), 32
  load_volume() (in module nighres.io), 31

M
  mgdm_segmentation() (in module nighres.brain), 15
  mp2rage_skullstripping() (in module nighres.brain), 17

P
  probability_to_levelset() (in module nighres.surface), 25
  profile_sampling() (in module nighres.laminar), 28

S
  save_mesh_data() (in module nighres.io), 33
  save_mesh_geometry() (in module nighres.io), 32
  save_volume() (in module nighres.io), 31

V
  volumetric_layering() (in module nighres.laminar), 27