pynibs Documentation

Release 0.1

Konstantin Weise

Dec 09, 2022

CONTENTS

pyNIBS is a Pyhon package to work with non-invasive brain stimulation (NIBS) data, foremost from transcranial magnetic stimulation (TMS) experiments.

We created this package to allow the mapping of causal structure-function relationships with TMS. A lot of analyses is based on calculations of the induced electric fields (e-fields) – for this we rely on SimNIBS.

Install **pyNIBS** via pip:

```
pip install pynibs
```

or directly grab from our repository at gitlab.gwdg.de/tms-localization/pynibs

CHAPTER

ONE

CONTENTS.

1.1 pynibs package

1.1.1 Subpackages

pynibs.congruence package

pynibs.congruence holds the initial congruence factor implementation as described in Weise, Numssen, et al., 2020 [1]. This code is mainly stored here for reproducibility reasons. The current approach (Numssen et al., 2021; [2]) uses the *pynibs.regression* methods.

References

Submodules

pynibs.congruence.congruence module

pynibs.congruence.congruence.**cf_curveshift_kernel**(*e_curve*, *mep_curve*)

Curve congruence (overlap) measure for multiple MEP curves per element. Determines the average displacement between the MEP curves. The congruence factor is weighted by `median(E)` and summed up. This favors elements which have greater E, as these are more likely to produce MEPs.

$$dE = \begin{bmatrix} dE_{11} & dE_{12} & ... & dE_{1n} \\ dE_{21} & dE_{22} & ... & dE_{2n} \\ ... & ... & ... & ... \\ dE_{n1} & dE_{n2} & ... & dE_{nn} \end{bmatrix}$$

-> `congruence_factor ~ np.linalg.norm(dE)/median(E)/n_cond/2`

Parameters

- **e_curve** (*list of np.ndarray of float*) – (n_cond) List over all conditions of electric field values corresponding to the mep amplitudes.
- **mep_curve** (*list of np.ndarray of float*) – (n_cond) List over all conditions of mep values corresponding to the electric field.

Returns

congruence_factor – Congruence factor for the n_cond electric field and MEP curves.

Return type

float

pynibs.congruence.congruence.**cf_curveshift_workhorse**(*elm_idx_list*, *mep*, *mep_params*, *e*, *n_samples=100*)

Worker function for congruence factor computation - call from `multiprocessing.Pool`. Calculates congruence factor for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements. The computations are parallelized in terms of element indices (`elm_idx_list`). n_samples are taken from fitted_mep, within the range of the *Mep*.

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for.
- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions.
- **mep_params** (*np.ndarray of float*) – (n_mep_params_total) List of all mep parameters of curve fits used to calculate the MEP, accumulated into one array.
  - e.g. [mep_#1_para_#1, mep_#1_para_#2, mep_#1_para_#3, mep_#2_para_#1, mep_#2_para_#1, ...]
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of n_datasets of the electric field to compute the congruence factor for, e.g. `(e_mag, e_norm, e_tan)`. Each dataset is a list over all conditions containing the electric field component of interest.
  - `len(e) = n_cond`
  - `len(e[0]) = n_comp` (e.g: `e_mag = e[0]`))
- **n_samples** (*int, default=100*) – Number of data points to generate discrete mep and e curves.

Returns

congruence_factor – (n_roi, n_datasets) Congruence factor in each element specified in elm_idx_list and for each input dataset.

Return type

np.ndarray of float

pynibs.congruence.congruence.**cf_curveshift_workhorse_stretch_correction**(*elm_idx_list*, *mep*, *mep_params*, *e*, *n_samples=100*)

Worker function for congruence factor computation - call from multiprocessing.pool Calculates congruence factor for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements. The computations are parallelized in terms of element indices (`elm_idx_list`). `n_samples` are taken from fitted_mep, within the range of the *Mep*.

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for
- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions.
- **mep_params** (*np.ndarray of float*) – (n_mep_params_total) List of all mep parameters of curve fits used to calculate the MEP, accumulated into one array.
  - e.g. [mep_#1_para_#1, mep_#1_para_#2, mep_#1_para_#3, mep_#2_para_#1, mep_#2_para_#1, ...]
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of n_datasets of the electric field to compute the congruence factor for, e.g. `(e_mag, e_norm, e_tan)`. Each dataset is a list over all conditions containing the electric field component of interest
  - e.g.: `len(e) = n_cond`
  - `len(e[0]) = n_comp` (e.g: `e_mag = e[0]`))

- **n_samples** (*int, default=100*) – Number of data points to generate discrete mep and e curves.

Returns

congruence_factor – (n_roi, n_datasets) Congruence factor in each element specified in elm_idx_list and for each input dataset.

Return type

np.ndarray of float

pynibs.congruence.congruence.**cf_curveshift_workhorse_stretch_correction_new**(*mep*, *mep_params*, *e*, *n_samples=100*, *ref_idx=0*)

Worker function for congruence factor computation - call from `multiprocessing.Pool`. Calculates congruence factor for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements. The computations are parallelized in terms of element indices (`elm_idx_list`). `n_samples` are taken from fitted_mep, within the range of the *Mep*.

Parameters

- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions.
- **mep_params** (*np.ndarray of float [n_mep_params_total]*) – List of all mep parameters of curve fits used to calculate the MEP (accumulated into one array)
  - e.g. [`mep_#1_para_#1`, `mep_#1_para_#2`, `mep_#1_para_#3`, `mep_#2_para_#1`, `mep_#2_para_#1`, ...]
- **e** (*np.ndarray of float*) – (n_elm, n_cond) Electric field in elements.
- **n_samples** (*int, default=100*) – Number of data points to generate discrete mep and e curves.

Returns

congruence_factor – (n_elm) Congruence factor in each element specified in elm_idx_list and for each input dataset.

Return type

np.ndarray of float

pynibs.congruence.congruence.**cf_curveshift_workhorse_stretch_correction_sign_new**(*mep*, *mep_params*, *e*, *n_samples=100*, *ref_idx=0*)

Worker function for congruence factor computation - call from `multiprocessing.Pool`. Calculates congruence factor for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements. The computations are parallelized in terms of element indices (`elm_idx_list`). ``n_sample``s are taken from fitted_mep, within the range of the *Mep*.

Parameters

- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions.
- **mep_params** (*np.ndarray of float*) – (n_mep_params_total) List of all mep parameters of curve fits used to calculate the MEP, accumulated into one array.
  - e.g. [`mep_#1_para_#1`, `mep_#1_para_#2`, `mep_#1_para_#3`, `mep_#2_para_#1`, `mep_#2_para_#1`, ...]
- **e** (*np.ndarray of float*) – (n_elm, n_cond) Electric field in elements.
- **n_samples** (*int, default=100*) – Number of data points to generate discrete mep and e curves.

Returns

congruence_factor – (n_elm, 1) Congruence factor in each element specified in elm_idx_list and for each input dataset.

Return type

np.ndarray of float

pynibs.congruence.congruence.**cf_curveshift_workhorse_stretch_correction_variance**(*elm_idx_list*, *mep*, *mep_params*, *e*, *n_samples=100*)

Worker function for congruence factor computation - call from `multiprocessing.Pool`. Calculates congruence factor for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements. The computations are parallelized in terms of element indices (`elm_idx_list`). `n_samples` are taken from fitted_mep, within the range of the *Mep*.

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for.
- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions
- **mep_params** (*np.ndarray of float [n_mep_params_total]*) – List of all mep parameters of curve fits used to calculate the MEP (accumulated into one array)
  - e.g. [`mep_#1_para_#1`, `mep_#1_para_#2`, `mep_#1_para_#3`, `mep_#2_para_#1`, `mep_#2_para_#1`, ...]
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of n_datasets of the electric field to compute the congruence factor for, e.g. `(e_mag, e_norm, e_tan)`. Each dataset is a list over all conditions containing the electric field component of interest
  - e.g.: `len(e) = n_cond`
  - `len(e[0]) = n_comp` (e.g: `e_mag = e[0]`))
- **n_samples** (*int, default=100*) – Number of data points to generate discrete mep and e curves

Returns

congruence_factor – Congruence factor in each element specified in elm_idx_list and for each input dataset

Return type

np.ndarray of float [n_roi, n_datasets]

pynibs.congruence.congruence.**cf_variance_sign_workhorse**(*elm_idx_list*, *mep*, *mep_params*, *e*)

Worker function for congruence factor computation - call from `multiprocessing.Pool`. Calculates congruence factor for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements.

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for.
- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions.
- **mep_params** (*np.ndarray of float*) – (n_mep_params_total) List of all mep parameters of curve fits used to calculate the MEP, accumulated into one array), e.g. [`mep_#1_para_#1`, `mep_#1_para_#2`, `mep_#1_para_#3`, `mep_#2_para_#1`, `mep_#2_para_#1`, ...])
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of `n_datasets` of the electric field to compute the congruence factor for, e.g.

(e_mag, e_norm, e_tan). Each dataset is a list over all conditions containing the electric field component of interest

- len(e) = n_cond
- len(e[0]) = n_comp (e.g: e_mag = e[0])

Returns

congruence_factor – (n_roi, n_datasets) Congruence factor in each element specified in elm_idx_list and for each input dataset.

Return type

np.ndarray of float

pynibs.congruence.congruence.**cf_variance_workhorse**(*elm_idx_list*, *mep*, *mep_params*, *e*, *old_style=True*)

Worker function for congruence factor computation - call from multiprocessing.Pool. Calculates congruence factor for e = (E_mag, E_norm and/or E_tan) for given zaps and elements.

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for.
- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions.
- **mep_params** (*np.ndarray of float*) – (n_mep_params_total) List of all mep parameters used to calculate the MEP, accumulated into one array).
  - e.g. [mep_#1_para_#1, mep_#1_para_#2, mep_#1_para_#3, mep_#2_para_#1, mep_#2_para_#1, ...])
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of n_datasets of the electric field to compute the congruence factor for, e.g. (e_mag, e_norm, e_tan). Each dataset is a list over all conditions containing the electric field component of interest
  - len(e) = n_cond
  - len(e[0]) = n_comp (e.g: e_mag = e[0])
- **old_style** (*boolean (default: True)*) – True: Weight var(x_0_prime(r)) with mean(e(r) * mean(Stimulator Intensity), taken from mep False: Weight var(x_0_prime(r)) with mean(E(r)), taken from *e*

Returns

congruence_factor – (n_roi, n_datasets) Congruence factor in each element specified in elm_idx_list and for each input dataset

Return type

np.ndarray of float

pynibs.congruence.congruence.**extract_condition_combination**(*fn_config_cfg*, *fn_results_hdf5*, *conds*, *fn_out_prefix*)

Extract and plot congruence factor results for specific condition combinations from permutation analysis.

Parameters

- **fn_config_cfg** (*str*) – Filename of .cfg file the permutation study was cinducted with
- **fn_results_hdf5** (*str*) – Filename of .hdf5 results file generated by 00_run_c_standard_compute_all_permutations.py containing congruence factors and condition combinations.
- **conds** (*list of str*) – (n_cond) List containing condition combinations to extract and plot, e.g. ['P_0', 'I_225', 'M1_0', 'I_675', 'P_225']).

- **fn_out_prefix** (*str*) – Prefix of output filenames of *_data.xdmf, *_data.hdf5 and *_geo.hdf5.

Returns

- **<fn_out_prefix_data.xdmf>** (*.xdmf file*) – Output file linking *_data.hdf5 and *_geo.hdf5 file to plot in paraview.
- **<fn_out_prefix_data.hdf5>** (*.hdf5 file*) – Output .hdf5 file containing the data.
- **<fn_out_prefix_geo.xdmf>** (*.hdf5 file*) – Output .hdf5 file containing the geometry information.

pynibs.congruence.ext_metrics module

pynibs.congruence.ext_metrics.**dvs_likelihood**(*params*, *x*, *y*, *verbose=True*, *normalize=False*, *bounds=[(1, 2), (1, 2)]*)

pynibs.congruence.ext_metrics.**e_cog_workhorse**(*elm_idx_list*, *mep*, *mep_params*, *e*)

Worker function for electric field center of gravity (e_cog) computation after Opitz et al. (2013) [1] - call from `multiprocessing.Pool`. Calculates the e_cog for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements. The electric field is weighted by the mean MEP amplitude (turning point of the sigmoid) and summed up. The computations are parallelized in terms of element indices (`elm_idx_list`).

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for.
- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions.
- **mep_params** (*np.ndarray of float*) – (n_mep_params_total) List of all mep parameters of curve fits used to calculate the MEP, accumulated into one array.
  - e.g. [`mep_#1_para_#1`, `mep_#1_para_#2`, `mep_#1_para_#3`, `mep_#2_para_#1`, `mep_#2_para_#1`, ...]
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of n_datasets of the electric field to compute the congruence factor for, e.g. `(e_mag, e_norm, e_tan)`. Each dataset is a list over all conditions containing the electric field component of interest
  - e.g.: `len(e) = n_cond`
  - `len(e[0]) = n_comp` (e.g: `e_mag = e[0]`))

Returns

e_cog – (n_roi, n_datasets) RSD inverse in each element specified in `elm_idx_list` and for each input dataset.

Return type

np.ndarray of float

Notes

pynibs.congruence.ext_metrics.**e_focal_workhorse**(*elm_idx_list*, *e*)

Worker function to determine the site of stimulation after Aonuma et al. (2018) [1], call from `multiprocessing.Pool`. Calculates the site of stimulation for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements by multiplying the electric fields with each other. The computations are parallelized in terms of element indices (`elm_idx_list`).

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of `n_datasets` of the electric field to compute the congruence factor for, e.g. `(e_mag, e_norm, e_tan)`. Each dataset is a list over all conditions containing the electric field component of interest
  - `len(e) = n_cond`
  - `len(e[0]) = n_comp` (e.g: `e_mag = e[0]`))

Returns

e_focal – (n_roi, n_datasets) Focal electric field in each element specified in `elm_idx_list` and for each input.

Return type

np.ndarray of float

Notes

pynibs.congruence.ext_metrics.**rsd_inverse_workhorse**(*elm_idx_list*, *mep*, *e*)

Worker function for RSD inverse computation after Bungert et al. (2017) [1], call from `multiprocessing.Pool`. Calculates the RSD inverse for `e = (E_mag, E_norm and/or E_tan)` for given zaps and elements. The computations are parallelized in terms of element indices (`elm_idx_list`).

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for
- **mep** (list of *Mep*) – (n_cond) List of fitted Mep object instances for all conditions.
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of `n_datasets` of the electric field to compute the congruence factor for, e.g. `(e_mag, e_norm, e_tan)`. Each dataset is a list over all conditions containing the electric field component of interest
  - `len(e) = n_cond`
  - `len(e[0]) = n_comp` (e.g: `e_mag = e[0]`))

Returns

rsd_inv – (n_roi, n_datasets) RSD inverse in each element specified in `elm_idx_list` and for each input dataset.

Return type

np.ndarray of float

Notes

pynibs.congruence.stimulation_threshold module

pynibs.congruence.stimulation_threshold.**intensity_thresh**(*mep_curve*, *intensities*, *mep_threshold*)

Determines the stimulation threshold of one particular condition (usually the most sensitive e.g. M1-45). The stimulation threshold is the stimulator intensity value in [A/us] where the mep curves exceed the value of mep_threshold (in [mV]).

Parameters

- **mep_curve** (*list [1] of np.ndarray of float [n_samples]*) – MEP curve values for every conditions
- **intensities** (*list [1] of np.ndarray of float [n_samples]*) – To the MEP values corresponding stimulator intensities in [A/us]
- **mep_threshold** (*float*) – MEP value in [mV], which has to be exceeded for threshold definition

Returns

stim_threshold_avg – Average stimulation threshold in [V/m] where c_factor is greater than c_factor_percentile

Return type

float

pynibs.congruence.stimulation_threshold.**mean_mep_threshold**(*elm_idx*, *mep_curve*, *intensities*, *e*, *mep_threshold*)

Determines the stimulation threshold by calculating the average electric field over all conditions, where the mep curves exceed the value of mep_threshold (in [mV]).

Parameters

- **elm_idx** (*list of np.ndarray of int*) – [n_datasets](n_elements) Element indices where the congruence factor exceeds a certain percentile, defined during the call of *stimulation_threshold()*.
- **mep_curve** (*list of np.ndarray of float*) – [n_conditions](n_samples) MEP curve values for every condition.
- **intensities** (*list of np.ndarray of float*) – [n_conditions](n_samples) To the MEP values corresponding stimulator intensities in [A/us].
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of n_datasets of the electric field to compute the congruence factor for, e.g. `(e_mag, e_norm, e_tan)`. Each dataset is a list over all conditions containing the electric field component of interest
  - e.g.: `len(e) = n_cond`
  - `len(e[0]) = n_comp` (e.g: `e_mag = e[0]`)
- **mep_threshold** (*float*) – MEP value in [mV], which has to be exceeded for threshold definition.

Returns

stim_threshold_avg – Average stimulation threshold in [V/m] where c_factor is greater than c_factor_percentile

Return type

float

pynibs.congruence.stimulation_threshold.**sigmoid_thresh**(*elm_idx*, *mep_curve*, *intensities*, *e*, *mep_threshold*)

Determines the stimulation threshold by calculating an equivalent *pynibs.expio.Mep.sigmoid* over all conditions. The stimulation threshold is the electric field value where the mep curves exceed the value of mep_threshold (in [mV]).

Parameters

- **elm_idx** (*list of np.ndarray of int*) – [n_datasets](n_elements) Element indices where the congruence factor exceeds a certain percentile, defined during the call of *stimulation_threshold()*.
- **mep_curve** (*list of np.ndarray of float*) – [n_conditions](n_samples) MEP curve values for every condition.
- **intensities** (*list of np.ndarray of float*) – [n_conditions](n_samples) To the MEP values corresponding stimulator intensities in [A/us].
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of n_datasets of the electric field to compute the congruence factor for, e.g. `(e_mag, e_norm, e_tan)`. Each dataset is a list over all conditions containing the electric field component of interest
  - e.g.: `len(e) = n_cond`
  - `len(e[0]) = n_comp` (e.g: `e_mag = e[0]`)
- **mep_threshold** (*float*) – MEP value in [mV], which has to be exceeded for threshold definition

Returns

stim_threshold_avg – Average stimulation threshold in [V/m] where c_factor is greater than c_factor_percentile

Return type

float

pynibs.congruence.stimulation_threshold.**stimulation_threshold**(*elm_idx_list*, *mep*, *mep_params*, *n_samples*, *e*, *c_factor_percentile=95*, *mep_threshold=0.5*, *c_factor=None*, *c_function=None*, *t_function=None*)

Computes the stimulation threshold in terms of the electric field in [V/m]. The threshold is defined as the electric field value where the mep exceeds mep_threshold. The average value is taken over all mep curves in each condition and over an area where the congruence factor exceeds `c_factor_percentile`.

Parameters

- **elm_idx_list** (*np.ndarray*) – (chunksize) List of element indices, the congruence factor is computed for.
- **mep** (*list of Mep object instances*) – (n_cond) List of fitted *Mep* object instances for all conditions.
- **mep_params** (*np.ndarray of float [n_mep_params_total]*) – List of all mep parameters of curve fits used to calculate the MEP (accumulated into one array)
  - e.g. [`mep_#1_para_#1`, `mep_#1_para_#2`, `mep_#1_para_#3`, `mep_#2_para_#1`, `mep_#2_para_#1`, ...]
- **n_samples** (*int*) – Number of data points to generate discrete mep and e curves.
- **e** (*list of list of np.ndarray of float*) – [n_cond][n_datasets][n_elm] Tuple of `n_datasets` of the electric field to compute the congruence factor for, e.g.

(e_mag, e_norm, e_tan). Each dataset is a list over all conditions containing the electric field component of interest

- e.g.: len(e) = n_cond
- len(e[0]) = n_comp (e.g: e_mag = e[0])

- **c_factor_percentile** (*float*) – Percentile of the c_factor taken into account for the threshold evaluation. Only c_factors are considered exceeding this.
- **mep_threshold** (*float*) – MEP value in [mV], which has to be exceeded for threshold definition.
- **c_factor** (*np.ndarray of float*) –
- **(n_roi** –
- **dataset.** *(n_datasets) Congruence factor in each element specified in elm_idx_list and for each input*) –
- **c_function** (*function*) – Defines the function to use during c_gpc to calculate the congruence factor.
  - congruence_factor_curveshift_workhorse: determines the average curve shift
  - congruence_factor_curveshift_workhorse_stretch_correction: determines the average curve shift
  - congruence_factor_curveshift_workhorse_stretch_correction_variance: determines the average curve shift
  - congruence_factor_variance_workhorse: evaluates the variance of the shifting and stretching parameters
- **t_function** (*function*) – Defines the function to determine the stimulation_threshold.
  - stimulation_threshold_mean_mep_threshold: uses mep_threshold to determine the corresponding e_threshold over all conditions and takes the average values as the stimulation threshold
  - stimulation_threshold_pynibs.sigmoid: Fits a new pynibs.sigmoid using all datapoints in the mep-vs-E space and evaluates the threshold from the turning point or the intersection of the derivative in the crossing point with the e-axis

Returns

stim_threshold_avg – Average stimulation threshold in [V/m] where c_factor is greater than c_factor_percentile.

Return type

float

pynibs.expio package

Submodules

pynibs.expio.Mep module

class pynibs.expio.Mep.**Mep**(*intensities*, *mep*, *intensity_min_threshold=None*, *mep_min_threshold=None*)

Bases: object

Mep object.

fun

Function type to fit data with (pynibs.sigmoid / pynibs.exp / pynibs.linear).

Type
function

fun_sig

Best fitting equivalent sigmoidal function (added by `self.add_sigmoidal_bestfit()`).

Type
function

popt

(N_para), Fitted optimal function parameters.

Type
np.ndarray of float

popt_sig

(3), Best fitting parameters `x0`, `r`, and `amp` of equivalent sigmoidal function.

Type
np.ndarray of float

copt

(N_para, N_para), covariance matrix of fitted parameters.

Type
np.ndarray of float

pstd

(N_para), Standard deviation of fitted parameters.

Type
np.ndarray of float

fit

Gmodel object instance of parameter fit.

Type
object instance

x_limits

(2), Minimal and maximal value of intensity data.

Type
np.ndarray of float

y_limits

(2), Minimal and maximal value of mep data.

Type
np.ndarray of float

mt

Motor threshold (MEP > 50 uV), evaluated from fitted curve, added after fitting.

Type
float

calc_motor_threshold(*threshold*)

Determine motor threshold of stimulator depending on MEP threshold given in [mV].

Parameters
threshold (*float*) – Threshold of MEP amplitude in [mV].

Notes

Add Attributes:

Mep.mt
[float] Motor threshold for given MEP threshold.

`eval`(*x*, *p*)

Evaluating fitted function with optimal parameters in points x.

Parameters

- **x** (*np.ndarray of float*) – (N_x) Function arguments.
- **p** (*tuple of float*) – Function parameters.

Returns
y – (N_x) Function values.

Return type
np.ndarray of float

`eval_fun_sig`(*x*, *p*)

Evaluating optimally fitted sigmoidal function with optimal parameters in points x.

Parameters

- **x** (*np.ndarray of float*) – (N_x) Function arguments.
- **p** (*tuple of float*) – Function parameters

Returns
y – (N_x) Function values.

Return type
np.ndarray of float

`eval_opt`(*x*)

Evaluating fitted function with optimal parameters in points x.

Parameters
x (*np.ndarray of float*) – (N_x) Function arguments.

Returns
y – (N_x) Function values.

Return type
np.ndarray of float

`eval_uncertainties`(*x*, *sigma=1*)

Evaluating approximated uncertainty interval around fitted distribution.

Parameters

- **x** (*np.ndarray of float*) – (N_x) Function values where uncertainty is evaluated.
- **sigma** (*float*) – Standard deviation of parameters taken into account when evaluating uncertainty interval.

Returns

- **y_min** (*np.ndarray of float*) – (N_x) Lower bounds of y-values.
- **y_max** (*np.ndarray of float*) – (N_x) Upper bounds of y-values.

`fit_mep_to_function`(*p0=None*)

Fits MEP data to function. The algorithm tries to fit the function first to a sigmoid, then to an exponential and finally to a linear function.

Parameters
p0 (*np.ndarray of float*) – Initial guess of parameter values.

Notes

Add Attributes:

Mep.popt
[np.ndarray of float] (N_para) Fitted optimal function parameters.

Mep.copt
[np.ndarray of float] (N_para, N_para) Covariance matrix of fitted parameters.

Mep.pstd
[np.ndarray of float] (N_para) Standard deviation of fitted parameters.

Mep.fun
[function] Function mep data was fitted with.

Mep.fit
[object instance] Gmodel object instance of parameter fit.

Mep.mt
[float] Motor threshold (MEP > 50 uV).

`fit_mep_to_function_multistart`(*p0=None*, *constraints=None*, *fun=None*)

Fits MEP data to function.

Parameters

- **p0** (*np.ndarray of float*) – Initial guess of parameter values.
- **constraints** (*dict*) – Dictionary with parameter names as keys and [min, max] values as constraints.
- **fun** (*list of functions*) – Functions to incorporate in the multistart fit (e.g. [`pynibs.sigmoid`, `pynibs.exp0`, `pynibs.linear`]).

Notes

Add Attributes:

Mep.popt
[np.ndarray of float] (N_para) Best fitted optimal function parameters.

Mep.copt
[np.ndarray of float] (N_para, N_para) Covariance matrix of best fitted parameters.

Mep.pstd
[np.ndarray of float] (N_para) Standard deviation of best fitted parameters.

Mep.fun
[function] Function of best fit mep data was fitted with.

Mep.fit
[list of fit object instances] Gmodel object instances of parameter fits.

Mep.best_fit_idx
[int] Index of best function fit (`fit[best_fit_idx]`).

Mep.constraints
[dict] Dictionary with parameter names as keys and [min, max] values as constraints.

plot(*label*, *sigma=3*, *plot_samples=True*, *show_plot=False*, *fname_plot=''*, *ylim=None*, *ylabel=None*, *fontsize_axis=10*, *fontsize_legend=10*, *fontsize_label=10*, *fontsize_title=10*, *fun=None*)

Plotting mep data and fitted curve together with uncertainties. If ``fun == None`, the optimal function is plotted.

Parameters

- **label** (*str*) – Plot title.
- **sigma** (*float*) – Factor of standard deviations the uncertainty of the fit is plotted with.
- **plot_samples** (*boolean*) – Plot sampling curves of the fit in the uncertainty interval.
- **show_plot** (*boolean*) – Show or hide plot window (TRUE / FALSE).
- **fname_plot** (*str*) – Filename of plot showing fitted data (with .png extension).
- **ylim** (*list of float [2]*) – Min and max values of y-axis.
- **fontsize_axis** (*int*) – Fontsize of axis numbers.
- **fontsize_legend** (*int*) – Fontsize of Legend labels.
- **fontsize_label** (*int*) – Fontsize of axis labels.
- **fontsize_title** (*int*) – Fontsize of title.
- **fun** (str or None (None, sigmoid, exp, linear)) – Which function to plot.

Returns
<File> – Plot of Mep data and fit (format: png).

Return type
.png file

run_fit_multistart(*fun*, *x*, *y*, *p0*, *constraints=None*, *verbose=False*, *n_multistart=100*)

Run multistart approach to fit data to function. n_multistart optimizations are performed based on random variations of the initial guess parameters p0. The fit with the lowest AIC (Akaike information criterion), i.e. best fit is returned as gmodel fit instance.

Parameters

- **fun** (*function*) – Function mep data has to be fitted with
- **x** (*np.ndarray of float*) – (N_data), Independent variable the data is fitted on.
- **y** (*np.ndarray of float*) – (N_data), Dependent data the curve is fitted through.
- **p0** (*np.ndarray of float or list of float*) – Initial guess of parameter values.
- **constraints** (*dict, optional*) – Dictionary with parameter names as keys and [min, max] values as constraints.
- **verbose** (*bool, default: False*) – Show output messages.
- **n_multistart** (*int*) – Number of repeated optimizations with different starting points to perform.

Returns
fit – Gmodel object instance of best parameter fit with lowest parameter variance.

Return type
object instance

pynibs.expio.Mep.**butter_lowpass**(*cutoff*, *fs*, *order=5*)

Setup Butter low-pass filter and return filter parameters.

Parameters

- **cutoff** (*float*) – Cutoff frequency in [Hz].

- **fs** (*float*) – Sampling frequency in [Hz].
- **order** (*int, default: 5*) – Filter order.

Returns
b, a – Numerator (b) and denominator (a) polynomials of the IIR filter.

Return type
np.ndarray, np.ndarray

pynibs.expio.Mep.**butter_lowpass_filter**(*data*, *cutoff*, *fs*, *order=5*)

Applies Butterworth lowpass filter

Parameters

- **data** (*np.ndarray of float [N_samples]*) – Input of the digital filter
- **cutoff** (*float*) – Cutoff frequency in [Hz]
- **fs** (*float*) – Sampling frequency in [Hz]
- **order** (*int*) – Filter order

Returns
y – Output of the digital filter

Return type
np.ndarray [N_samples]

pynibs.expio.Mep.**calc_p2p**(*sweep*, *tms_pulse_time=0.2*, *start_mep=20*, *end_mep=35*, *measurement_start_time=0*, *sampling_rate=4000*, *cutoff_freq=500*, *fn_plot=None*)

Calc peak-to-peak values of and mep sweep.

Parameters

- **sweep** (*np.ndarray of float [Nx1]*) – Input curve
- **tms_pulse_time** (*float (Default: .2)*) – onset time of TMS pulse trigger in [s]
- **start_mep** (*int (Default: 18)*) – start of p2p search window after TMS pulse. In [ms].
- **end_mep** (*int (Default: 35)*) – end of p2p search window after TMS pulse. In [ms].
- **measurement_start_time** (*float (Default: 0)*) – start time of the EMG measurement. In [ms].
- **sampling_rate** (*int (Default: 2000)*) – Sampling rate in Hz
- **cutoff_freq** (*int (Default: 500)*) – Desired cutoff frequency of the filter, Hz
- **fn_plot** (*str, default: None*) – Filename of sweep to plot (.png). If None, plot is omitted.

Returns

- **p2p** (*float*) – Peak-to-peak value of input curve
- **sweep_filt** (*np.ndarray of float*) – Filtered input curve (Butter lowpass filter with specified cutoff_freq)
- **onset** (*float*) – MEP onset after tms_pulse_time

pynibs.expio.Mep.**calc_p2p_old_exp0**(*sweep*, *start_mep=None*, *end_mep=None*, *tms_pulse_time=None*, *sampling_rate=None*)

Calc peak-to-peak values of an mep sweep. This version was probably used in the ancient times of experiment 0.

Parameters

- **sweep** (*np.ndarray of float*) – (Nx1) Input curve.
- **start_mep** (*None*) – Not used.
- **end_mep** (*None*) – Not used.
- **tms_pulse_time** (*None*) – Not used.
- **sampling_rate** (*int (Default: 2000)*) – Sampling rate in Hz.

Returns
p2p – Peak-to-peak value of input curve

Return type
float

pynibs.expio.Mep.**calc_p2p_old_exp1**(*sweep*, *start_mep=18*, *end_mep=35*, *tms_pulse_time=None*, *sampling_rate=2000*)

Calc peak-to-peak values of an mep sweep. This version was probably used for the first fits of experiment 1.

Parameters

- **sweep** (*np.ndarray of float [Nx1]*) – Input curve

  start_mep: Int or Float (Default: 18)
  Starttime in [ms] after TMS for MEP seach window.

  end_mep: Int or Float (Default: 35)
  Endtime in [ms] after TMS for MEP seach window.
- **tms_pulse_time** (*None*) – Not used.
- **sampling_rate** (*int (Default: 2000)*) – Sampling rate in Hz

Returns
p2p – Peak-to-peak value of input curve

Return type
float

pynibs.expio.Mep.**dummy_fun**(*x*, *a*)

Dummy function for congruence factor calculation.

pynibs.expio.Mep.**exp**(*x*, *x0*, *r*, *y0*)

Parametrized exponential function.

$$y = y_0 + e^{r(x-x_0)}$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) X-values the function is evaluated in.
- **x0** (*float*) – Horizontal shift along the abscissa.
- **r** (*float*) – Slope parameter.
- **y0** (*float*) – Offset parameter.

Returns
y – Function value at x

Return type
np.ndarray of float [N_x]

pynibs.expio.Mep.**exp0**(*x*, *x0*, *r*)

Parametrized exponential function w/o offset.

$$y = e^{r(x-x_0)}$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) X-values the function is evaluated in.
- **x0** (*float*) – Horizontal shift along the abscissa.
- **r** (*float*) – Slope parameter.

Returns
y – (N_x) Function value at x.

Return type
np.ndarray of float

pynibs.expio.Mep.**exp0_log**(*x*, *x0*, *r*)

Parametrized exponential function w/o offset.

$$y = e^{r(x-x_0)}$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) X-values the function is evaluated in.
- **x0** (*float*) – Horizontal shift along the abscissa.
- **r** (*float*) – Slope parameter.

Returns
y – (N_x) Function value at x.

Return type
np.ndarray of float

pynibs.expio.Mep.**exp_log**(*x*, *x0*, *r*, *y0*)

Parametrized exponential function (log)

$$y = y_0 + e^{r(x-x_0)}$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) X-values the function is evaluated in.
- **x0** (*float*) – Horizontal shift along the abscissa.
- **r** (*float*) – Slope parameter.
- **y0** (*float*) – y-offset parameter.

Returns
y – (N_x) Function value at x.

Return type
np.ndarray of float

pynibs.expio.Mep.**get_mep_elements**(*mep_fn*, *tms_pulse_time*, *drop_mep_idx=None*, *cfs_data_column=0*, *channels=None*, *time_format='delta'*, *plot=False*, *start_mep=18*, *end_mep=35*)

Read EMG data from CED .cfs or .txt file and returns MEP amplitudes.

Parameters

- **mep_fn** (*string*) – path to .cfs-file or .txt file (Signal export).
- **tms_pulse_time** (*float*) – Time in [s] of TMS pulse as specified in signal.
- **drop_mep_idx** (*List of int or None, default: None*) – Which MEPs to remove before matching.
- **cfs_data_column** (*int or list of int*) – Column(s) of dataset in cfs file. +1 for .txt.
- **channels** (*list of str, default: None*) – Channel names.

- **time_format** (*str, default:"delta"*) – Format of mep time stamps in time_mep_lst to return.
  - "delta" returns list of datetime.timedelta in seconds
  - "hms" returns datetime.datetime(year, month, day, hour, minute, second, microsecond)
- **plot** (*bool, default: False*) – Plot MEPs.
- **start_mep** (*float, default: 18*) – Start of time frame after TMS pulse where p2p value is evaluated (in ms).
- **end_mep** (*float, default: 35*) – End of time frame after TMS pulse where p2p value is evaluated (in ms).

Returns

- **p2p_array** (*np.ndarray of float*) – (N_stim) Peak to peak values of N sweeps.
- **time_mep_lst** (*list of datetime.timedelta*) – MEP-timestamps
- **mep_raw_data** (*np.ndarray of float*) – (N_channel, N_stim, N_samples) Raw (unfiltered) MEP data.
- **mep_filt_data** (*np.ndarray of float*) – (N_channel, N_stim, N_samples) Filtered MEP data (Butterworth lowpass filter).
- **time** (*np.ndarray of float*) – (N_samples) Time axis corresponding to MEP data.
- **mep_onset_array** (*np.ndarray of float*) – (S_samples) MEP onset after TMS pulse.

pynibs.expio.Mep.**get_mep_sampling_rate**(*cfs_path*)

Returns sampling rate [Hz] for CED Signal EMG data in .cfs, .mat or .txt file.

The sampling rate is saved in the cfs header like this: .. code-block:: sh

``Samplingrate" : 3999.999810,

``

Parameters

cfs_path
[str] Path to .cfs file or .txt file.

Returns

float
[sampling rate]

pynibs.expio.Mep.**get_mep_virtual_subject_DVS**(*x*, *x0=0.5*, *r=10*, *amp=1*, *sigma_x=0*, *sigma_y=0*, *y0=0.01*, *seed=None*)

Creates random mep data using the 2 variability source model from Goetz et al. 2014 [1] together with a standard 3 parametric sigmoid function.

References

Parameters

- **x** (*np.ndarray of float*) – (n_x) Normalized stimulator intensities [0 ... 1].
- **x0** (*float, default: 0.5*) – Location of turning point sigmoidal function.
- **r** (*float, default: 0.25*) – Steepness of sigmoidal function.
- **amp** (*float, default: 1.0*) – Saturation amplitude of sigmoidal function.

- **sigma_x** (*float, default: 0.1*) – Standard deviation of additive x variability source.
- **sigma_y** (*float, default: 0.1*) – Standard deviation of additive y variability source.
- **y0** (*float, default: 1e-2*) – y-offset.
- **seed** (*int, default: None*) – Seed to use.

Returns
mep – (n_x) Motor evoked potential values

Return type
np.ndarray of float

pynibs.expio.Mep.**get_mep_virtual_subject_TVS**(*x*, *p1=-5.0818*, *p2=-2.4677*, *p3=3.6466*, *p4=0.42639*, *p5=1.6665*, *mu_y_add=8.283235329759169e-06*, *mu_y_mult=-0.9645334*, *mu_x_add=0.68827324*, *sigma_y_add=1.4739e-06*, *k=0.39316*, *sigma2_y_mult=0.022759*, *sigma2_x_add=0.023671*, *subject_variability=False*, *trial_variability=True*)

Creates random mep data using the 3 variability source model from [1]. There are typos in the paper but the code seems to be correct. Originally from S. Goetz: https://github.com/sgoetzduke/Statistical-MEP-Model . Rewritten from MATLAB to Python by Konstantin Weise.

Parameters

- **x** (*np.ndarray of float*) – (n_x) Normalized stimulator intensities [0 ... 1],
- **p1** (*float, default: -5.0818*) – First parameter of sigmoidal hilltype function.
- **p2** (*float, default: 4.5323*) – Second parameter of sigmoidal hilltype function.
- **p3** (*float, default: 3.6466*) – Third parameter of sigmoidal hilltype function.
- **p4** (*float, default: 0.42639*) – Fourth parameter of sigmoidal hilltype function.
- **p5** (*float, default: 1.6665*) – Fifth parameter of sigmoidal hilltype function.
- **mu_y_add** (*float, default: 10**-5.0818*) – Mean value of additive y variability source.
- **mu_y_mult** (*float, default: -0.9645334*) – Mean value of multiplicative y variability source.
- **mu_x_add** (*float, default: -0.68827324*) – Mean value of additive x variability source.
- **sigma_y_add** (*float, default: 1.4739*1e-6*) – Standard deviation of additive y variability source.
- **k** (*float, default: 0.39316*) – Shape parameter of generalized extreme value distribution.
- **sigma2_y_mult** (*float, default: 2.2759*1e-2*) – Standard deviation of multiplicative y variability source.
- **sigma2_x_add** (*float, default: 2.3671*1e-2*) – Standard deviation of additive x variability source.
- **subject_variability** (*bool, default: False*) – Choose if shape parameters from IO curve are sampled from random distributions to model subject variability. This does not influence the trial-to-trial variability.
- **trial_variability** (*bool, default: True*) – Enable or disable trial-to-trial variability. Disabling it will result in ideal IO curves w/o noise.

Returns
mep – (n_x) Motor evoked potential values in mV

Return type
np.ndarray of float

References

pynibs.expio.Mep.**get_time_date**(*cfs_paths*)
Get time and date of the start of the recording out of .cfs file.

Parameters
cfs_paths (*str*) – Path to .cfs mep file.

Returns
time_date – Date an time.

Return type
str

pynibs.expio.Mep.**linear**(*x*, *m*, *n*)
Parametrized linear function.

$$y = mx + n$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) X-values the function is evaluated in.
- **m** (*float*) – Slope parameter.
- **n** (*float*) – y-offset.

Returns
y – (N_x) Function value at argument x.

Return type
np.ndarray of float

pynibs.expio.Mep.**linear_log**(*x*, *m*, *n*)
Parametrized log linear function

$$y = mx + n$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) X-values the function is evaluated in.
- **m** (*float*) – Slope.
- **n** (*float*) – Y-offset.

Returns
y – (N_x) Function value at x.

Return type
np.ndarray of float

pynibs.expio.Mep.**read_biosig_emg_data**(*fn_data*, *include_first_trigger=False*, *type='cfs'*)

pynibs.expio.Mep.**sigmoid**(*x*, *x0*, *r*, *amp*)
Parametrized sigmoid function.

$$y = \frac{amp}{1 + e^{-r(x - x_0)}}$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) X-values the function is evaluated in.
- **x0** (*float*) – Horizontal shift along the abscissa.
- **r** (*float*) – Slope parameter (steepness).
- **amp** (*float*) – Maximum value the sigmoid converges to.

Returns
y – (N_x) Function value at argument x

Return type
np.ndarray of float

pynibs.expio.Mep.**sigmoid4**(*x*, *x0*, *r*, *amp*, *y0*)

Parametrized sigmoid function with 4 parameters.

$$y = y_0 + \frac{amp - y_0}{1 + e^{-r(x - x_0)}}$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) x-values the function is evaluated in.
- **x0** (*float*) – Horizontal shift along the abscissa.
- **r** (*float*) – Slope parameter (steepness).
- **amp** (*float*) – Maximum value the sigmoid converges to.
- **y0** (*float*) – Offset value of the sigmoid.

Returns
y – (N_x) Function value at argument x.

Return type
np.ndarray of float

pynibs.expio.Mep.**sigmoid4_log**(*x*, *x0*, *r*, *amp*, *y0*)

Parametrized log transformed sigmoid function with 4 parameters.

$$y = \log\left(y_0 + \frac{amp - y_0}{1 + e^{-r(x - x_0)}}\right)$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) X-values the function is evaluated in.
- **x0** (*float*) – Horizontal shift along the abscissa.
- **r** (*float*) – Slope parameter (steepness).
- **amp** (*float*) – Maximum value the sigmoid converges to (upper saturation).
- **y0** (*float*) – Y-offset value of the sigmoid.

Returns
y – (N_x) Function value at argument x.

Return type
np.ndarray of float

pynibs.expio.Mep.**sigmoid_log**(*x*, *x0*, *r*, *amp*)

Parametrized log transformed sigmoid function.

$$y = \log\left(\frac{amp}{1 + e^{-r(x - x_0)}}\right)$$

Parameters

- **x** (*np.ndarray of float*) – (N_x) x-values the function is evaluated in.

- **x0** (*float*) – Horizontal shift along the abscissa.
- **r** (*float*) – Slope parameter (steepness).
- **amp** (*float*) – Maximum value the sigmoid converges to.

Returns
y – (N_x) Function value at argument x

Return type
np.ndarray of float

pynibs.expio.brainsight module

class pynibs.expio.brainsight.**BrainsightCSVParser**

Bases: object

class State

Bases: object

DATA = 3

DATA_HEADER = 2

FILE_HEADER = 0

FIN = 4

SKIPPING = 1

check_state_transition(*line*)

parse_line(*line*)

pynibs.expio.brainsight.**create_merged_nnav_emg_file_brainsight**(*fn_brainsight_nnav_info*, *fn_emg_info*, *fn_out*, *p2p_window_start=18*, *p2p_window_end=35*)

pynibs.expio.brainsight.**merge_exp_data_brainsight**(*subject*, *exp_idx*, *mesh_idx*, *coil_outlier_corr_cond=False*, *remove_coil_skin_distance_outlier=True*, *coil_distance_corr=True*, *verbose=False*, *plot=False*, *start_mep=18*, *end_mep=35*)

Merge the TMS coil positions and the mep data into an experiment.hdf5 file.

Parameters

- **subject** (*pynibs.Subject*) – Subject object
- **exp_idx** (*str*) – Experiment ID
- **mesh_idx** (*str*) – Mesh ID
- **coil_outlier_corr_cond** (*bool*) – Correct outlier of coil position and orientation (+-2 mm, +-3 deg) in case of conditions
- **remove_coil_skin_distance_outlier** (*bool*) – Remove outlier of coil position lying too far away from the skin surface (+- 5 mm)
- **coil_distance_corr** (*bool*) – Perform coil <-> head distance correction (coil is moved towards head surface until coil touches scalp)
- **verbose** (*bool*) – Plot output messages
- **plot** (*bool, optional, default: False*) – Plot MEPs and p2p evaluation

- **start_mep** (*float, optional, default: 18*) – Start of time frame after TMS pulse where p2p value is evaluated (in ms)
- **end_mep** (*float, optional, default: 35*) – End of time frame after TMS pulse where p2p value is evaluated (in ms)

pynibs.expio.brainsight.**read_targets_brainsight**(*fn*)

Reads target coil position and orientations from .txt file and returns it as 4 x 4 x N_targets numpy array.

Parameters
fn (*str*) – Filename of output file.

Returns
m_nnav – Tensor containing the 4x4 matrices with coil orientation and position.

Return type
ndarray of float [4 x 4 x N_targets]

pynibs.expio.brainsight.**write_targets_brainsight**(*targets*, *fn_out*, *names=None*, *overwrite=True*)

Writes coil position and orientations in .txt file for import into Brainsight.

Parameters

- **targets** (*ndarray of float [4 x 4 x N_targets]*) – Tensor containing the 4x4 matrices with coil orientation and position
- **fn_out** (*str*) – Filename of output file
- **names** (*list of str, optional, default: None*) – Target names (If nothing is provided they will be numbered by their order)
- **overwrite** (*bool, optional, default: True*) – Overwrite existing .txt file

Return type
<file> .txt file containing the targets for import into Brainsight

pynibs.expio.brainvis module

pynibs.expio.brainvis.**read_channel_names**(*fname*)

Reads the channel names from .vhdr (info) file, which is recorded during EEG.

Parameters
fname (*str*) – Filename of .vhdr info file.

Returns
channel_names – List containing the channel names.

Return type
list of str

pynibs.expio.brainvis.**read_sampling_frequency**(*fname*)

Reads the sampling frequency from .vhdr (info) file, which is recorded during EEG.

Parameters
fname (*str*) – Filename of .vhdr info file.

Returns
sampling_frequency – Sampling frequency.

Return type
float

pynibs.expio.cobot module

pynibs.expio.cobot.**merge_exp_data_cobot**(*subject*, *exp_idx*, *mesh_idx*, *coil_outlier_corr_cond=False*, *remove_coil_skin_distance_outlier=True*, *coil_distance_corr=True*, *verbose=False*, *plot=False*)

Merge the TMS coil positions and the mep data into an experiment.hdf5 file.

Parameters

- **subject** (*pyfempp.subject*) – Subject object
- **exp_idx** (*str*) – Experiment ID
- **mesh_idx** (*str*) – Mesh ID
- **coil_outlier_corr_cond** (*bool*) – Correct outlier of coil position and orientation (+-2 mm, +-3 deg) in case of conditions
- **remove_coil_skin_distance_outlier** (*bool*) – Remove outlier of coil position lying too far away from the skin surface (+- 5 mm)
- **coil_distance_corr** (*bool*) – Perform coil <-> head distance correction (coil is moved towards head surface until coil touches scalp)
- **verbose** (*bool*) – Plot output messages
- **plot** (*bool, optional, default: False*) – Plot MEPs and p2p evaluation

pynibs.expio.exp module

pynibs.expio.exp.**add_sigmoidal_bestfit**(*mep*, *p0*, *constraints=None*)

Add best fitting sigmoidal function to instance (determined by multistart approach)

Parameters

- **mep** (*pynibs.Mep*) – Mep object class instance
- **p0** (*float*) –
- **constraints** (*dict*) – Dictionary with parameter names as keys and [min, max] values as constraints

Returns

mep – Updated Mep object class instance with the following attributes

Return type

object

Notes

Adds Attributes

Mep.fun_sig
[function] Sigmoid function

Mep.popt_sig
[np.ndarray of float [3]] Parameters of sigmoid function

pynibs.expio.exp.**calc_outlier**(*coords*, *dev_location*, *dev_radius*, *target=None*, *fn_out=None*, *print_msg=True*)

Computes median coil position and angle, identifies outliers, plots neat figure. Returns a list of idx that are not outliers

Parameters

- **coords** (*4 x 4 x n_zaps np.ndArray*) –
- **dev_location** (*float*) –
- **dev_radis** (*flat*) –
- **target** (*np.ndarray*) – 4*4 matrix with target coordinates. Optional.
- **fn_out** (*string, optional*) –

Returns
list of int, list of int, list of int

Return type
idx_keep, idx_zero, idx_outlier

pynibs.expio.exp.**cfs2hdf5**(*fn_cfs*, *fn_hdf5=None*)

Converts EMG data included in .cfs file to .hdf5 format.

Parameters

- **fn_cfs** (*str*) – Filename of .cfs file
- **fn_hdf5** (*str, optional, default: None*) – Filename of .hdf5 file (if not provided, a file with same name as fn_cfs will be created with .hdf5 extension)

Returns
<file> – File containing: - EMG data in f["emg"][:] - Time axis in f["time"][:]

Return type
.hdf5 File

pynibs.expio.exp.**coil_distance_correction**(*exp=None*, *fn_exp=None*, *fn_exp_out=None*, *fn_geo_hdf5=None*, *remove_coil_skin_distance_outlier=False*, *fn_plot=None*, *min_dist=-5*, *max_dist=2*)

Corrects the distance between the coil and the head assuming that the coil is touching the head surface during the experiments. This is done since the different coil tracker result in different coil head distances due to tracking inaccuracies. Also averages positions and orientations over the respective condition and writes both mean position and orientation for every condition in fn_exp_out.

Depending on if exp (dict containing lists) or fn_exp (csv file) is provided it returns the outlier corrected dict or writes a new <fn_exp_out>.csv file.

Parameters

- **exp** (*list of dict or dict of list, optional, default: None*) – List of dictionaries containing the experimental data
- **fn_exp** (*str*) – Filename (incl. path) of experimental .csv file
- **fn_exp_out** (*str*) – Filename (incl. path) of distance corrected experimental .csv file
- **fn_geo_hdf5** (*str*) – Filename (incl. path) of geometry mesh file (.hdf5)
- **remove_coil_skin_distance_outlier** (*bool*) – Remove coil positions, which are more than +- 2 mm located from the zero mean skin surface.
- **fn_plot** (*str, default: None (fn_geo_hdf5 folder)*) – Folder where plots will be saved in.
- **min_dist** (*int*) – Ignored.
- **max_dist** (*int*) – Ignored.

Returns

- **<File>** (*.csv file*) – experimental_dc.csv file with distance corrected coil positions
- *or*

- **exp** (*dict*) – Dictionary containing the outlier corrected experimental data

pynibs.expio.exp.**coil_distance_correction_matsimnibs**(*matsimnibs*, *fn_mesh_hdf5*, *distance=0*, *remove_coil_skin_distance_outlier=False*)

Corrects the distance between the coil and the head assuming that the coil is located at a distance "d" with respect to the head surface during the experiments. This is done since the different coil tracker result in different coil head distances due to tracking inaccuracies.

Parameters

- **matsimnibs** (*ndarray of float [4 x 4] or [4 x 4 x n_mat]*) – Tensor containing matsimnibs matrices
- **fn_mesh_hdf5** (*str*) – .hdf5 file containing the head mesh
- **distance** (*float*) – Target distance in (mm) between coil and head due to hair layer. All coil positions are moved to this distance.
- **remove_coil_skin_distance_outlier** (*bool*) – Remove coil positions, which are more than +- 6 mm located from the skin surface.

Returns

matsimnibs – Tensor containing matsimnibs matrices with distance corrected coil positions

Return type

ndarray of float [4 x 4 x n_mat]

pynibs.expio.exp.**coil_outlier_correction_cond**(*exp=None*, *fn_exp=None*, *fn_exp_out=None*, *outlier_angle=5.0*, *outlier_loc=3.0*)

Searches and removes outliers of coil orientation and location w.r.t. the average orientation and location from all zaps. It generates plots of the individual conditions showing the outliers in the folder of fn_exp_out. Depending on if exp (dict containing lists) or fn_exp (csv file) is provided it returns the outlier corrected dict or writes a new <fn_exp_out>.csv file. If _exp_ is provided, all keys are kept.

Parameters

- **exp** (*list of dict, optional, default: None*) – List of dictionaries containing the experimental data
- **fn_exp** (*str, optional, default: None*) – Filename (incl. path) of experimental .csv file
- **fn_exp_out** (*str, optional, default: None*) – Filename (incl. path) of corrected experimental .csv file
- **outlier_angle** (*float, optional, default: 5.*) – Coil orientation outlier "cone" around axes in +- deg. All zaps with coil orientations outside of this cone are removed.
- **outlier_loc** (*float, optional, default: 3.*) – Coil position outlier "sphere" in +- mm. All zaps with coil locations outside of this sphere are removed.

Returns

- **<File>** (*.csv file*) – experimental_oc.csv file with outlier corrected coil positions
- **<Files>** (*.png files*) – Plot showing the coil orientations and locations (folder_of_fn_exp_out/COND_X_coil_position.png)
- *or*
- **exp** (*dict*) – Dictionary containing the outlier corrected experimental data

pynibs.expio.exp.**combine_nnav_mep**(*xml_paths*, *cfs_paths*, *im*, *coil_sn*, *nii_exp_path*, *nii_conform_path*, *patient_id*, *tms_pulse_time*, *drop_mep_idx*, *mep_onsets*, *nnav_system*, *mesh_approach='headreco'*, *temp_dir=None*, *cfs_data_column=0*, *channels=None*, *plot=False*, *start_mep=18*, *end_mep=35*)

Creates dictionary containing all experimental data.

Parameters

- **xml_paths** (*list of str*) – Paths to coil0-file and optionally coil1-file if there is no coil1-file, use empty string
- **cfs_paths** (*list of str*) – Paths to .cfs mep file
- **im** (*list of str*) – List of path to the instrument-marker-file or list of strings containing the instrument marker
- **coil_sn** (*str*) – Coil-serial-number
- **nii_exp_path** (*str*) – Path to the .nii file that was used in the experiment
- **nii_conform_path** (*str*) – Path to the conform*.nii file used to calculate the E-fields with SimNIBS
- **patient_id** (*str*) – Patient id
- **tms_pulse_time** (*float*) – Time in [s] of TMS pulse as specified in signal
- **drop_mep_idx** (*List of int or None*) – Which MEPs to remove before matching.
- **mep_onsets** (*List of int or None (Default: None)*) – If there are multiple .cfs per TMS Navigator sessions, onsets in [ms] of .cfs. E.g.: [0, 71186].
- **temp_dir** (*str, optional, default: None (fn_exp_mri_nii folder)*) – Directory to save temporary files (transformation .nii and .mat files)
- **cfs_data_column** (*int or list of int*) – Column(s) of dataset in .cfs file.
- **channels** (*list of str, optional, default: None*) – Channel names
- **nnav_system** (*str*) – Type of neuronavigation system (“Localite”, “Visor”)
- **mesh_approach** (*str, optional, default: "headreco"*) – Approach the mesh is generated with (“headreco” or “mri2mesh”)
- **plot** (*bool, optional, default: False*) – Plot MEPs and p2p evaluation
- **start_mep** (*float, optional, default: 18*) – Start of time frame after TMS pulse where p2p value is evaluated (in ms)
- **end_mep** (*float, optional, default: 35*) – End of time frame after TMS pulse where p2p value is evaluated (in ms)

Returns

dict_lst – ‘number’ ‘condition’ ‘current’ ‘mep_raw_data’ ‘mep’ ‘mep_latency’ ‘mep_filt_data’ ‘mep_raw_data_time’ ‘time_tms’ ‘ts_tms’ ‘time_mep’ ‘date’ ‘coil_sn’ ‘patient_id’

Return type

list of dicts, one dict for each zap

pynibs.expio.exp.**combine_nnav_rt**(*xml_paths*, *behavior_paths*, *im*, *coil_sn*, *nii_exp_path*, *nii_conform_path*, *patient_id*, *drop_trial_idx*, *nnav_system*, *cond*, *mesh_approach='headreco'*, *temp_dir=None*, *plot=False*)

Creates dictionary containing all experimental data.

Parameters

- **xml_paths** (*list of str*) – Paths to coil0-file and optionally coil1-file if there is no coil1-file, use empty string
- **behavior_paths** (*list of str*) – Paths to .cfs mep file
- **im** (*list of str*) – List of path to the instrument-marker-file or list of strings containing the instrument marker

- **coil_sn** (*str*) – Coil-serial-number
- **nii_exp_path** (*str*) – Path to the .nii file that was used in the experiment
- **nii_conform_path** (*str*) – Path to the conform*.nii file used to calculate the E-fields with SimNIBS
- **patient_id** (*str*) – Patient id
- **drop_trial_idx** (*List of int or None*) – Which MEPs to remove before matching.
- **temp_dir** (*str, optional, default: None (fn_exp_mri_nii folder)*) – Directory to save temporary files (transformation .nii and .mat files)
- **nnav_system** (*str*) – Type of neuronavigation system ("Localite", "Visor")
- **cond** (*str*) – Condition name in data_path
- **mesh_approach** (*str, optional, default: "headreco"*) – Approach the mesh is generated with ("headreco" or "mri2mesh")
- **plot** (*bool, optional, default: False*) – Plot MEPs and p2p evaluation

Returns

dict_lst – 'number' 'condition' 'current' 'mep_raw_data' 'mep' 'mep_latency' 'mep_filt_data' 'mep_raw_data_time' 'time_tms' 'ts_tms' 'time_mep' 'date' 'coil_sn' 'patient_id'

Return type

list of dicts, one dict for each zap

pynibs.expio.exp.**convert_csv_to_hdf5**(*fn_csv*, *fn_hdf5*, *overwrite_arr=True*, *verbose=False*)

Wrapper from experiment.csv to experiment.hdf5

Saves all relevant columns from the (old) experiment.csv file to an .hdf5 file. fn_hdf5:/stim_data/

|–coil_sn |–current |–date |–time_diff |–time_mep |–time_tms |–ts_tms |–coil0 # <- all coil0_** columns |–coil1 # <- all coil1_** columns |–coil_mean # <- all coil_mean_** columns

All columns not found in experiment.csv are ignored (and a warning is thrown).

Parameters

- **fn_csv** (*str*) – experiment.csv filename
- **fn_hdf5** (*str*) – experiment.hdf5 filename. File is created if not existing.
- **overwrite_arr** (*bool*) – Overwrite existing arrays. Otherwise: fail. Default: True.
- **verbose** (*bool*) – Print some information (default: false).

pynibs.expio.exp.**get_cnt_infos**(*fn_cnt*)

Read some meta information from .cnt file

Return type

dict d

pynibs.expio.exp.**get_coil_flip_m**(*source_system='simnibs'*, *target_system=None*)

Returns a flimp matrix 4x4 to flip coil axis from one system to another.

Parameters

- **source_system** (*str*) – Atm only possible source: 'simnibs'
- **target_system** – tmsnavigator, visor, brainsight

Returns

flip_m – shape: 4x4

Return type
np.ndarray

pynibs.expio.exp.**get_coil_sn_lst**(*fn_coil*)

pynibs.expio.exp.**get_csv_matrix**(*dictionary*)

pynibs.expio.exp.**get_intensity_e**(*e1*, *e2*, *target1*, *target2*, *radius1*, *radius2*, *headmesh*, *rmt=1*, *roi='midlayer_lh_rh'*, *verbose=False*)

Computes the stimulator intensity adjustment factor based on the electric field

Parameters

- **e1** (*str*) – .hdf5 e field with midlayer
- **e2** (*str*) – .hdf5 e field with midlayer
- **target1** (*np.ndarray (3,)*) – Coordinates of cortical site of MT
- **target2** (*np.ndarray (3,)*) – Coordinates of cortical target site
- **radius1** (*float*) – Electric field of field1 is averaged over elements inside this radius around target1
- **radius2** (*float*) – Electric field of field2 is averaged over elements inside this radius around target2
- **headmesh** (*str*) – .hdf5 headmesh
- **rmt** (*float, optional, default=0*) – Resting motor threshold to be corrected
- **roi** (*str, optional, default='midlayer_lh_rh'*) – Name of roi. Expected to sit in mesh['/data/midlayer/roi_surface/']
- **verbose** (*bool, optional, Default: false*) – Print verbosity information.

Returns
rmt_e_corr – Adjusted stimulation intensity for target2

Return type
float

pynibs.expio.exp.**get_intensity_e_old**(*mesh1*, *mesh2*, *target1*, *target2*, *radius1*, *radius2*, *rmt=1*, *verbose=False*)

Computes the stimulator intensity adjustment factor based on the electric field

Something weird is going on here - check simnibs coordinates of midlayer before usage.

Parameters

- **mesh1** (*str or simnibs.msh.mesh_io.Msh*) – Midlayer mesh containing results of the optimal coil position of MT in the midlayer (e.g.: …/subject_overlays/00001.hd_fixed_TMS_1-0001_MagVenture_MCF_B65_REF_highres.ccd_scalar_central.msh)
- **mesh2** (*str or simnibs.msh.mesh_io.Msh*) – Midlayer mesh containing results of the optimal coil position of the target in the midlayer (e.g.: …/subject_overlays/00001.hd_fixed_TMS_1-0001_MagVenture_MCF_B65_REF_highres.ccd_scalar_central.msh)
- **target1** (*np.ndarray (3,)*) – Coordinates of cortical site of MT
- **target2** (*np.ndarray (3,)*) – Coordinates of cortical target site
- **radius1** (*float*) – Electric field in target 1 is averaged over elements inside this radius
- **radius2** (*float*) – Electric field in target 2 is averaged over elements inside this radius
- **rmt** (*float, optional, default=0*) – Resting motor threshold, which will be corrected

- **verbose** (*bool, optional, Default: false*) – Print verbosity information.

Returns
rmt_e_corr – Adjusted stimulation intensity for target2

Return type
float

pynibs.expio.exp.**get_intensity_stokes**(*mesh*, *target1*, *target2*, *spat_grad=3*, *rmt=0*, *verbose=False*, *scalp_tag=1005*, *roi=None*)

Computes the stimulator intensity adjustment factor according to Stokes et al. 2005 (doi:10.1152/jn.00067.2005). Adjustment is based on target-scalp distance differences: adj = (Dist2-Dist1)*spat_grad

Parameters

- **mesh** (*str or simnibs.msh.mesh_io.Msh*) – Mesh of the head model
- **target1** (*np.ndarray (3,)*) – Coordinates of cortical site of MT
- **target2** (*np.ndarray (3,)*) – Coordinates of cortical target site
- **spat_grad** (*float*) – Spatial gradient. Default: 3
- **rmt** (*float, optional, default=0*) – Resting motor threshold, which will be corrected
- **scalp_tag** (*int, optional, default: 1005*) – Tag in the mesh where the scalp is to be set. Default: 1005
- **verbose** (*bool, optional, Default: false*) – Print verbosity information.
- **roi** (*np.ndarray (3,N)*) – Array of nodes to project targets onto

Returns
rmt_stokes – Adjusted stimulation intensity for target2

Return type
float

pynibs.expio.exp.**get_patient_id**(*xml_path*)

Read patiend-ID.

Parameters
xml_path (*str*) – Path to coil0-file

Returns
xml_pd.find('patientID').text – ID of patient

Return type
str

pynibs.expio.exp.**get_trial_data_from_csv**(*behavior_fn*, *cond*, *drop_trial_idx=None*, *only_corr=True*, *startatzero=True*)

Reads trial data from csv file. Reaction time is in [0.1ms]

Parameters
behavior_fn (*str*) – Filename with columns: 'trialtype', 'onset_time', 'rt'

cond
[str] Which condition to choose from .csv file

drop_trial_idx
[list of int, optional] 'trialnum' to remove

only_corr
[bool, default: True] Only return trials with correct responses

startatzero
[bool, default: True] Shift onset_time axis to zero

Returns

- **rt** (*list of float*)
- **onsets** (*list of float*)
- **mean_isi** (*tuple of int*) – in [s]

pynibs.expio.exp.**load_matsimnibs_txt**(*fn_matsimnibs*)

Loading matsimnibs matrices from .txt file.

Parameters
fn_matsimnibs (*str*) – Filename of .txt file the matsimnibs matrices are stored in

Returns
matsimnibs – Tensor containing matsimnibs matrices

Return type
ndarray of float [4 x 4] or [4 x 4 x n_mat]

pynibs.expio.exp.**match_behave_and_triggermarker**(*mep_time_lst*, *xml_paths*, *bnd_factor=0.495*, *isi=None*)

Sort out timestamps of mep and tms files that do not match.

Parameters

- **mep_time_lst** (*list of datetime.timedelta*) – timedeltas of MEP recordings.
- **xml_paths** (*list of str*) – Paths to coil0-file and optionally coil1-file if there is no coil1-file, use empty string
- **bnd_factor** (*float, optional, default: 0.99/2*) – Bound factor relative to interstimulus interval in which +- interval to match neuronavigation and mep data from their timestamps. (0 means perfect matching, 0.5 means +- half interstimulus interval)
- **isi** (*float, optional.*) – Interstimulus intevals. Of not provided it's estimated from the first trial.

Returns

- **tms_index_lst** (*list of int*) – Indices of tms-timestamps that match
- **mep_index_lst** (*list of int*) – Indices of mep-timestamps that match
- **tms_time_lst** (*list of datetime*) – TMS timstamps

pynibs.expio.exp.**nnav2simnibs**(*fn_exp_nii*, *fn_conform_nii*, *m_nnav*, *nnav_system*, *mesh_approach='headreco'*, *fiducials=None*, *orientation='RAS'*, *fsl_cmd=None*, *target='simnibs'*, *temp_dir=None*, *rem_tmp=False*, *verbose=True*)

Transforming TMS coil positions from neuronavigation to simnibs space

Parameters

- **fn_exp_nii** (*str*) – Filename of .nii file the experiments were conducted with
- **fn_conform_nii** (*str*) – Filename of .nii file from SimNIBS mri2msh function (e.g.: .../fs_subjectID/subjectID_T1fs_conform.nii.gz)
- **m_nnav** (*np.ndarray [4 x 4 x N]*) – Position matrices from neuronavigation
- **nnav_system** (*str*) – Neuronavigation system: - "Localite" ... Localite neuronavigation system - "Visor" ... Visor neuronavigation system from ANT - "Brainsight" ... Brainsight neuronavigation system from Rougue Research

- **mesh_approach** (*str, optional, default: "headreco"*) – Approach the mesh is generated with ("headreco" or "mri2mesh")
- **fiducials** (*np.array of float [3 x 3]*) – Fiducial points in ANT nifti space from file (e.g.: /data/pt_01756/probands/33791.b8/exp/1/33791.b8_recording/MRI/33791.b8_recording.mri) (x frontal-occipital, y right-left, z inferior-superior) VoxelOnPositiveXAxis (Nasion, first row) 221 131 127 VoxelOnPositiveYAxis (left ear, second row) 121 203 105 VoxelOnNegativeYAxis (right ear, third row) 121 57 105
- **orientation** (*str*) – Orientation convention ('RAS' or 'LPS') (can be read from neuronavigation .xml file under coordinateSpace="RAS")
- **fsl_cmd** (*str, optional*) – bash command to start FSL environment (Default: None)
- **target** (*str, optional, default: 'simnibs'*) – Either transform to 'simnibs' or to 'nnav' space
- **temp_dir** (*str, optional, default: None (fn_exp_mri_nii folder)*) – Directory to save temporary files (transformation .nii and .mat files)
- **rem_tmp** (*bool, optional, default: False*) – Remove temporary files from registration
- **verbose** (*boolean*) – Print output (True / False)

Returns
m_simnibs

Return type
np.ndarray of float [4 x 4 x N]

pynibs.expio.exp.**outliers_mask**(*data*, *m=2.0*)

pynibs.expio.exp.**print_time**(*relation*, *tms_time*, *tms_time_index*, *mep_time*, *mep_time_index*, *time_bnd_l*, *time_bnd_h*)

Print timestamps that do not match.

Parameters

- **relation** (*str*) – 'bigger' or 'smaller'
- **tms_time** (*datetime.timedelta*) – TMS timestamps
- **tms_time_index** (*int*) – Index of tms timestamp
- **mep_time** (*datetime.timedelta*) – Mep timestamps
- **mep_time_index** (*int*) – Index of mep timestamps
- **time_bnd_l** (*datetime.timedelta*) – Lowest datetime timestamp for matching
- **time_bnd_h** (*datetime.timdelta*) – Highest datetime timestamp for matching

pynibs.expio.exp.**read_csv**(*csv_path*)

Read dictionary from .csv-file.

Parameters
csv_path (*str*) – Path to .csv-file

Returns
dict_lst – Field name of the .csv-file as the key

Return type
dict of list

pynibs.expio.exp.**read_exp_stimulations**(*fname_results_conditions*, *fname_simpos*, *filter_bad_trials=False*, *drop_idx=None*)

Reads results_conditions.csv and simPos.csv and returns data.

Parameters

- **fname_results_conditions** (*str*) – Filename of results_conditions.csv file
- **fname_simpos** (*str*) – Filename of simPos.csv file
- **filter_bad_trials** (*bool, False*) – If true, some filtering will be done to exclude erroneous data
- **drop_idx** (*list, empty*) – Indices of trials to drop

Returns

- **positions_all** (*list of np.ndarrays of float [N_zaps x [4 x 4]]*) – List of position matrices of TMS coil, formatted in simnibs style

$$\begin{bmatrix} | & | & | & | \\ x & y & z & pos \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **conditions** (*list of str [N_zaps]*) – Str labels of the condition corresponding to each zap
- **position_list** (*list of float and str [N_zaps x 55]*) – List of data stored in results_conditions.csv (condition, MEP amplitude, locations of neuronavigation trackers)
- **mep_amp** (*np.array of float [N_zaps]*) – MEP amplitude in [V] corresponding to each zap
- **intensities** (*np.array of float [N_zaps]*) – Stimulator intensities corresponding to each zap
- **fails_idx** (*np.array(N_fails_idx x 1) (only if filter_bad_trials)*) – Which trials were dropped through filtering?

pynibs.expio.exp.**save_matsimnibs_txt**(*fn_matsimnibs*, *matsimnibs*)

Saving matsimnibs matrices in .txt file.

Parameters

- **fn_matsimnibs** (*str*) – Filename of .txt file the matsimnibs matrices are stored in
- **matsimnibs** (*ndarray of float [4 x 4] or [4 x 4 x n_mat]*) – Tensor containing matsimnibs matrices

Returns

<File> – Textfile containing the matsimnibs matrices

Return type

.txt file

pynibs.expio.exp.**sort_by_condition**(*exp*, *conditions_selected=None*)

Sort experimental dictionary from experimental.csv into list by conditions.

Parameters

- **exp** (*dict or list of dict*) – Dictionary containing the experimental data information
- **conditions_selected** (*str or list of str, Default=None*) – List of conditions returned by the function (in this order), the others are omitted, If None, all conditions are returned

Returns

exp_cond – List of dictionaries containing the experimental data information sorted by condition

Return type

list of dict

pynibs.expio.exp.**sort_data_by_condition**(*conditions*, *return_alph_sorted=True*, *conditions_selected=None*, **data*)

Sorts data by condition and returns tuples of data with corresponding labels.

Parameters

- **conditions** (*list of str [N_zaps]*) – Str labels of the condition corresponding to each data
- **return_alph_sorted** (*Boolean, Default True*) – Shall returns be in alphabetically or original order
- **conditions_selected** (*list of str or None*) – List of conditions returned by the function (in this order), the others are omitted
- **data** (*tuple of data indexed by condition [N_data x N_zaps x m]*) – Data to sort

Returns

- **cond_labels** (*list of str [N_cond]*) – Labels of conditions
- **data_sorted** (*tuple of sorted data [N_cond x N_data x N_zaps x m]*) – Sorted data by condition

pynibs.expio.exp.**splitext_niigz**(*fn*)

Splitting extension(s) from .nii or .nii.gz file

Parameters

fn (*str*) – Filename of input image .nii or .nii.gz file

Returns

- **path** (*str*) – Path and filename without extension(s)
- **ext** (*str*) – Extension, either .nii or .nii.gz

pynibs.expio.exp.**square**(*x*, *a*, *b*, *c*)

Parametrized quadratic function

$$y = ax^2 + bx + c$$

Parameters

- **x** (*np.ndarray of float [N_x]*) – X-values the function is evaluated in
- **a** (*float*) – Slope parameter of x^2
- **b** (*float*) – Slope parameter of x
- **c** (*float*) – Offset parameter

Returns

y – Function value at argument x

Return type

np.ndarray of float [N_x]

pynibs.expio.exp.**toRAS**(*fn_in*, *fn_out*)

Transforming MRI .nii image to RAS space.

Parameters

- **fn_in** (*str*) – Filename of input image .nii file
- **fn_out** (*str*) – Filename of output image .nii file in RAS space

Returns

<File> – .nii image in RAS space (fn_out)

Return type
.nii file

pynibs.expio.exp.**write_csv**(*csv_output_path*, *dict_lst*)

Write dictionary into .csv-file.

Parameters

- **csv_output_path** (*str*) – Path to output-file
- **dict_lst** (*list of dict*) – Fields of the .csv-file

pynibs.expio.exp.**write_triggermarker_stats**(*tm_array*, *idx_keep*, *idx_outlier*, *idx_zero*, *fn*, ***kwargs*)

Write some stats about the triggermarker analyses to a .csv . Use kwargs to add some more information, like subject id, experiment, conditions, etc

pynibs.expio.localite module

Functions to import data from Localite TMS Navigator

pynibs.expio.localite.**arrays_similar**(*tm_matrix*, *tm_matrix_post*, *pos_rtol=0*, *pos_atol=3.6*, *ang_rtol=0.1*, *ang_atol=0.1*)

Compares angles and position for similarity.

Splitting the comparison into angles and position is angebracht, as the absolute tolereance (atol) should be different for angles (degree) and position (millimeter) comparisons.

Parameters

- **tm_matrix** (*array-like, shape = (4,4)*) – TMS Navigator triggermarker or instrument marker array
- **tm_matrix_post** (*array-like, shape = (4,4)*) – TMS Navigator triggermarker or instrument marker array
- **tm_mean_last** (*array-like, shape = (4,4), optional*) – Mean TMS Navigator triggermarker or instrument marker array for n zaps

pynibs.expio.localite.**get_instrument_marker**(*im_path*)

pynibs.expio.localite.**get_marker**(*im_path*, *markertype*)

Read instrument-marker and conditions from Neuronavigator .xml-file.

Parameters
im_path (*str or list of str*) – Path to instrument-marker-file

Returns

- **im_array** (*np.array of float [Mx4x4]*) – Instrument-marker-matrices
- **im_cond_lst** (*list of str*) – Labels of the instrument-marker-conditions
- **im_marker_times** (*list of float*) – Onset times

pynibs.expio.localite.**get_single_marker_file**(*im_path*, *markertype*)

Read instrument-marker and conditions from Neuronavigator .xml-file.

Parameters
im_path (*str or list of str*) – Path to instrument-marker-file

Returns

- **im_array** (*np.array of float [Mx4x4]*) – Instrument-marker-matrices
- **im_cond_lst** (*list of str*) – Labels of the instrument-marker-conditions

pynibs.expio.localite.**get_tms_elements**(*xml_paths*, *verbose=False*)

Read needed elements out of the tms-xml-file.

Parameters

- **xml_paths** (*list of str or str*) – Paths to coil0-file and optionally coil1-file if there is no coil1-file, use empty string
- **verbose** (*bool, optional, default: False*) – Print output messages

Returns

- **coils_array** (*nparray of float [3xNx4x4]*) – Coil0, coil1 and mean-value of N 4x4 coil-arrays
- **ts_tms_lst** (*list of int [N]*) – Timestamps of valid tms-measurements
- **current_lst** (*list of int [N]*) – Measured currents
- **idx_invalid** (*list of int*) – List of indices of invalid coil positions (w.r.t. all timestamps incl invalid)

pynibs.expio.localite.**marker_is_empty**(*arr*)

pynibs.expio.localite.**markers_are_empty**(*arr*)

pynibs.expio.localite.**match_instrument_marker_file**(*xml_paths*, *im_path*)

Assign right instrument marker condition to every triggermarker (get instrument marker out of file).

Parameters

- **xml_paths** (*list of str*) – Paths to coil0-file and optionally coil1-file if there is no coil1-file, use empty string
- **im_path** (*str*) – Path to instrument-marker-file

Returns
coil_cond_lst – Right conditions

Return type
list of str

pynibs.expio.localite.**match_instrument_marker_string**(*xml_paths*, *condition_order*)

Assign right instrument marker condition to every triggermarker (get instrument marker out of list of strings).

Parameters

- **xml_paths** (*list of str*) – Paths to coil0-file and optionally coil1-file if there is no coil1-file, use empty string
- **condition_order** (*list of str*) – Conditions in the right order

Returns
coil_cond_lst

Return type
list of strings containing the right conditions

pynibs.expio.localite.**match_tm_to_im**(*tm_array*, *im_array*, *tms_time_arr*, *tms_cur_arr*)

Match triggermarker with instrumentmarker and get index of best fitting instrumentmarker.

Parameters

- **tm_array** (*ndarray of float [Nx4x4]*) – Mean-value of Nx4x4 coil matrices
- **im_array** (*ndarray of float [Mx4x4]*) – Instrument-marker-matrices

Returns
im_index_lst – Indices of best fitting instrumentmarkers

Return type
list of int

pynibs.expio.localite.**merge_exp_data_localite**(*subject*, *coil_outlier_corr_cond*, *remove_coil_skin_distance_outlier*, *coil_distance_corr*, *exp_idx=0*, *mesh_idx=0*, *drop_mep_idx=None*, *mep_onsets=None*, *cfs_data_column=None*, *channels=None*, *verbose=False*, *plot=False*, *start_mep=18*, *end_mep=35*)

Merge the TMS coil positions (TriggerMarker) and the mep data into an experiment.hdf5 file.

Parameters

- **subject** (*pyfempp.subject*) – Subject object
- **exp_idx** (*str*) – Experiment ID
- **mesh_idx** (*str*) – Mesh ID
- **coil_outlier_corr_cond** (*bool*) – Correct outlier of coil position and orientation (+-2 mm, +-3 deg) in case of conditions
- **remove_coil_skin_distance_outlier** (*bool*) – Remove outlier of coil position lying too far away from the skin surface (+- 5 mm)
- **coil_distance_corr** (*bool*) – Perform coil <-> head distance correction (coil is moved towards head surface until coil touches scalp)
- **drop_mep_idx** (*List of int or None*) – Which MEPs to remove before matching.
- **mep_onsets** (*List of int or None (Default: None)*) – If there are multiple .cfs per TMS Navigator sessions, onsets in [ms] of .cfs. E.g.: [0, 71186].
- **cfs_data_column** (*int or list of int*) – Column(s) of dataset in .cfs file.
- **channels** (*list of str*) – List of channel names
- **verbose** (*bool*) – Plot output messages
- **plot** (*bool, optional, default: False*) – Plot MEPs and p2p evaluation
- **start_mep** (*float, optional, default: 18*) – Start of time frame after TMS pulse where p2p value is evaluated (in ms)
- **end_mep** (*float, optional, default: 35*) – End of time frame after TMS pulse where p2p value is evaluated (in ms)

pynibs.expio.localite.**merge_exp_data_rt**(*subject*, *coil_outlier_corr_cond*, *remove_coil_skin_distance_outlier*, *coil_distance_corr*, *cond=None*, *exp_idx=0*, *mesh_idx=0*, *drop_trial_idx=None*, *verbose=False*, *plot=False*)

Merge the TMS coil positions (TriggerMarker) and the mep data into an experiment.hdf5 file.

Parameters

- **subject** (*pynibs.subject*) – Subject object
- **exp_idx** (*str*) – Experiment ID
- **mesh_idx** (*str*) – Mesh ID
- **coil_outlier_corr_cond** (*bool*) – Correct outlier of coil position and orientation (+-2 mm, +-3 deg) in case of conditions
- **cond** (*string*) – Which condition to analyse
- **remove_coil_skin_distance_outlier** (*bool*) – Remove outlier of coil position lying too far away from the skin surface (+- 5 mm)

- **coil_distance_corr** (*bool*) – Perform coil <-> head distance correction (coil is moved towards head surface until coil touches scalp)
- **drop_trial_idx** (*List of int or None*) – Which MEPs to remove before matching.
- **verbose** (*bool*) – Plot output messages
- **plot** (*bool, optional, default: False*) – Plot MEPs and p2p evaluation

pynibs.expio.localite.**read_coil**(*xml_ma*)

Read coil-data from xml element.

Parameters
xml_ma (*xml-element*) – Coil data

Returns
coil – Coil elements

Return type
nparray of float [4x4]

pynibs.expio.localite.**read_triggermarker_localite**(*fn_xml*)

Read instrument-marker and conditions from neuronavigator .xml-file.

Parameters
fn_xml (*str*) – Path to TriggerMarker.xml file (e.g. Subject_0/Sessions/Session_YYYYMMDDHHMMSS/TMSTrigger/TriggerMarkers_Coil1_YYYYMMDDHHMMSS

Returns

m_nnav
[ndarray of float [4x4xN]] Neuronavigation coordinates of N stimuli (4x4 matrices)

didt
[ndarray of float [N]] Rate of change of coil current in (A/us)

stim_int
[ndarray of float [N]] Stimulator intensity in (%)

descr
[list of str [N]] Labels of the instrument-marker-conditions

rec_time
[list of str [N]] Recording time in ms

Return type
list of

pynibs.expio.visor module

Functions to import data from ANT Visor 2 / ANT EEG software go here

pynibs.expio.visor.**filter_emg**(*emg*, *fs*)

Filter emg signals

Parameters

- **emg** (*list of np.array [n_stimuli]*) – Raw EMG data. Each list entry contains a np.ndarray of size [n_samples x n_channel] Each channel is filtered in the same way.
- **fs** (*float*) – Sampling frequency

Returns
emg_filt – Filtered EMG data

Return type
list of np.ndarray [n_stimuli]

pynibs.expio.visor.**get_cnt_data**(*fn*, *channels='all'*, *trigger_val='1'*, *max_duration=10*, *fn_hdf5=None*, *path_hdf5=None*, *verbose=False*, *return_data=False*)

Reads ANT .cnt EMG/EEG data file and chunks timesieries into triggerN - trigggerN+1 Can directly write the zaps into hdf5 if argument is provided. Starting with the first trigger and ending with get_sample_count()-1

Parameters

- **fn** (*str*) – .cnt Filename
- **channels** (*str | int | list of int | list of str*) – Which channel(s) to return. Default: all. Can be channel number(s) or channel name(s).
- **trigger_val** (*str*) – Trigger value to read as zap trigger. Default: '1'
- **max_duration** (*int*) – Maximum duration in [s] per chunk. Rest is dropped.
- **fn_hdf5** (*str, optional, default: None*) – If given, cnt data is written into hdf5 file under "path_hdf5" as pandas dataframe with column name "qoi_name" and nothing is returned. Default: None
- **path_hdf5** (*str, optional, default: None*) – If fn_hdf5, path of pandas dataframe where the data is saved (e.g. "/phys_data/raw/EEG")
- **verbose** (*bool*) – Some verbosity messages (default: False)
- **return_data** (*bool*) – Return data as list of np.array

Returns
data_lst – List of EEG/EMG data. Only returned if fn_hdf5 is not None

Return type
list of np.ndarray with shape=(samples,channels), optional

pynibs.expio.visor.**get_instrument_marker**(*im_path*, *verbose=False*)

Return all instrument markers from visor .cnt file Coordinate system in raw ANT space (NLR) defined as:

origin: intersection between line of ear fiducials and nasion x-axis: origin -> nasion y-axis: origin -> left ear z-axis: origin -> superior

Parameters

- **im_path** (*str*) – Path to instrument-marker-file .cnt file
- **verbose** (*bool*) – Some verbosity messages (default: False)

Returns
im_list – List containing stimulation parameters: - coil_mean_raw [4 x 4 matrix] - StimulusID - etc…

Return type
list of dict

pynibs.expio.visor.**merge_exp_data_visor**(*subject*, *exp_id=0*, *mesh_idx=0*, *verbose=False*, *start_mep=18*, *end_mep=35*)

Merges all experimental data from visor experiment into one .hdf5 file

Parameters

- **subject** (*Subject object*) – Subject object
- **exp_id** (*int*) – Experiment index
- **mesh_idx** (*int*) – Mesh index
- **verbose** (*bool*) – Print output

- **start_mep** (*float, optional, default: 18*) – Start of time frame after TMS pulse where p2p value is evaluated (in ms)
- **end_mep** (*float, optional, default: 35*) – End of time frame after TMS pulse where p2p value is evaluated (in ms)

Returns
<File> – File containing the stimulation and physiological data as pandas dataframes: - "stim_data": Stimulation parameters (e.g. coil positions, etc.) - "phys_data/info/EMG": Information about EMG data recordings (e.g. sampling frequency, etc.) - "phys_data/info/EEG": Information about EEG data recordings (e.g. sampling frequency, etc.) - "phys_data/raw/EMG": Raw EMG data - "phys_data/raw/EEG": Raw EEG data - "phys_data/postproc/EMG": Post-processed EMG data (e.g. filtered, p2p, etc.) - "phys_data/postproc/EEG": Post-processed EEG data (e.g. filtered, p2p, etc.)

Return type
.hdf5 file

pynibs.expio.visor.**read_nlr**(*fname*)

Reads NLR coordinates from *_recording.mri file

Parameters
fname –

Returns
fiducials – The rows contain the fiducial points in ANT nifit space (nasion, left ear, right ear)

Return type
np.array of float [3 x 3]

pynibs.mesh package

Submodules

pynibs.mesh.mesh_struct module

class pynibs.mesh.mesh_struct.**Mesh**(*mesh_name*, *subject_id*, *subject_folder*)

Bases: object

" Mesh class to initialize default attributes.

fill_defaults(*approach*)

Initializes attributes for a headreco mesh.

Parameters
approach (*str*) – 'headreco' 'mri2mesh' 'charm'

print()

Print self information.

write_to_hdf5(*fn_hdf5*, *check_file_exist=False*, *verbose=False*)

Write this mesh' attributes to .hdf5 file.

Parameters

- **fn_hdf5** (*str*) –
- **check_file_exist** (*bool*) – Check if provided filenames exist, warn if not.
- **verbose** (*bool*) – Print self information

class pynibs.mesh.mesh_struct.**ROI**(*subject_id*, *roi_name*, *mesh_name*)

Bases: object

" Region of interest class to initialize default attributes.

print()

Print self information.

write_to_hdf5(*fn_hdf5*, *check_file_exist=False*, *verbose=False*)

Write this mesh' attributes to .hdf5 file.

Parameters

- **fn_hdf5** (*str*) –
- **check_file_exist** (*bool*) – Check if provided filenames exist, warn if not.
- **verbose** (*bool*) – Print self information

class pynibs.mesh.mesh_struct.**TetrahedraLinear**(*points*, *triangles*, *triangles_regions*, *tetrahedra*, *tetrahedra_regions*)

Bases: object

Mesh, consisting of linear tetrahedra.

Parameters

- **points** (*array of float [N_points x 3]*) – Vertices of FE mesh
- **triangles** (*np.ndarray of int [N_tri x 3]*) – Connectivity of points forming triangles
- **triangles_regions** (*np.ndarray of int [N_tri x 1]*) – Region identifiers of triangles
- **tetrahedra** (*np.ndarray of int [N_tet x 4]*) – Connectivity of points forming tetrahedra
- **tetrahedra_regions** (*np.ndarray of int [N_tet x 1]*) – Region identifiers of tetrahedra

N_points

Number of vertices

Type
int

N_tet

Number of tetrahedra

Type
int

N_tri

Number of triangles

Type
int

N_region

Number of regions

Type
int

region

Region labels

Type
np.ndarray of int

tetrahedra_volume

Volumes of tetrahedra

Type
np.ndarray of float [N_tet x 1]

tetrahedra_center

Center of tetrahedra

Type
np.ndarray of float [N_tet x 1]

triangles_center

Center of triangles

Type
np.ndarray of float [N_tri x 1]

triangles_normal

Normal components of triangles pointing outwards

Type
np.ndarray of float [N_tri x 3]

calc_E(*grad_phi*, *omegaA*)

Calculate electric field with gradient of electric potential and omega-scaled magnetic vector potential A.

$$\mathbf{E} = -\nabla\varphi - \omega\mathbf{A}$$

Parameters

- **grad_phi** (*np.ndarray of float [N_tet x 3]*) – Gradient of Scalar DOF in tetrahedra center
- **omegaA** (*np.ndarray of float [N_tet x 3]*) – Magnetic vector potential in tetrahedra center (scaled with angular frequency omega)

Returns
E – Electric field in tetrahedra center

Return type
np.ndarray of float [N_tet x 3]

calc_E_normal_tangential_surface(*E*, *fname*)

Calculate normal and tangential component of electric field on given surfaces of mesh instance.

Parameters

- **E** (*np.ndarray of float [N_tri x 3]*) – Electric field data on surfaces
- **fname** (*str*) – Filename of the .txt file containing the tetrahedra indices, which are adjacent to the surface triangles generated by the method "calc_surface_adjacent_tetrahedra_idx_list(self, fname)"

Returns

- **En_pos** (*np.ndarray of float [N_tri x 3]*) – Normal component of electric field of top side (outside) of surface
- **En_neg** (*np.ndarray of float [N_tri x 3]*) – Normal component of electric field of bottom side (inside) of surface

- **n** (*np.ndarray of float [N_tri x 3]*) – Normal vector
- **Et** (*np.ndarray of float [N_tri x 3]*) – Tangential component of electric field lying in surface
- **t** (*np.ndarray of float [N_tri x 3]*) – Tangential vector

`calc_E_on_GM_WM_surface`(*E*, *roi*)

Determines the normal and tangential component of the induced electric field on a GM-WM surface using nearest neighbour principle.

Parameters

- **E** (*np.ndarray of float [N_tri x 3]*) – Induced electric field given in the tetrahedra centre of the mesh instance
- **roi** (pynibs.roi.RegionOfInterestSurface) – RegionOfInterestSurface object class instance

Returns

- **E_normal** (*np.ndarray of float [N_points x 3]*) – Normal vector of electric field on GM-WM surface
- **E_tangential** (*np.ndarray of float [N_points x 3]*) – Tangential vector of electric field on GM-WM surface

`calc_E_on_GM_WM_surface3`(*phi*, *dAdt*, *roi*, *verbose=True*, *mode='components'*)

Determines the normal and tangential component of the induced electric field on a GM-WM surface by recalculating phi and dA/dt in an epsilon environment around the GM/WM surface (upper and lower GM-WM surface).

Parameters

- **phi** (*np.ndarray of float [N_nodes x 1]*) – Scalar electric potential given in the nodes of the mesh
- **dAdt** (*np.ndarray of float [N_nodes x 3]*) – Magnetic vector potential given in the nodes of the mesh
- **roi** (*object instance*) – RegionOfInterestSurface object class instance
- **verbose** (*boolean*) – Print information to stdout
- **mode** (*str*) – Select mode of output: - "components" : return x, y, and z component of tangential and normal components - "magnitude" : return magnitude of tangential and normal component (normal with sign for direction)

Returns

- **E_normal** (*np.ndarray of float [N_points x 3]*) – Normal vector of electric field on GM-WM surface
- **E_tangential** (*np.ndarray of float [N_points x 3]*) – Tangential vector of electric field on GM-WM surface

`calc_E_on_GM_WM_surface_simnibs`(*phi*, *dAdt*, *roi*, *subject*, *verbose=False*, *mesh_idx=0*)

Determines the normal and tangential component of the induced electric field on a GM-WM surface by recalculating phi and dA/dt in an epsilon environment around the GM/WM surface (upper and lower GM-WM surface) or by using the Simnibs interpolation function.

Parameters

- **phi** (*np.ndarray of float [N_nodes x 1]*) – Scalar electric potential given in the nodes of the mesh
- **dAdt** (*np.ndarray of float [N_nodes x 3]*) – Magnetic vector potential given in the nodes of the mesh

- **roi** (*object instance*) – RegionOfInterestSurface object class instance
- **subject** (*Subject object*) – Subject object loaded from .hdf5 file
- **verbose** (*boolean*) – Print information to stdout
- **mesh_idx** (*int*) – Mesh index

Returns

- **E_normal** (*np.ndarray of float [N_points x 3]*) – Normal vector of electric field on GM-WM surface
- **E_tangential** (*np.ndarray of float [N_points x 3]*) – Tangential vector of electric field on GM-WM surface

calc_E_on_GM_WM_surface_simnibs_KW(*phi*, *dAdt*, *roi*, *subject*, *verbose=False*, *mesh_idx=0*)

Determines the normal and tangential component of the induced electric field on a GM-WM surface by recalculating phi and dA/dt in an epsilon environment around the GM/WM surface (upper and lower GM-WM surface) or by using the Simnibs interpolation function.

Parameters

- **phi** (*np.ndarray of float [N_nodes x 1]*) – Scalar electric potential given in the nodes of the mesh
- **dAdt** (*np.ndarray of float [N_nodes x 3]*) – Magnetic vector potential given in the nodes of the mesh
- **roi** (*object instance*) – RegionOfInterestSurface object class instance
- **subject** (*Subject object*) – Subject object loaded from .hdf5 file
- **verbose** (*boolean*) – Print information to stdout
- **mesh_idx** (*int*) – Mesh index

Returns

- **E_normal** (*np.ndarray of float [N_points x 3]*) – Normal vector of electric field on GM-WM surface
- **E_tangential** (*np.ndarray of float [N_points x 3]*) – Tangential vector of electric field on GM-WM surface

calc_J(*E*, *sigma*)

Calculate current density J. The conductivity sigma is a list of np.arrays containing conductivities of regions (scalar and/or tensor).

$$\mathbf{J} = [\sigma]\mathbf{E}$$

Parameters

- **E** (*np.ndarray of float [N_tet x 3]*) – Electric field in tetrahedra center
- **sigma** (*list of np.ndarray of float [N_regions][3 x 3]*) – Conductivities of regions (scalar and/or tensor).

Returns
E – Electric field in tetrahedra center

Return type
np.ndarray of float [N_tet x 3]

calc_QOI_in_points(*qoi*, *points_out*)

Calculate QOI_out in points_out using the mesh instance and the quantity of interest (QOI).

Parameters

- **qoi** (*np.ndarray of float*) – Quantity of interest in nodes of tetrahedra mesh instance
- **points_out** (*np.ndarray of float*) – Point coordinates (x, y, z) where the qoi is going to be interpolated by linear basis functions

Returns
qoi_out – Quantity of interest in points_out

Return type
np.ndarray of float

`calc_QOI_in_points_tet_idx`(*qoi*, *points_out*, *tet_idx*)

Calculate QOI_out in points_out sitting in tet_idx using the mesh instance and the quantity of interest (QOI).

Parameters

- **qoi** (*np.ndarray of float*) – Quantity of interest in nodes of tetrahedra mesh instance
- **points_out** (*np.ndarray of float*) – Point coordinates (x, y, z) where the qoi is going to be interpolated by linear basis functions
- **tet_idx** (*np.ndarray of int*) – Element indices where the points_out are sitting

Returns
qoi_out – Quantity of interest in points_out

Return type
np.ndarray of float

`calc_gradient`(*phi*)

Calculate gradient of scalar DOF in tetrahedra center.

Parameters
phi (*np.ndarray of float [N_nodes]*) – Scalar DOF the gradient is calculated for

Returns
grad_phi – Gradient of Scalar DOF in tetrahedra center

Return type
np.ndarray of float [N_tet x 3]

`calc_surface_adjacent_tetrahedra_idx_list`(*fname*)

Determine the indices of the tetrahedra touching the surfaces and save the indices into a .txt file specified with fname.

Parameters
fname (*str*) – Filename of output .txt file

Returns
<File> – Element indices of the tetrahedra touching the surfaces (outer-most elements)

Return type
.txt file

`data_elements2nodes`(*data*)

Transforms an data in tetrahedra into the nodes after Zienkiewicz et al. (1992) [1]. Can only transform volume data, i.e. needs the data in the surrounding tetrahedra to average it to the nodes. Will not work well for discontinuous fields (like E, if several tissues are used).

Parameters
data (*np.ndarray [N_elements x N_data]*) – Data in tetrahedra

Returns
data_nodes – Data in nodes

Return type
np.ndarray [N_nodes x N_data]

Notes

`data_nodes2elements`(*data*)

Interpolate data given in the nodes to the tetrahedra center.

Parameters
data (*np.ndarray [N_nodes x N_data]*) – Data in nodes

Returns
data_elements – Data in elements

Return type
np.ndarray [N_elements x N_data]

`get_faces`(*tetrahedra_indexes=None*)

Creates a list of nodes in each face and a list of faces in each tetrahedra.

Parameters
tetrahedra_indexes (*np.ndarray*) – Indices of the tetrehedra where the faces are to be determined (default: all tetrahedra)

Returns

- **faces** (*np.ndarray*) – List of nodes in faces, in arbitrary order
- **th_faces** (*np.ndarray*) – List of faces in each tetrahedra, starts at 0, order=((0, 2, 1), (0, 1, 3), (0, 3, 2), (1, 2, 3))
- **face_adjacency_list** (*np.ndarray*) – List of tetrahedron adjacent to each face, filled with -1 if a face is in a single tetrahedron. Not in the normal element ordering, but only in the order the tetrahedra are presented

`get_outside_faces`(*tetrahedra_indexes=None*)

Creates a list of nodes in each face that are in the outer volume.

Parameters
tetrahedra_indices (*np.ndarray*) – Indices of the tetrehedra where the outer volume is to be determined (default: all tetrahedra)

Returns
faces – List of nodes in faces in arbitrary order

Return type
np.ndarray

pynibs.mesh.transformations module

pynibs.mesh.transformations.`cell_data_to_point_data`(*tris*, *data_tris*, *nodes*, *method='nearest'*)

A wrapper for scipy.interpolate.griddata to interpolate cell data to node data.

Parameters

- **tris** (*np.ndarray*) – element number list, (n_tri, 3)
- **data_tris** (*np.ndarray*) – data in tris, (n_tri x 3)
- **nodes** (*np.ndarray*) – nodes coordinates, (n_nodes, 3
- **method** (*str, default: 'nearest'*) – Which method to use for interpolation. Default uses NearestNDInterpolator

Returns
data_nodes – Data in nodes

Return type
np.ndarray

pynibs.mesh.transformations.**data_elements2nodes**(*data*, *con*, *precise=False*)

Transforms data in elements (triangles or tetrahedra) to nodes. Data can be list of multiple data arrays.

Parameters

- **data** (*np.ndarray of float or list of np.ndarray*) – (N_elements, N_data) Data given in the elements (multiple datasets who fit to con may be passed in a list)
- **con** (*np.ndarray of int*) – triangles: (N_elements. 3) tetrahedra: (N_elements, 4) Connectivity index list forming the elements
- **precise** (*bool, default: False*) – Compute data transformation precisely but slow. Better for near-0 values.

Returns
out – (N_nodes, N_data) Data in nodes

Return type
np.ndarray of float or list of np.ndarray

pynibs.mesh.transformations.**data_nodes2elements**(*data*, *con*)

Transforms data in nodes to elements (triangles or tetrahedra)

Parameters

- **data** (*np.ndarray of float*) – (N_nodes, N_data) Data given in the nodes
- **con** (*np.ndarray of int*) – triangles: (N_elements, 3) tetrahedra: (N_elements, 4) Connectivity index list forming the elements

Returns
out – (N_elements, N_data) Data given in the element scenters

Return type
np.ndarray of float

pynibs.mesh.transformations.**map_data_to_surface**(*datasets*, *points_datasets*, *con_datasets*, *fname_fsl_gm*, *fname_fsl_wm*, *fname_midlayer=None*, *delta=0.5*, *input_data_in_center=True*, *return_data_in_center=True*, *data_substitute=-1*)

Maps data from ROI of fsl surface (wm, gm, or midlayer) to given Freesurfer brain surface (wm, gm, inflated).

Parameters

- **datasets** (*np.ndarray of float [N_points x N_data] or list of np.ndarray*) – Data in nodes or center of triangles in ROI (specify this in "data_in_center")
- **points_datasets** (*np.ndarray of float [N_points x 3] or list of np.ndarray*) – Point coordinates (x,y,z) of ROI where data in datasets list is given, the points have to be a subset of the GM/WM surface (has to be provided for each dataset)
- **con_datasets** (*np.ndarray of int [N_tri x 3] or list of np.ndarray*) – Connectivity matrix of dataset points (has to be provided for each dataset)
- **fname_fsl_gm** (*str or list of str or list of None*) – Filename of pial surface fsl file(s) (one or two hemispheres) e.g. in mri2msh: .../fs_ID/surf/lh.pial
- **fname_fsl_wm** (*str or list of str or list of None*) – Filename of wm surface fsl file(s) (one or two hemispheres) e.g. in mri2msh: .../fs_ID/surf/lh.white

- **fname_midlayer** (*str or list of str*) – Filename of midlayer surface fsl file(s) (one or two hemispheres) e.g. in headreco: .../fs_ID/surf/lh.central
- **delta** (*float*) – Distance parameter where gm-wm surface was generated 0...1 (default: 0.5) 0 -> WM surface 1 -> GM surface
- **input_data_in_center** (*bool*) – Flag if data in datasets in given in triangle centers or in points (Default: True)
- **return_data_in_center** (*bool*) – Flag if data should be returned in nodes or in elements (Default: True)
- **data_substitute** (*float*) – Data substitute with this number for all points in the inflated brain, which do not belong to the given data set

Returns

data_mapped – Mapped data to target brain surface. In points or elements

Return type

np.ndarray of float [N_points_inf x N_data]

pynibs.mesh.transformations.**midlayer_2_surf**(*midlayer_data*, *coords_target*, *coords_midlayer*, *midlayer_con=None*, *midlayer_data_in_nodes=False*, *max_dist=5*, *outside_roi_val=0*, *precise_map=True*)

Convert midlayer data to whole-brain surface data, e.g. grey matter. Output is returned as data in nodes.

Parameters

- **midlayer_data** (*np.ndarray of float*) – (n_elm_midlayer,) or (n_nodes_midlayer,), the data in the midlayer.
- **coords_target** (*np.ndarray of float*) – (n_nodes_target, 3) Coordinates of the nodes of the target surface.
- **coords_midlayer** (*np.ndarray of float*) – (n_nodes_midlayer, 3) Coordinates of the nodes of the midlayer surface.
- **midlayer_con** (*np.ndarray of int, optional*) – (n_elm_midlayer, 3) Connectivity of the midlayer elements. Provide if data_in_points == True.
- **midlayer_data_in_nodes** (*bool, default=False*) – If midlayer data is provided in nodes, set to True and provide midlayer_con.
- **max_dist** (*float, default=5*) – Maximum distance between target and midlayer nodes to pull data from midlayer_data for.
- **outside_roi_val** (*float, default=0*) – Areas outside of max_dist are filled with outside_roi_val.
- **precise_map** (*bool, default=True*) – If elements to nodes mapping is done, perform this precise and slow or not.
- **Returns** –
- **--------** –
- **data_target** (*np.ndarray*) – (n_nodes_target, 1) The data in nodes of the target surface.

pynibs.mesh.transformations.**project_on_scalp**(*coords*, *mesh*, *scalp_tag=1005*)

Find the node in the scalp closest to each coordinate

Parameters

- **coords** (*nx3 np.ndarray*) – Vectors to be transformed
- **mesh** (*pynibs.TetrahedraLinear or simnibs.msh.mesh_io.Msh*) – Mesh structure in simnibs or pynibs format

- **scalp_tag** (*int, optional, default: 1005*) – Tag in the mesh where the scalp is to be set. Default: 1005

Returns
points_closest – (n, 3) coordinates projected scalp (closest skin points)

Return type
np.ndarry

pynibs.mesh.transformations.**project_on_scalp_hdf5**(*coords*, *mesh*, *scalp_tag=1005*)

Find the node in the scalp closest to each coordinate

Parameters

- **coords** (*nx3 np.ndarray*) – Vectors to be transformed
- **mesh** (*str or pynibs.TetrahedraLinear*) – Filename of mesh in .hdf5 format or Mesh structure
- **scalp_tag** (*int (optional)*) – Tag in the mesh where the scalp is to be set. Default: 1005

Returns
points_closest – coordinates projected scalp (closest skin points)

Return type
nx3 np.ndarray

pynibs.mesh.transformations.**refine_surface**(*fn_surf*, *fn_surf_refined*, *center*, *radius*, *repair=True*, *remesh=True*, *verbose=True*)

Refines surface (.stl) in spherical ROI an saves as .stl file.

Parameters

- **fn_surf** (*str*) – Input filename (.stl)
- **fn_surf_refined** (*str*) – Output filename (.stl)
- **center** (*np.ndarray of float (3)*) – Center of spherical ROI (x,y,z)
- **radius** (*float*) – Radius of ROI
- **repair** (*bool, optional, default: True*) – Repair surface mesh to ensure that it is watertight and forms a volume
- **remesh** (*bool, optional, default:False*) – Perform remeshing with meshfix (also removes possibly overlapping facets and intersections)
- **verbose** (*bool, optional, default: True*) – Print output messages

Returns
<file>

Return type
.stl file

pynibs.mesh.utils module

pynibs.mesh.utils.**calc_gradient_surface**(*phi*, *points*, *triangles*)

Calculate gradient of potential phi on surface (i.e. tangential component) given in vertices of a triangular mesh forming a 2D surface.

Parameters

- **phi** (*np.ndarray of float [N_points x 1]*) – Potential in nodes
- **points** (*np.ndarray of float [N_points x 3]*) – Coordinates of nodes (x,y,z)

- **triangles** (*np.ndarray of int32 [N_tri x 3]*) – Connectivity of triangular mesh

Returns
grad_phi – Gradient of potential phi on surface

Return type
np.ndarray of float [N_tri x 3]

pynibs.mesh.utils.**calc_tet_volume**(*points*, *abs=True*)

Calculate tetrahedra volumes.

Parameters

- **points** (*np.ndarray*) – shape: (n_tets,4,3) [[[Ax, Ay, Az],
  [Bx, By, Bz], [Cx, Cy, Cz], [Dx, Dy, Dy]],
  [[Ax, Ay, Az],
  [Bx, By, Bz], [Cx, Cy, Cz], [Dx, Dy, Dy]],
  …
  ]
- **abs** (*bool*) – Return magnitude. Default: True.

Returns
volume – shape: (n_tets)

Return type
np.ndarray

pynibs.mesh.utils.**calc_tetrahedra_volume_cross**(*P1*, *P2*, *P3*, *P4*)

Calculates volume of tetrahedra specified by the 4 points P1…P4 multiple tetrahedra can be defined by P1…P4 as 2-D np.arrays using the cross and vector dot product

$$P1 = \begin{bmatrix} x_{tet_1} & y_{tet_1} & z_{tet_1} \\ x_{tet_2} & y_{tet_2} & z_{tet_2} \\ \dots & \dots & \dots \\ x_{tet_N} & y_{tet_N} & z_{tet_N} \end{bmatrix}$$

Parameters

- **P1** (*np.ndarray of float [N_tet x 3]*) – Coordinates of first point of tetrahedra
- **P2** (*np.ndarray of float [N_tet x 3]*) – Coordinates of second point of tetrahedra
- **P3** (*np.ndarray of float [N_tet x 3]*) – Coordinates of third point of tetrahedra
- **P4** (*np.ndarray of float [N_tet x 3]*) – Coordinates of fourth point of tetrahedra

Returns
tetrahedra_volume – Volumes of tetrahedra

Return type
np.ndarray of float [N_tet x 1]

pynibs.mesh.utils.**calc_tetrahedra_volume_det**(*P1*, *P2*, *P3*, *P4*)

Calculate volume of tetrahedron specified by 4 points P1…P4 multiple tetrahedra can be defined by P1…P4 as 2-D np.arrays using the determinant.

$$P1 = \begin{bmatrix} x_{tet_1} & y_{tet_1} & z_{tet_1} \\ x_{tet_2} & y_{tet_2} & z_{tet_2} \\ \dots & \dots & \dots \\ x_{tet_N} & y_{tet_N} & z_{tet_N} \end{bmatrix}$$

Parameters

- **P1** (*np.ndarray of float [N_tet x 3]*) – Coordinates of first point of tetrahedra
- **P2** (*np.ndarray of float [N_tet x 3]*) – Coordinates of second point of tetrahedra
- **P3** (*np.ndarray of float [N_tet x 3]*) – Coordinates of third point of tetrahedra
- **P4** (*np.ndarray of float [N_tet x 3]*) – Coordinates of fourth point of tetrahedra

Returns
tetrahedra_volume – Volumes of tetrahedra

Return type
np.ndarray of float [N_tet x 1]

pynibs.mesh.utils.**calc_tri_surface**(*points*)

Calculate triangle surface areas.

Parameters
points (*np.ndarray*) – (n_triangles,3,3)

Returns
triange_area

Return type
np.ndarray

pynibs.mesh.utils.**check_islands_for_single_elm**(*source_elm*, *connectivity=None*, *adjacency=None*, *island_crit=1*)

This identifies islands in a mesh for a given element. An island is a set of elements, that is only connect via a single node to another set of elements. These islands usually crash the FEM solver and should be removed.

1. Find all elements connect to source_elm via one node (1-node-neighbor)
2. Start with source_elm and visit all 2-node-neighbors ('shared-edge)
3. Continue recursively with all 2-node-neighbors and visit their 2-node-neighbors
4. See if any 1-node-neighbors have not been visited with this strategy. If so, an island has been found

Parameters

- **source_elm** (*int*) – The source element to check
- **connectivity** (*np.ndarray, optional*) – Connectivity ('node_number_list') starting with 0. Can be triangles or tetrahedra (n_elms, 3) or (n_elms_4).
- **adjacency** (*np.ndparray, optional*) – Adjenceny matrix (n_elm, n_elm). Weights are supposed to be number of shared nodes. Computed from neighbors if not provided.
- **island_crit** (*int, optional, default: 'any'*) – How many nodes to define islands? 'any' -> Elements connected via a single node or single edge are defined as an island. 'node' -> Elements connected via a single _node_ are defined as an island. 'edge' -> Elements connected via a single _edge_ are defined as an island.

Returns

- **n_visited** (*int*)
- **n_not_visited** (*int*)
- **neighbors_visited** (*dict, which neighbors have been visited and which have not*)

pynibs.mesh.utils.**determine_e_midlayer**(*fn_e_results*, *fn_mesh_hdf5*, *subject*, *mesh_idx*, *roi_idx*, *n_cpu=4*, *midlayer_fun='simnibs'*, *phi_scaling=1.0*, *verbose=False*)

Parallel version to determine the midlayer e-fields from a list of .hdf5 results files

Parameters

- **fn_e_results** (*list of str*) – List of results filenames (.hdf5 format)
- **fn_mesh_hdf5** (*str*) – Filename of corresponding mesh file
- **subject** (*pynibs.Subject*) – Subject object
- **mesh_idx** (*int*) – Mesh index
- **roi_idx** (*int*) – ROI index
- **n_cpu** (*int, optional, default: 4*) – Number of parallel computations
- **midlayer_fun** (*str, optional, default: "simnibs"*) – Method to determine the midlayer e-fields ("pynibs" or "simnibs")
- **phi_scaling** (*float, optional, default: 1.0*) – Scaling factor of scalar potential to change between "m" and "mm"

Returns
Adds midlayer e-field results to ROI

Return type
<File> .hdf5 file

pynibs.mesh.utils.**determine_e_midlayer_workhorse**(*fn_e_results*, *subject*, *mesh_idx*, *midlayer_fun*, *fn_mesh_hdf5*, *roi_idx*, *phi_scaling=1.0*, *verbose=False*)

phi_scaling: float
simnibs < 3.0 : 1000. simnibs >= 3.0 : 1. (Default)

pynibs.mesh.utils.**find_element_idx_by_points**(*nodes*, *con*, *points*)

Finds the tetrahedral element index of an arbitrary point in the FEM mesh.

Parameters

- **nodes** (*np.ndarray [N_nodes x 3]*) – Coordinates (x, y, z) of the nodes
- **con** (*np.ndarray [N_tet x 4]*) – Connectivity matrix
- **points** (*np.ndarray [N_points x 3]*) – Points for which the element indices are found.

Returns
ele_idx – Element indices of tetrahedra where corresponding 'points' are lying in

Return type
np.ndarray [N_points]

pynibs.mesh.utils.**find_island_elms**(*connectivity=None*, *adjacency=None*, *verbose=False*, *island_crit='edge'*, *decision='cumulative'*)

Searches for islands in a mesh and returns element indices of the smallest island. Island is defines as a set of elements, which share a single node and/or single edge with the rest of the mesh.

Parameters

- **connectivity** (*np.ndarray, optional*) – Connectivity ('node_number_list') starting with 0. Can be triangles or tetrahedra (n_elms, 3) or (n_elms_4).
- **adjacency** (*np.ndparray, optional*) – Adjenceny matrix (n_elm, n_elm). Weights are supposed to be number of shared nodes. Computed from neighbors if not provided.

- **island_crit** (*int, optional, default: 'edge'*) – How many nodes to define islands? 'node' -> Elements connected via a single _node_ are defined as an island. 'edge' -> Elements connected via a single _edge_ are defined as an island.
- **decision** (*string, optional, default: cumulative*) – 'cumulative' -> Return all element indices that are not visited any times 'smallest' -> Return smallest island.
- **verbose** (*bool, optional*) – Print some verbosity information. Default: False

Returns
island

Return type
list of island-elms

pynibs.mesh.utils.**find_islands**(*connectivity=None*, *adjacency=None*, *island_crit='any'*, *verbose=False*, *largest=False*)

This identifies islands in a mesh. An island is a set of elements, that is only connect via a single node to another set of elements. These islands usually crash the FEM solver and should be removed.

For each element:

1. Find all elements connect to source_elm via one node (1-node-neighbor)
2. Start with source_elm and visit all 2-node-neighbors ('shared-edge)
3. Continue recursively with all 2-node-neighbors and visit their 2-node-neighbors
4. See if any 1-node-neighbors have not been visited with this strategy. If so, an island has been found

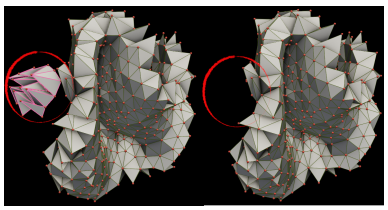

Fig. 1: Islands are groups of elements that are only connected via a single node/edge to another group.

Parameters

- **connectivity** (*np.ndarray, optional*) – Connectivity ('node_number_list') starting with 0. Can be triangles or tetrahedra (n_elms, 3) or (n_elms_4).
- **adjacency** (*np.ndparray, optional*) – Adjenceny matrix (n_elm, n_elm). Weights are supposed to be number of shared nodes. Computed from neighbors if not provided.
- **island_crit** (*int, optional, default: 'any'*) – How many nodes to define islands? 'any' -> Elements connected via a single node or single edge are defined as an island. 'node' -> Elements connected via a single _node_ are defined as an island. 'edge' -> Elements connected via a single _edge_ are defined as an island.
- **largest** (*book, default: False*) – Only return largest island, speeds up computation quite a bit if only one large, and many small islands exist.
- **verbose** (*bool, optional*) – Print some verbosity information. Default: False

Returns

- **elms_with_island** (*list*) – Elements with neighboring islands
- **counter_visited** (*np.ndarray*) – shape = (n_elms). How often as each element been visited.
- **counter_not_visited** (*np.ndarray*) – shape = (n_elms). How often as each element not been visited.

pynibs.mesh.utils.**find_nearest**(*array*, *value*)

Given an "array", and given a "value" , returns an index j such that "value" is between array[j] and array[j+1]. "array" must be monotonic increasing. j=-1 or j=len(array) is returned to indicate that "value" is out of range below and above respectively.

Parameters

- **array** (*np.ndarray of float*) – Monotonic increasing array
- **value** (*float*) – Target value the nearest neighbor index in "array" is computed for

Returns

idx – Index j such that "value" is between array[j] and array[j+1]

Return type

int

pynibs.mesh.utils.**get_indices_discontinuous_data**(*data*, *con*, *neighbor=False*, *deviation_factor=2*, *min_val=None*, *not_fitted_elms=None*, *crit='median'*, *neigh_style='point'*)

Get element indices (and the best neighbor index), where the data is discontinuous

Parameters

- **data** (*np.ndarray of float [n_data]*) – Data array to analyze given in the element center
- **con** (*np.ndarray of float [n_data, 3 or 4]*) – Connectivity matrix
- **neighbor** (*boolean, optional, default=False*) – Return also the element index of the "best" neighbor (w.r.t. median of data)
- **deviation_factor** (*float*) – Allows data deviation from 1/deviation_factor < data[i]/median < deviation_factor
- **min_val** (*float, optional*) – If given, only return elements which have a neighbor with data higher than min_val.
- **not_fitted_elms** (*np.ndarray*) – If given, these elements are not used as neighbors
- **crit** (*str, default: median*) – Criterium for best neighbor. Either median or max value
- **neigh_style** (*str, default: 'point'*) – Should neighbors share point or 'edge'

Returns

- **idx_disc** (*list of int [n_disc]*) – Index list containing the indices of the discontinuous elements
- **idx_neighbor** (*list of int [n_disc]*) – Index list containing the indices of the "best" neighbors of the discontinuous elements

pynibs.mesh.utils.**get_sphere**(*mesh=None*, *mesh_fn=None*, *target=None*, *radius=None*, *roi_idx=None*, *roi=None*, *elmtype='tris'*)

Return element idx of elements within a certain distance to provided target. Element indices are 0-based (tris and tets start at 0, 'pynibs' style) Elements might be 'tris' (default) or 'tets'

If roi object / idx and mesh fn is provided, the roi is expected to have midlayer information and the roi geometry is used.

Parameters

- **mesh** (*pynibs.mesh.TetrahedraLinear, optional*) –
- **mesh_fn** (*str, optional*) – Filename to SimNIBS .msh or pyNIBS .hdf5 mesh file
- **target** (*np.ndarray of float, optional*) – (3,) X, Y, Z coordinates of target
- **radius** (*float, optional*) – Sphere radius im mm

- **roi_idx** (*str or int*) – ROI name
- **elmtype** (*str, optional*) – Return triangles or tetrahedra in sphere around target. One of ('tris', 'tets')

Returns
elms_in_sphere – (n_elements): Indices of elements found in ROI

Return type
np.ndarray

pynibs.mesh.utils.**in_hull**(*points*, *hull*)

Test if points in *points* are in *hull*. *points* should be a [N x K] coordinates of N points in K dimensions. *hull* is either a scipy.spatial.Delaunay object or the [M x K] array of the coordinates of M points in Kdimensions for which Delaunay triangulation will be computed.

Parameters

- **points** (*np.ndarray*) – (N_points x 3) Set of floating point data to test whether they are lying inside the hull or not
- **hull** (*scipy.spatial.Delaunay or np.ndarray*) – (M x K) Surface data

Returns
inside – TRUE: point inside the hull FALSE: point outside the hull

Return type
np.ndarray of bool

pynibs.mesh.utils.**sample_sphere**(*n_points*, *r*)

Creates n_points evenly spread in a sphere of radius r.

Parameters

- **n_points** (*int*) – Number of points to be spread, must be odd
- **r** (*float*) – Radius of sphere

Returns
points – Evenly spread points in a unit sphere

Return type
np.ndarray of float [N x 3]

pynibs.mesh.utils.**tets_in_sphere**(*mesh*, *target*, *radius*, *roi*)

Worker function for get_sphere()

Returns element idx of elements within a certain distance to provided target. If roi object / idx and mesh fn is provided, the roi is expected to have midlayer information and the roi geometry is used.

Parameters

- **mesh** (*pynibs.TetrahedraLinear, optional*) –
- **target** (*np.ndarray of float, optional*) – (3,) X, Y, Z coordinates of target
- **radius** (*float, optional*) – Sphere radius im mm
- **roi** (*pynibs.mesh.ROI, optional*) – Region of interest

Returns
tets_in_sphere – (n_tets): Indices of elements found in ROI

Return type
np.ndarray

pynibs.mesh.utils.**tris_in_sphere**(*mesh*, *target*, *radius*, *roi*)

Worker function for get_sphere().

Returns triangle idx of elements within a certain distance to provided target. If roi object / idx and mesh fn is provided, the roi is expected to have midlayer information and the roi geometry is used.

Parameters

- **mesh** (*pynibs.mesh.TetrahedraLinear, optional*) –
- **target** (*np.ndarray of float, optional*) – (3,) X, Y, Z coordinates of target
- **radius** (*float, optional*) – Sphere radius im mm
- **roi** (*pynibs.mesh.ROI, optional*) – ROI

Returns
tris_in_sphere – (n_triangles): Indices of elements found in sphere

Return type
np.ndarray

pynibs.models package

pynibs.neuron package

Submodules

pynibs.neuron.neuron_regression module

pynibs.neuron.neuron_regression.**calc_e_effective**(*e*, *layerid*, *theta*, *gradient=None*, *neuronmodel='threshold'*, *mep=None*, *waveform='biphasic'*)

Determines the effective electric field using a neuron mean field model. Transforms the electric field by subtracting the threshold map (in V/m) from the original electric field. The remaining field is the effective electric field (e_eff), generating the stimulation effect, i.e. behavioural effects start at e_eff > 0 because lower fields were not able to stimulate neurons.

Parameters

- **e** (*np.ndarray*) – Electric field (matrix) [N_stim x N_ele]
- **layerid** (*str*) – Choose from the neocortical layers (e.g. "L1", "L23", "L4", "L5", "L6")
- **theta** (*np.ndarray*) – Theta angle (matrix) [N_stim x N_ele] of electric field with respect to surface normal
- **gradient** (*np.ndarray, optional, default: None*) – Electric field gradient (matrix) [N_stim x N_ele] between layer 1 and layer 6. Optional, the neuron mean field model is more accurate when provided.
- **neuronmodel** (*str, optional, default: "threshold"*) – Select neuron model to modify the electric field values - "threshold": subtract mean threshold from electric field - "IOcurve": subtract value read from precomputed neuron IO curve from electric field
- **mep** (*np.array of float [N_stim], optional, default: None*) – MEP data (required in case of "IOcurve" approach (neuronmodel)
- **waveform** (*str, optional, default: biphasic*) – Waveform of TMS pulse: - "monophasic" - "biphasic"

Returns
e_eff – Effective electric field (matrix) [N_stim x N_ele] the regression analysis can be performed with.

Return type
np.ndarray

pynibs.neuron.neuron_regression.**calc_e_threshold**(*layerid*, *theta*, *gradient=None*, *mep=None*, *neuronmodel='threshold'*, *waveform='biphasic'*)

Determine sensitivity map of electric field

Parameters

- **layerid** (*str*) – Choose from the neocortical layers (e.g. "L1", "L23", "L4", "L5", "L6"). The respective threshold model will be loaded.
- **theta** (*np.ndarray*) – Theta angle (matrix) [N_stim x N_ele] of electric field with respect to surface normal. in degrees [0 .. 180]
- **gradient** (*np.ndarray, optional, default: None*) – Electric field gradient (matrix) [N_stim x N_ele] between layer 1 and layer 6. Optional, the neuron mean field model is more accurate when provided. Percent [-100 .. 100]
- **mep** (*np.array of float [N_stim], optional, default: None*) – MEP data (required in case of "IOcurve" approach (neuronmodel)
- **neuronmodel** (*str, optional, default: "threshold"*) – Select neuron model to modify the electric field values - "threshold": subtract mean threshold from electric field - "IOcurve": subtract value read from precomputed neuron IO curve from electric field
- **waveform** (*str, optional, default: biphasic*) – Waveform of TMS pulse: - "monophasic" - "biphasic"

Returns
e_sens – Electric field sensitivity maps [N_stim x N_ele]

Return type
np.ndarray

pynibs.neuron.neuron_regression.**load_cell_model**(*fn_csv*)

Load interpolation points of the mean field model from the specified CSV file.

Parameters
fn_csv (*str*) – Fully qualified path to the CSV containing the interpolation points of the mean field model.

Returns

- *- scipy.interpolate.LinearNDInterpolator*
- *- interpolation points 'theta'*
- *- interpolation points 'gradient'*

pynibs.neuron.neuron_regression.**workhorse_interp**(*idx_list*, *interp*, *params*)

Single core workhorse to interpolate data.

Parameters

- **idx_list** (*np.array or list of float [n_interpolations]*) – Indices in params array where the interpolation has to be performed (subset of all indices in params array)
- **interp** (*scipy.interpolate instance*) – Interpolator instance
- **params** (*np.array of float [N_interpolations x N_params]*) – Array containing the parameters the function is evaluated (total array with all parameters)

Returns
res – Interpolation results (subset params[idx_list, :])

Return type
np.array of float [n_interpolations]

pynibs.neuron.util module

pynibs.neuron.util.**DI_wave**(*t*, *intensity*, *t0=5*, *dt=1.4*, *width=0.25*)
Determines cortical DI waves from TMS

Parameters

- **t** (*ndarray of float [n_t]*) – Time axis in ms
- **intensity** (*float*) – Stimulator intensity w.r.t resting motor threshold (typical range: [0 … 2])
- **t0** (*float*) – offset time
- **dt** (*float*) – Spacing of waves in ms
- **width** (*float*) – Width of waves

Returns
y – DI waves

Return type
ndarray of float [n_t]

pynibs.pckg package

Subpackages

pynibs.pckg.libeep package

Submodules

pynibs.pckg.libeep.pyeep module

pynibs.pckg.libeep.pyeep.**add_channel**()
add channel to channel info handle

pynibs.pckg.libeep.pyeep.**add_samples**()
add samples

pynibs.pckg.libeep.pyeep.**close**()
close handle

pynibs.pckg.libeep.pyeep.**close_channel_info**()
close channel info handle

pynibs.pckg.libeep.pyeep.**create_channel_info**()
create channel info handle

pynibs.pckg.libeep.pyeep.**get_channel_count**()
get channel count

pynibs.pckg.libeep.pyeep.**get_channel_label**()
get channel label

pynibs.pckg.libeep.pyeep.**get_channel_reference**()
get channel reference

pynibs.pckg.libeep.pyeep.**get_channel_unit**()
get channel unit

pynibs.pckg.libeep.pyeep.**get_sample_count**()
get sample count

pynibs.pckg.libeep.pyeep.**get_sample_frequency**()
get sample frequency

pynibs.pckg.libeep.pyeep.**get_samples**()
get samples

pynibs.pckg.libeep.pyeep.**get_trigger**()
get samples

pynibs.pckg.libeep.pyeep.**get_trigger_count**()
get trigger count

pynibs.pckg.libeep.pyeep.**get_version**()
get libeep version

pynibs.pckg.libeep.pyeep.**read**()
open libeep file for reading

pynibs.pckg.libeep.pyeep.**write_cnt**()
open libeep cnt file for writing

pynibs.regression package

This holds methods for the TMS-based cortical localization approach as published in [1]

References

Submodules

pynibs.regression.regression module

class pynibs.regression.regression.**Element**(*x*, *y*, *ele_id*, *fun=<function sigmoid4>*, *score_type='R2'*, *select_signed_data=False*, *constants=None*, ***kwargs*)

Bases: object

Fit Element object class

calc_score()
Determine goodness-of-fit score

run_fit(*max_nfev=1000*)
Perform data fit with lmfit

run_select_signed_data()
Selects positive or negative data by performing an initial linear fit by comparing the resulting p-values, slopes and R2 values. Either positive or negative data (w.r.t. x-axis) yielding a fit with a p-value < 0.05, a positive slope and the higher R2 value is used and the remaining data with the other sign is omitted from the analysis

set_constants(*value*)

Sets constants in self.constants and gmodel instance

set_init_vals(*value*)

Sets initial values in self.init_vals and gmodel instance

set_limits(*value*)

Sets limits in self.limits and gmodel instance

Parameters
value (*dict*) – Parameters (keys) to set in self as limits.

set_random_init_vals()

Set random initial values

setup_model()

Setup model parameters (limits, initial values, etc. ...)

pynibs.regression.regression.**fit_elms**(*elm_idx_list*, *e_matrix*, *mep*, *zap_idx=None*, *fun=<function sigmoid4>*, *init_vals=None*, *limits=None*, *log_scale=False*, *constants=None*, *max_nfev=None*, *bad_elm_idx=None*, *score_type='R2'*, *verbose=False*)

Workhorse for Mass-univariate nonlinear regressions on raw MEP_{AMP} ~ E. That is, for each element in elm_idx_list, it's E (mag | norm | tan) for each zap regressed on the raw MEP amplitude. An element wise r2 score is returned.

Parameters

- **elm_idx_list** (*list of int or np.ndarray*) – List containing the element indices the fit is performed for
- **e_matrix** (*np.ndarray of float [n_zaps x n_ele]*) – Electric field matrix
- **mep** (*np.ndarray of float [n_zaps]*) – Motor evoked potentials for every stimulation
- **zap_idx** (*np.ndarray [n_zaps], optional, default: None*) – Indices of zaps the congruence factor is calculated with (default: all)
- **fun** (*str*) – A function name of pynibs.exp.Mep (exp0, sigmoid)
- **init_vals** (*np.ndarray of dict*) – Dictionary containing the initial values for each element as np.ndarray [len(elm_idx_list)]. The keys are the free parameters of fun, e.g. "x0", "amp", etc
- **limits** (*pd.DataFrame*) – Dictionary containing the limits of each parameter for each element e.g.: limits["x0"][elm_idx] = [min, max]
- **log_scale** (*bool, optional, default: False*) – Log-transform data before fit (necessary for functions defined in the log domain)
- **constants** (*dict of <string>:<num>, optional, default: None*) – key:value pair of model parameters not to optimize.
- **max_nfev** (*int, default: None*) – Max fits, passed to model.fit() as max_nfev=max_nfev*len(x).
- **bad_elm_idx** (*np.ndarray*) – Indices of elements not to fit, with indices corresponding to indices (not values) of elm_idx_list
- **score_type** (*str, optional, default: "R2"*) – Goodness of fit measure; Choose SR for nonlinear fits and R2 or SR for linear fits: - "R2": R2 score (Model variance / Total variance) [0, 1] for linear models; 0: bad fit; 1: perfect fit - "SR": Relative standard error of regression (1 - Error 2-norm / Data 2-norm) [-inf, 1]; 1: perfect fit

- **mask_e_field** (*np.ndarray of bool [n_zaps x n_ele], optional, default: None*) – Mask indicating for which e-field (and mep) values the fit is performed for. Changes for normal component in each element because of the sign and p-values. If None, all data is used in each element.
- **verbose** (*bool, optional, default: False*) – Print verbosity information

Returns

- **r2** (*np.ndarray of float [n_roi, 1]*) – R2 for each element in elm_idx_list
- **best_values** (*np.ndarray of object*) – Fit parameters returned from the optimizer

pynibs.regression.regression.**get_bad_elms**(*x*, *y*, *method='lstsq'*, *verbose=False*)

This does an element-wise fast linear regression fit to identify bad elements. Bad is defined here as a negative slope.

x
[np.ndarray of float [n_zaps x n_ele]] Electric field matrix

y
[np.ndarray of float [n_zaps]] Motor evoked potentials for every stimulation

method
[str, default: "lstsq"] Which method to use. (numpy.linalg.)lstsq, (scipy.stats.)linregress, or pinv

verbose
[bool, optional, default: False] Print verbosity information

Returns

idx: np.ndarray
Indices of bad elements

pynibs.regression.regression.**get_model_init_values**(*fun*, *elm_idx_list*, *e_matrix*, *mep*, *mask_e_field=None*, *rem_empty_hints=True*)

Calc appropriate init, limit, and max values for models fits depending on the data. If negative and positive x-data is present in case of e.g. normal component values are set according to the side (positive or negative) where more values are present. When more positive x-axis values are present, negative x-axis values will be ignored. When more negative x-axis values are present, the absolute values will be taken and the positive values are ignored.

Only parameters for sigmoid* are optimized.

Parameters

- **fun** (*pyfempp.exp.Mep*) – IO curve function object
- **elm_idx_list** (*np.ndarray of int*) – Array containing the element indices the fit is performed for
- **e_matrix** (*np.ndarray of float [n_zaps x n_ele]*) – Electric field matrix
- **mep** (*np.ndarray of float [n_zaps]*) – Motor evoked potentials for every stimulation
- **mask_e_field** (*np.ndarray of bool [n_zaps x n_ele], optional, default: None*) – Mask indicating for which e-field (and mep) values the fit is performed for. Changes for normal component in each element because of the sign and p-values. If None, all data is used in each element.
- **rem_empty_hints** (*bool, default: True*) – Remove any non-filled param hints from limits dict.

Returns

- **log_scale** (*bool*) – Log scale

- **limits** (*dict of list [n elm_index_list]*) – Element-wise limit values for function fitting.
- **init_vals** (*dict of list [n elm_index_list]*) – Element-wise init values for function fitting.
- **max_vals_refit** (*dict of list [n elm_index_list]*) – Element-wise perturbation range for refitting function.

pynibs.regression.regression.**init**(*l*, *zap_lists*, *res_fn*)

Pool init function to use with regression_nl_hdf5_single_core_write

Parameters

- **l** (*multiprocessing.Lock()*) –
- **zap_lists** (*list of list of int*) – Which zaps to compute.
- **res_fn** (*str*) – .hdf5 fn

pynibs.regression.regression.**logistic_regression**()

Some ideas on how to improve regression approach

1. De-log data

Data range has to be transformed to a reasonable range. For a full sigmoid, -10:10 looks ok

sig <- function(z) {
return(1 / (1 + exp(-z)))

}

desig <- function(x) {
return(- log((1/x) - 1))

}

This might be a reasonable fast approach, but the parameter range has to be estimated. Maybe remove some outliters?

2. fit logistic regression to raw data scipy.optimize provides fit_curve(), which does OLS-ish fitting to a given function https://stackoverflow.com/questions/54376900/fit-sigmoid-curve-in-python

I expect this to be rather slow.

3. Use the sklearn logistic_regression classifyer and access raw fit data The logistic_regression is implemented as a classifyer, maybe it's possible to use its regression fit results. Implementation should be pretty fast.

pynibs.regression.regression.**nl_hdf5**(*elm_idx_list=None*, *fn_reg_hdf5=None*, *qoi_path_hdf5=None*, *e_matrix=None*, *mep=None*, *fun=<function sigmoid4>*, *zap_idx=None*, *n_cpu=4*, *con=None*, *n_refit=50*, *return_fits=False*, *score_type='R2'*, *verbose=False*, *pool=None*, *refit_discontinuities=True*)

Mass-univariate nonlinear regressions on raw MEP_{AMP} ~ E. That is, for each element in elm_idx_list, it's E (mag | norm | tan) for each zap regressed on the raw MEP amplitude. An element wise r2 score is returned. The function reads the precomputed array of E-MEP data from an .hdf5 file.

Parameters

- **elm_idx_list** (*np.ndarray of int*) – List containing the element indices the fit is performed for
- **fn_reg_hdf5** (*str*) – Filename (incl. path) containing the precomputed E-MEP dataframes
- **qoi_path_hdf5** (*Union[str, list[str]]*) – Path in .hdf5 file to dataset of electric field qoi e.g.: ["E", "E_norm", "E_tan"]
- **e_matrix** (*np.ndarray of float [n_zaps x n_ele]*) – Electric field matrix

- **mep** (*np.ndarray of float [n_zaps]*) – Motor evoked potentials for every stimulation
- **zap_idx** (*np.array [n_zaps], optional, default: None*) – Indices of zaps the congruence factor is calculated with (default: all)
- **fun** (*pynibs.exp.Mep function*) – A function of pynibs.exp.Mep (exp0, sigmoid)
- **n_cpu** (*int*) – Number of threads
- **con** (*np.ndarray of float [n_roi, 3 or 4], optional, default: None*) – Connectivity matrix of ROI (needed in case of refit because of discontinuity check)
- **n_refit** (*int, optional, default: 50*) – Maximum number of refits of zero elements. No refit is applied in case of n_refit = 0.
- **return_fits** (*bool, optional, default: False*) – Return fit objects containing the parameter estimates
- **score_type** (*str, optional, default: "R2"*) – Error measure of fit: - "R2": R2 score (Model variance / Total variance); linear fits: [0, 1], 1 … perfect fit - "SR": Relative standard error of regression (1 - Error 2-norm / Data 2-norm); [-Inf, 1], 1 … perfect fit
- **verbose** (*bool, optional, default: False*) – Plot output messages
- **pool** (*multiprocessing.Pool()*) – pool instance to use.
- **refit_discontinuities** (*bool, optional, default: True*) – Run refit for discontinuous elements at the end

Returns
r2 – R2 for each element in elm_idx_list

Return type
np.ndarray of float [n_roi, n_qoi]

pynibs.regression.regression.**nl_hdf5_single_core**(*zap_idx*, *elm_idx_list*, *fn_reg_hdf5=None*, *qoi_path_hdf5=None*, *e_matrix=None*, *mep=None*, *fun=<function sigmoid4>*, *con=None*, *n_refit=50*, *return_fits=False*, *constants=None*, *verbose=False*, *seed=None*, *rem_bad_elms=True*, *return_e_field_stats=True*)

Mass-univariate nonlinear regressions on raw MEP_{AMP} ~ E. That is, for each element in elm_idx_list, it's E (mag | norm | tan) for each zap regressed on the raw MEP amplitude. An element wise r2 score is returned. The function reads the precomputed array of E-MEP data from an .hdf5 file.

Parameters

- **elm_idx_list** (*np.ndarray of int*) – List containing the element indices the fit is performed for
- **fn_reg_hdf5** (*str*) – Filename (incl. path) containing the precomputed E-MEP dataframes
- **qoi_path_hdf5** (*str or list of str*) – Path in .hdf5 file to dataset of electric field qoi e.g.: ["E", "E_norm", "E_tan"]
- **e_matrix** (*np.ndarray of float [n_zaps x n_ele]*) – Electric field matrix
- **mep** (*np.ndarray of float [n_zaps]*) – Motor evoked potentials for every stimulation
- **zap_idx** (*np.array [n_zaps], optional, default: None*) – Indices of zaps the congruence factor is calculated with (default: all)

- **fun** (*function object*) – A function of pynibs.exp.Mep (exp0, sigmoid)
- **con** (*np.ndarray of float [n_roi, 3 or 4], optional, default: None*) – Connectivity matrix of ROI (needed in case of refit because of discontinuity check)
- **n_refit** (*int, optional, default: 50*) – Maximum number of refits of zero elements. No refit is applied in case of n_refit = 0.
- **return_fits** (*bool, optional, default: False*) – Return fit objects containing the parameter estimates
- **constants** (*dict of <string>:<num>, optional, default: None*) – key:value pair of model parameters not to optimize.
- **verbose** (*bool, optional, default: False*) – Plot output messages
- **seed** (*int, optional, default: None*) – Seed to use.
- **rem_bad_elms** (*bool, default: True*) – Remove elements based on a fast linear regression slope estimation
- **return_e_field_stats** (*bool, default: True*) – Return some stats on the efield variance

Returns

r2 – R2 for each element in elm_idx_list

Return type

np.ndarray of float [n_roi, n_qoi]

pynibs.regression.regression.**nl_hdf5_single_core_write**(*i*, *elm_idx_list*, *fn_reg_hdf5=None*, *qoi_path_hdf5=None*, *e_matrix=None*, *mep=None*, *fun=<function sigmoid4>*, *con=None*, *n_refit=50*, *return_fits=False*, *constants=None*, *verbose=False*, *seed=None*, *stepdown=False*, *score_type='R2'*, *return_progress=False*, *geo=None*)

pynibs.regression.regression.**regress_data**(*e_matrix*, *mep*, *elm_idx_list=None*, *element_list=None*, *fun=<function sigmoid4>*, *n_cpu=4*, *con=None*, *n_refit=50*, *zap_idx=None*, *return_fits=False*, *score_type='R2'*, *verbose=False*, *pool=None*, *refit_discontinuities=True*, *select_signed_data=False*, ***kwargs*)

Mass-univariate nonlinear regressions on raw MEP_{AMP} ~ E. That is, for each element in elm_idx_list, it's E (mag | norm | tan) for each zap regressed on the raw MEP amplitude. An element wise R2 score is returned. The function reads the precomputed array of E-MEP data from an .hdf5 file.

Parameters

- **e_matrix** (*np.ndarray of float*) – (n_zaps, n_ele) Electric field matrix
- **mep** (*np.ndarray of float*) – (n_zaps,) Motor evoked potential for each stimulation
- **elm_idx_list** (*np.ndarray of int or list of int*) – (n_zaps,) List containing the element indices the fit is performed for
- **element_list** (*list of Element object instances, optional*) – [n_ele] pynibs.Element objects ot skip initialization here.
- **fun** (*pynibs.exp.Mep, default: pynibs.sigmoid4*) – A pynibs.exp.Mep function (exp0, sigmoid, sigmoid4, ...)
- **n_cpu** (*int*) – Number of threads, if n_cpu=1 no parallel pool will be opened and all calculations are done in serial

- **con** (*np.ndarray of float, optional*) – (n_ele, 3 or 4) Connectivity matrix of ROI. Needed in case of refit for discontinuity checks
- **n_refit** (*int, default: 50*) – Maximum number of refits of zero elements. No refit is applied in case of n_refit = 0.
- **zap_idx** (*np.ndarray of int or list of int, optional*) – Which e/mep pairs to use.
- **return_fits** (*bool, optional*) – Return fit objects containing the parameter estimates
- **score_type** (*str, default: "R2"*) – Error measure of fit: - "R2": R2 score (Model variance / Total variance); linear fits: [0, 1], 1 … perfect fit - "SR": Relative standard error of regression (1 - Error 2-norm / Data 2-norm); [-Inf, 1], 1 … perfect fit - "rho": Spearman correlation coefficient [-1, 1]; finds any monotonous correlation (0 means no correlation)
- **verbose** (*bool, default: False*) – Plot output messages
- **pool** (*multiprocessing.Pool()*) – pool instance to use
- **refit_discontinuities** (*bool, default: True*) – Refit discontinuous elements. If True, provide _con_
- ****kwargs** – Passed on to pynibs.Element() to set fit parameters.

Returns

- **score** (*np.ndarray of float (n_roi, n_qoi)*) – Score for each element
- **best_values** (*list of dict (n_ele)*) – List of parameter fits

pynibs.regression.regression.**ridge_from_hdf5**(*elm_idx_list*, *fn_reg_hdf5*, *qoi_path_hdf5*, *zap_idx=None*)

Mass-univariate ridge regressions on raw MEP_{AMP} ~ E. That is, for each element in elm_idx_list, it's E (mag | norm | tan) for each zap regressed on the raw MEP amplitude. An element wise sklearn.metrics.regression.r2_score is returned. The function reads the precomputed array of E-MEP data from an .hdf5 file. Always uses all cores of a machine!

elm_idx_list
[list of int] List containing the element indices the fit is performed for

fn_hdf5
[str] Filename (incl. path) containing the precomputed E-MEP dataframes

qoi_path_hdf5
[str] Path in .hdf5 file to dataset of electric field qoi

zap_idx
[np.ndarray [n_zaps], optional, default: None] Indices of zaps the congruence factor is calculated with (default: all)

Returns
r2 – R^2 for each element in elm_idx_list

Return type
nparray of float [n_roi, n_datasets]

pynibs.regression.regression.**sing_elm_fitted**(*elm_idx_list*, *mep_lst*, *mep_params*, *e*, *alpha=1000*, *n_samples=100*)

Mass-univariate ridge regressions on fitted MEP_{AMP} ~ E. That is, for each element in elm_idx_list, it's E (mag | norm | tan) for each zap regressed on the raw MEP amplitude. An element wise sklearn.metrics.regression.r2_score is returned.

elm_idx_list: nparray [chunksize]
List of element indices, the congruence factor is computed for

mep_lst: list of Mep object instances [n_cond]
List of fitted Mep object instances for all conditions (see exp.py for more information of Mep class)

mep_params: nparray of float [n_mep_params_total]
List of all mep parameters of curve fits used to calculate the MEP (accumulated into one array) (e.g.: [mep_#1_para_#1, mep_#1_para_#2, mep_#1_para_#3, mep_#2_para_#1, mep_#2_para_#1, ...])

e: nparray of float with e.shape == (n_elm, n_cond, n_qoi)
array of the electric field to compute the r2 factor for, e.g. (e_mag, e_norm, e_tan)

n_samples
[int, default=100] Number of data points to generate discrete mep and e curves

Returns
r2 – R^2 for each element in elm_idx_list

Return type
nparray of float [n_roi, n_datasets]

pynibs.regression.regression.**sing_elm_raw**(*elm_idx_list*, *mep_lst*, *mep_params*, *e*, *alpha=1000*)

Mass-univariate ridge regressions on raw MEP_{AMP} ~ E. That is, for each element in elm_idx_list, it's E (mag | norm | tan) for each zap regressed on the raw MEP amplitude. An element wise sklearn.metrics.regression.r2_score is returned.

elm_idx_list: nparray [chunksize]
List of element indices, the congruence factor is computed for

mep: list of Mep object instances [n_cond]
List of fitted Mep object instances for all conditions (see exp.py for more information of Mep class)

mep_params: nparray of float [n_mep_params_total]
List of all mep parameters of curve fits used to calculate the MEP (accumulated into one array) (e.g.: [mep_#1_para_#1, mep_#1_para_#2, mep_#1_para_#3, mep_#2_para_#1, mep_#2_para_#1, ...])

e: nparray of float with e.shape == (n_elm, n_cond, n_qoi)
array of the electric field to compute the r2 factor for, e.g. (e_mag, e_norm, e_tan)

Returns
r2 – R^2 for each element in elm_idx_list

Return type
nparray of float [n_roi, n_datasets]

pynibs.regression.regression.**single_fit**(*x*, *y*, *fun*)

Performs a single fit and returns fit object

Parameters

- **x** (*ndarray of float*) – x-values
- **y** (*ndarray of float*) – y-values
- **fun** (*function*) – Function for fitting

Returns
fit – Fit object

Return type
gmodel fit object

pynibs.regression.regression.**stepdown_approach**(*zap_idx*, *elm_idx_list*, *fn_reg_hdf5=None*, *qoi_path_hdf5=None*, *e_matrix=None*, *mep=None*, *fun=<function sigmoid4>*, *con=None*, *n_refit=50*, *return_fits=False*, *constants=None*, *verbose=False*, *seed=None*, *rem_bad_elms=True*, *return_e_field_stats=True*, *score_type='R2'*, *return_progress=False*, *smooth_data=True*, *geo=None*)

Mass-univariate nonlinear regressions on raw MEP_{AMP} ~ E in a stepdown manner to speed up computation.

Initially, one set of fits is done for the complete dataset. Afterwards, the best 1% of the elements are used
as initial fitting parameters for their neighboring elements. Then, neighboring elements are fitted accordingly and iterativley. Finally, discontinuous elements are refitted until a smooth map is found or n_refit is hit.

Can be sped up with rem_bad_elms that computes a fast linear fit to identify elements with a negative slope.

The function reads the precomputed array of E-MEP data from an .hdf5 file.

Parameters

- **elm_idx_list** (*np.ndarray of int*) – List containing the element indices the fit is performed for
- **fn_reg_hdf5** (*str*) – Filename (incl. path) containing the precomputed E-MEP dataframes
- **qoi_path_hdf5** (*str or list of str*) – Path in .hdf5 file to dataset of electric field qoi e.g.: ["E", "E_norm", "E_tan"]
- **e_matrix** (*np.ndarray of float [n_zaps x n_ele] | [n_zapidx x n_ele]*) – Electric field matrix
- **mep** (*np.ndarray of float [n_zaps]*) – Motor evoked potentials for every stimulation
- **zap_idx** (*np.array [n_zaps], optional, default: None*) – Indices of zaps the congruence factor is calculated with (default: all)
- **fun** (*function object*) – A function of pynibs.exp.Mep (exp0, sigmoid)
- **con** (*np.ndarray of float [n_elm_roi, 3 or 4], optional, default: None*) – Connectivity matrix of ROI (needed in case of refit because of discontinuity check)
- **n_refit** (*int, optional, default: 50*) – Maximum number of refits of zero elements. No refit is applied in case of n_refit = 0.
- **return_fits** (*bool, optional, default: False*) – Return fit objects containing the parameter estimates
- **constants** (*dict of <string>:<num>, optional, default: None*) – key:value pair of model parameters not to optimize.
- **verbose** (*bool, optional, default: False*) – Plot output messages
- **seed** (*int, optional, default: None*) – Seed to use.
- **rem_bad_elms** (*bool, default: True*) – Remove elements based on a fast linear regression slope estimation
- **return_e_field_stats** (*bool, default: True*) – Return some stats on the efield variance

Returns

r2
[np.ndarray of float [n_roi, n_qoi]] R2 for each element in elm_idx_list

best_values: list of dict, optional
fit information

stats
[dict, optional]

'mc': float, optional
Mutual coherence for e fields

'sv_rat'
[float, optional] SVD singular value ratio

progress : cmaps for each step

Return type
dict

pynibs.regression.regression.**workhorse_element_init**(*ele_id*, *e_matrix*, *mep*, *fun*, *score_type*, *select_signed_data*, *constants*, ***kwargs*)

Workhorse to initialize Elements.

pynibs.regression.regression.**workhorse_element_run_fit**(*element*, *max_nfev=10*)

Workhorse to run single Element fit. If status is False, the element will not be fitted.

pynibs.regression.regression.**write_regression_hdf5**(*fn_exp_hdf5*, *fn_reg_hdf5*, *qoi_path_hdf5*, *qoi_phys*, *e_results_folder*, *qoi_e*, *roi_idx*, *conds_exclude*)

Reads the stimulation intensities from the experiment.hdf5 file. Reads the qoi from the experiment.hdf5 file. Reads the electric fields from the electric field folder. Weights the electric field voxel wise with the respective intensities and writes an .hdf5 file containing the preprocessed data (pandas dataframe).

fn_exp_hdf5: str
Filename of the experiment.hdf5 file

fn_reg_hdf5: str
Filename of output regression.hdf5 file

qoi_path_hdf5: str
Path in experiment.hdf5 file pointing to the pandas dataframe containing the qoi (e.g.: "phys_data/postproc/EMG")

qoi: str
Name of QOI the congruence factor is calculated with (e.g.: "p2p")

fn_e_results: str
Folder containing the electric fields (e.g.: "/data/pt_01756/probands/13061.30/results/electric_field/1")

qoi_e: str or list of str
Quantities of the electric field used to calculate the congruence factor (e.g. ["E", "E_norm", "E_tan"] Has to be included in e.hdf5 -> e.g.: "data/midlayer/roi_surface/1/E"

roi_idx: int
ROI index

conds_exclude: str or list of str
Conditions to exclude

Returns
<File> – File containing the intensity (current) scaled E-fields of the conditions in the ROI. Saved in datasets with the same name as qoi_e ["E", "E_norm", "E_tan"]

Return type
.hdf5 file

pynibs.regression.score_types module

pynibs.util package

Submodules

pynibs.util.plot module

pynibs.util.plot.**plot_surface**(*data*, *con*, *points*)

Plots data in triangle center

Parameters

- **data** (*np.array of float [n_tris]*) – Data in triangular center
- **con** (*np.array of float [n_tris x 3]*) – Connectivity list of triangles
- **points** (*np.array of float [n_points x 3]*) – Points (vertices) of surface mesh

pynibs.util.quality_measures module

pynibs.util.quality_measures.**c_map_comparison**(*c1*, *c2*, *t1*, *t2*, *nodes*, *tris*)

Compares two c-maps in terms of NRMSD and calculates the geodesic distance between the hotspots.

Parameters

- **c1** (*np.ndarray of float [n_ele]*) – First c-map
- **c2** (*np.ndarray of float [n_ele]*) – Second c-map (reference)
- **t1** (*np.ndarray of float [3]*) – Coordinates of the hotspot in the first c-map
- **t2** (*np.ndarray of float [3]*) – Coordinates of the hotspot in the second c-map
- **nodes** (*np.ndarray of float [n_nodes x 3]*) – Node coordinates
- **tris** (*np.ndarray of float [n_tris x 3]*) – Connectivity of ROI elements

Returns

- **nrmsd** (*float*) – Normalized root-mean-square deviation between the two c-maps in (%)
- **gdist** (*float*) – Geodesic distance between the two hotspots in (mm)

pynibs.util.quality_measures.**euclidean_dist**(*nodes*, *tris*, *source*, *source_is_node=True*)

Returns euclidean_dist distance of all nodes to source node (tria).

This is just a wrapper for the gdist package.

Example

```
with h5py.File(fn,'r') as f:
    tris = f['/mesh/elm/triangle_number_list'][:]
    nodes = f['/mesh/nodes/node_coord'][:]
nodes_dist_euc, tris_dist_euc = euclidean_dist(nodes, tris, 3017)

pynibs.write_data_hdf5(data_fn,
                       data=[tris_dist_euc, nodes_dist_euc],
                       data_names=["tris_dist_euc", "nodes_dist_euc", "])
pynibs.write_xdmf(data_fn,hdf5_geo_fn=fn)
```

Parameters

- **nodes** (*np.ndarray(n_nodes,3)*) –
- **tris** (*np.ndarray(n_tris,3)*) –

- **source** (*np.ndarray(int) or int*) – geodesic distances of all nodes to this will be computed
- **source_is_node** (*bool*) – Is source a node idx or a tr idx

Returns

- **nodes_dist** (*np.ndarray(n_nodes,)*)
- **tres_dist** (*np.ndarray(n_tris,)*)

pynibs.util.quality_measures.**geodesic_dist**(*nodes*, *tris*, *source*, *source_is_node=True*)

Returns geodesic distance of all nodes to source node (or tri).

This is just a wrapper for the gdist package.

Example

```
with h5py.File(fn,'r') as f:
    tris = f['/mesh/elm/triangle_number_list'][:]
    nodes = f['/mesh/nodes/node_coord'][:]
nodes_dist_ged, tris_dist_ged = geodesic_dist(nodes, tris, 3017)

pynibs.write_data_hdf5(data_fn,
                       data=[tris_dist_ged, nodes_dist_ged],
                       data_names=["tris_dist_ged", "nodes_dist_ged"])
pynibs.write_xdmf(data_fn,hdf5_geo_fn=fn)
```

Parameters

- **nodes** (*np.ndarray(n_nodes,3)*) –
- **tris** (*np.ndarray(n_tris,3)*) –
- **source** (*np.ndarray(int) or int*) – Geodesic distances of all nodes (or tri) to this will be computed
- **source_is_node** (*bool*) – Is source a node idx or a tr idx

Returns

- **nodes_dist** (*np.ndarray(n_nodes,)*)
- **tris_dist** (*np.ndarray(n_tris,)*)

pynibs.util.quality_measures.**nrmsd**(*array*, *array_ref*, *error_norm='relative'*, *x_axis=False*)

Determine the normalized root-mean-square deviation between input data and reference data.

Parameters

- **array** (*np.ndarray*) – input data [(x), y0, y1, y2 …]
- **array_ref** (*np.ndarray*) – reference data [(x_ref), y0_ref, y1_ref, y2_ref …] if array_ref is 1D, all sizes have to match
- **error_norm** (*str, optional, default="relative"*) – Decide if error is determined "relative" or "absolute"
- **x_axis** (*boolean, optional, default=False*) – If True, the first column of array and array_ref is interpreted as the x-axis, where the data points are evaluated. If False, the data points are assumed to be at the same location.

Returns

normalized_rms – Normalized root-mean-square deviation between the columns of array and array_ref

Return type

np.ndarray of float [array.shape[1]]

pynibs.util.simnibs module

pynibs.util.simnibs.**calc_e_in_midlayer_roi**(*phi_dadt*, *roi*, *subject*, *phi_scaling=1.0*, *mesh_idx=0*, *mesh=None*, *roi_hem='lh'*, *depth=0.5*, *qoi=None*, *layer_gm_wm_info=None*)

This is to be called by Simnibs as postprocessing function per FEM solve.

Parameters

- **phi_dadt** (*(array-like, elmdata)*) –
- **roi** (*pynibs.roi.RegionOfInterestSurface()*) –
- **subject** (*pynibs.subject.Subject()*) –
- **phi_scaling** (*float*) –
- **mesh_idx** (*int*) –
- **mesh** (*simnibs.msh.Mesh()*) –
- **roi_hem** (*'lh'*) –
- **depth** (*float*) –
- **qoi** (*list of str*) – List of identifiers of the to-be calculated quantities of interest.
- **layer_gm_wm_info** (*dict[str, dict[str,np.ndarray]]*) – For each layer defined in the mesh (outer dict key: layer_id), provide pre-computed geometrical information as a dictionary (outer dict value): - key: gm_nodes

  For each layer node, the corresponding point on the GM surface.

  - key: wm_nodes For each layer node, the corresponding point on the WM surface.
  - key: gm_center_distance The distance between the layer nodes and the corresponding points on the GM.
  - key: wm_center_distance The distance between the layer nodes and the corresponding points on the WM.

Returns

- **(roi.n_tris,4)** (*np.vstack((e_mag, e_norm, e_tan, e_angle)).transpose() for the midlayer*)
- **(len(roi.layers)x(roi.layers[idx].surface.n_tris,4))** (*np.vstack((e_mag, e_norm, e_tan, e_angle)).transpose()*)

pynibs.util.simnibs.**check_mesh**(*mesh*, *verbose=False*)

Check a simmibs.Mesh for degenerated elements:

- zero surface triangles
- zerso volume tetrahedra
- negative volume tetrahedra

Parameters

- **mesh** (*str or simnibs.Mesh*) –
- **verbose** (*book, default: False*) – Print some verbosity messages.

Returns

- **zero_tris** (*np.ndarray*) – Element indices for zero surface tris (0-indexed)
- **zero_tets** (*np.ndarray*) – Element indices for zero volume tets (0-indexed)
- **neg_tets** (*np.ndarray*) – Element indicies for negative volume tets (0-indexed)

pynibs.util.simnibs.**e_field_angle_theta**(*surface*, *mesh*)

Compute angle between local the E-field vector and surface vector at the ROI nodes.

Parameters

- **surface** (*simnibs.Msh*) – The surface object representing the ROI.
- **mesh** (*simnibs.Msh*) – The (volumetric) head mesh.

Returns

theta – The angle between local the E-field vector and surface vector for each surface element of the ROI.

Return type

simnibs.mesh.mesh_io.ElementData

pynibs.util.simnibs.**e_field_gradient_between_wm_gm**(*roi_surf*, *mesh*, *gm_nodes*, *wm_nodes*, *gm_center_distance*, *wm_center_distance*)

Compute local the E-field gradient at the ROI nodes between the gray and white matter boundary sufaces. Adapted from neuronibs/cortical_layer.py/add_e_field_gradient_between_wm_gm_field

Parameters

- **roi_surf** (*simnibs.Msh*) – The surface object representing the ROI.
- **volumetric_mesh** (*simnibs.Msh*) – The (volumetric) head mesh.

Returns

e_field_gradient_per_mm – The E-field gradient from the GM to the WM surface normalized to 1 mm with respect to the gray matter thickness. per ROI node

Return type

simnibs.mesh.mesh_io.ElementData

pynibs.util.simnibs.**fix_mesh**(*mesh*, *verbose=False*)

Fixes simnibs.Mesh by removing any zero surface tris and zero volume tets and by fixing negative volume tets.

Parameters

- **mesh** (*str or simnibs.Mesh*) – Filename of mesh or mesh object.
- **verbose** (*bool, default: False*) – Print some verbosity messages.

Returns

fixed_mesh

Return type

simnibs.Mesh

pynibs.util.simnibs.**precompute_geo_info_for_layer_field_interpolation**(*simnibs_mesh*, *roi*)

Precomputes geometric properties of the corresponding GM and WM nodes in the 'simnibs_mesh' of each node of each layer in 'roi'. > The corresponding point on the GM and WM surface to each vertex of te ROI surface

are determined (by raycasting and nearest neighbour search as fallback) (interpolation will take place between these nodes)

> The nodes are moved inside the gray matter by 20 % of their total distance from the GM/WM boundary to the midlayer.

> The distance of the relocated GM and WM nodes to the ROI nodes is determined (required to computed the gradient per mm).

Parameters

simnibs_mesh

[simnibs.Msh] The head model volume mesh.

roi: pynibs.roi.RegionOfInterestSurface
The ROI object containing the layers.

Returns

layer_gm_wm_info
[dict[str, dict[str,np.ndarray]]] For each layer defined in the mesh (outer dict key: layer_id), provide pre-computed geometrical information as a dictionary (outer dict value): - key: gm_nodes

For each layer node, the corresponding point on the GM surface.

- key: wm_nodes For each layer node, the corresponding point on the WM surface.
- key: gm_center_distance The distance between the layer nodes and the corresponding points on the GM.
- key: wm_center_distance The distance between the layer nodes and the corresponding points on the WM.

pynibs.util.simnibs.**read_coil_geo**(*fn_coil_geo*)

Reads a coil .geo file.

Parameters
fn_coil_geo (*str*) – Filename of .geo file created from SimNIBS containing the dipole information

Returns

- **dipole_pos** (*np.ndarray of float*)
- (n_dip, 3) Dipole positions `(x, y, z)`.
- **dipole_mag** (*np.ndarray of float*)
- *(n_dip, 1) Dipole magnitude.*

pynibs.util.simnibs.**smooth_mesh**(*mesh*, *output_fn*, *smooth=0.8*, *approach='taubin'*, *skin_only_output=False*, *smooth_tissue='skin'*)

Smoothes the skin compartment of a simnibs mesh. Uses one of three trimesh.smoothing approaches. Because tetrahedra and triangle share the same nodes, this also smoothes the volume domain.

Parameters

- **mesh** (*str or simnibs.Mesh*) –
- **output_fn** (*str*) –
- **smooth** (*float, default: 0.8*) – Smoothing aggressiveness. [0, . . . , 1]
- **approach** (*str, default: 'taubin'*) – Which smoothing approach to use. One of ('taubin', laplacian', 'humphrey)
- **smooth_tissue** (*str or list of int, default: 'skin'*) – Which tissue type to smooth. E.g. 'gm' or [2, 1002].
- **skin_only_output** (*bool, default: True*) – If true, a skin only mesh is written out instead of the full mesh.

Returns
<file>

Return type
The smoothed mesh.

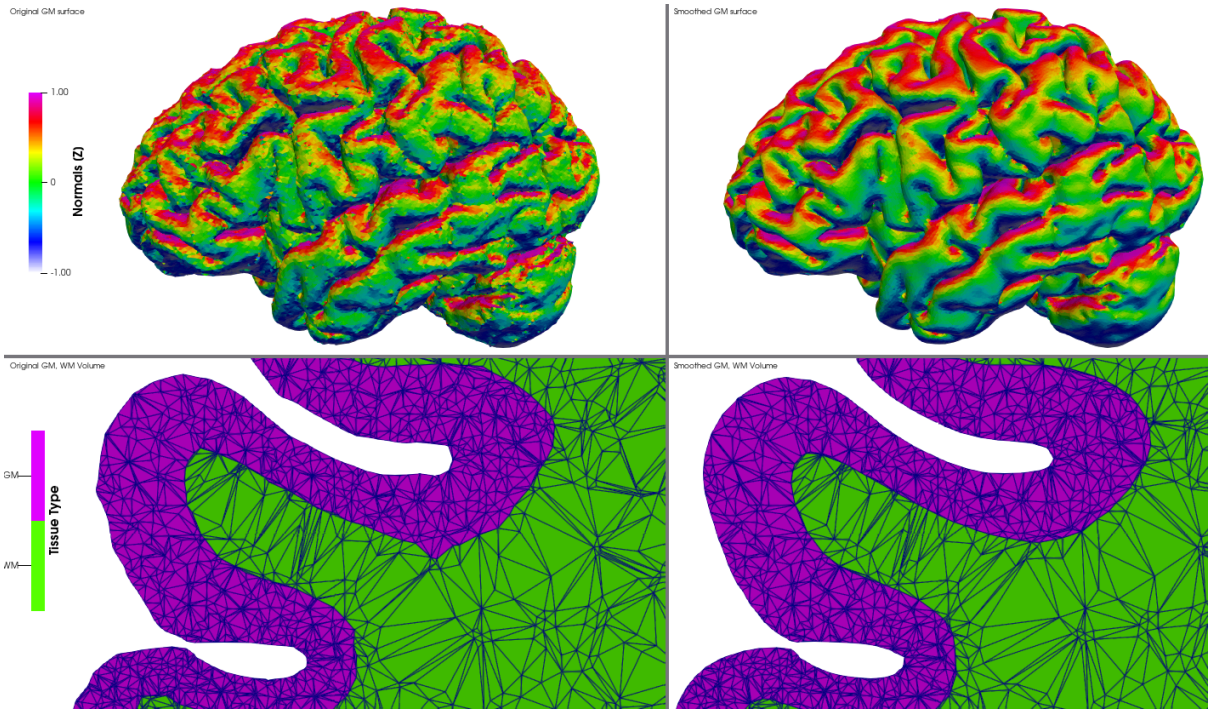

Fig. 2: Left: original, spiky mesh. Right: smoothed mesh.

pynibs.util.util module

pynibs.util.util.**add_center**(*var*)

Adds center to argument list.

Parameters
var (*list of float [2]*) – List containing two values [f1,f2]

Returns
out – List containing the average value in the middle [f1, mean(f1,f2), f2]

Return type
list of float [3]

pynibs.util.util.**bash**(*command*)

Executes bash command and returns output message in stdout (uses os.popen).

Parameters
command (*str*) – Bash command

Returns

- **output** (*str*) – Output from stdout
- **error** (*str*) – Error message from stdout

pynibs.util.util.**bash_call**(*command*)

Executes bash command and returns output message in stdout (uses subprocess.Popen).

Parameters
command (*str*) – bash command

pynibs.util.util.**calc_n_network_combs**(*n_e*, *n_c*, *n_i*)

Determine number of combinations if all conditions may be replaced between N_i elements (mixed interaction)

Parameters

- **n_e** (*int*) – Number of elements in the ROI
- **n_c** (*int*) – Number of conditions (I/O curves)
- **n_i** (*int*) – Number of maximum interactions

Returns
n_comb – Number of combinations

Return type
int

pynibs.util.util.**compute_chunks**(*seq*, *num*)

Splits up a sequence _seq_ into _num_ chunks of similar size. If len(seq) < num, (num-len(seq)) empty chunks are returned so that len(out) == num

Parameters

- **seq** (*list of something [N_ele]*) – List containing data or indices, which is divided into chunks
- **num** (*int*) – Number of chunks to generate

Returns
out – num sub-lists of seq with each of a similar number of elements (or empty).

Return type
list of num sublists

pynibs.util.util.**cross_product**(*A*, *B*)

Evaluates the cross product between the vector pairs in a and b using pure Python.

Parameters

- **A** (*nparray of float 2 x [N x 3]*) –
- **B** (*nparray of float 2 x [N x 3]*) – Input vectors, the cross product is evaluated between

Returns
c – Cross product between vector pairs in a and b

Return type
nparray of float [N x 3]

pynibs.util.util.**cross_product_einsum2**(*a*, *b*)

Evaluates the cross product between the vector pairs in a and b using the double Einstein sum.

Parameters

- **a** (*nparray of float 2 x [N x 3]*) –
- **b** (*nparray of float 2 x [N x 3]*) – Input vectors, the cross product is evaluated between

Returns
c – Cross product between vector pairs in a and b

Return type
nparray of float [N x 3]

pynibs.util.util.**differential_evolution**(*fobj*, *bounds*, *mut=0.8*, *crossp=0.7*, *popsize=20*, *its=1000*, ***kwargs*)

Differential evolution optimization algorithm

Parameters

- **fobj** (*function object*) – Function to optimize
- **bounds** (*dict*) – Dictionary containing the bounds of the free variables to optimize

- **mut** (*float*) – Mutation factor
- **crossp** (*float*) – Cross population factor
- **popsize** (*int*) – Population size
- **its** (*int*) – Number of iterations
- **kwargs** (*dict*) – Arguments passed to fobj (constants etc. . .)

Returns

- **best** (*dict*) – Dictionary containing the best values
- **fitness** (*float*) – Fitness value of best solution

pynibs.util.util.**euler_angles_to_rotation_matrix**(*theta*)

Determines the rotation matrix from the three Euler angles theta = [Psi, Theta, Phi] (in rad), which rotate the coordinate system in the order z, y', x".

Parameters
theta (*nparray [3]*) – Euler angles in rad

Returns
r – Rotation matrix (z, y', x")

Return type
nparray [3 x 3]

pynibs.util.util.**generalized_extreme_value_distribution**(*x*, *mu*, *sigma*, *k*)

Generalized extreme value distribution

Parameters

- **x** (*ndarray of float [n_x]*) – Events
- **mu** (*float*) – Mean value
- **sigma** (*float*) – Standard deviation
- **k** (*float*) – Shape parameter

Returns
y – Probability density of events

Return type
ndarray of float [n_x]

pynibs.util.util.**get_cartesian_product**(*array_list*)

Generate a cartesian product of input arrays (all combinations).

cartesian_product = get_cartesian_product(array_list)

Parameters
array_list (*list of 1D ndarray of float*) – Arrays to compute the cartesian product with

Returns
cartesian_product – Array containing the cartesian products (all combinations of input vectors) (M, len(arrays))

Return type
ndarray of float

Examples

```
>>> import pygpc
>>> out = pygpc.get_cartesian_product(([1, 2, 3], [4, 5], [6, 7]))
>>> out
```

pynibs.util.util.**invert**(*trans*)

Invert rotation matrix.

Parameters
trans (*nparray of float [3 x 3]*) – Rotation matrix

Returns
rot_inv – Inverse rotation matrix

Return type
nparray of float [3 x 3]

pynibs.util.util.**likelihood_posterior**(*x*, *y*, *fun*, *bounds=None*, *verbose=True*, *normalized_params=False*, ***params*)

Determines the likelihood of the data following the function "fun" assuming a two variability source of the data pairs (x, y) using the posterior distribution.

Parameters

- **x** (*ndarray of float [n_points]*) – x data
- **y** (*ndarray of float [n_points]*) – y data
- **fun** (*function*) – Function to fit the data to (e.g. sigmoid)
- **bounds** (*dict, optional, default: None*) – Dictionary containing the bounds of "sigma_x" and "sigma_y" and the free parameters of fun
- **verbose** (*bool, optional, default: True*) – Print function output after every calculation
- **normalized_params** (*bool, optional, default: False*) – Are the parameters passed in normalized space between [0, 1]? If so, bounds are used to denormalize them before calculation
- ****params** (*dict*) – Free parameters to optimize. Contains "sigma_x", "sigma_y", and the free parameters of fun

Returns
l – Negative likelihood

Return type
float

pynibs.util.util.**list2dict**(*l*)

Transform list of dicts with same keys to dict of list

Parameters
l (*list of dict*) – List containing dictionaries with same keys

Returns
d – Dictionary containing the entries in a list

Return type
dict of lists

pynibs.util.util.**load_muaps**(*fn_muaps*, *fs=1000000.0*, *fs_downsample=100000.0*)

pynibs.util.util.**mutual_coherence**(*array*)

Calculate the mutual coherence of a matrix A. It can also be referred as the cosine of the smallest angle between two columns.

mutual_coherence = mutual_coherence(array)

Parameters
array (*ndarray of float*) – Input matrix

Returns
mutual_coherence – Mutual coherence

Return type
float

pynibs.util.util.**norm_percentile**(*data*, *percentile*)

Normalizes data to a given percentile.

Parameters

- **data** (*nparray [n_data,]*) – Dataset to normalize
- **percentile** (*float*) – Percentile of normalization value [0 … 100]

Returns
data_norm – Normalized dataset

Return type
nparray [n_data,]

pynibs.util.util.**normalize_rot**(*rot*)

Normalize rotation matrix.

Parameters
rot (*nparray of float [3 x 3]*) – Rotation matrix

Returns
rot_norm – Normalized rotation matrix

Return type
nparray of float [3 x 3]

pynibs.util.util.**quat_rotation_angle**(*q*)

Computes the rotation angle from the quaternion in rad.

Parameters
q (*nparray of float*) – Quaternion, either only the imaginary part (length=3) [qx, qy, qz] or the full quaternion (length=4) [qw, qx, qy, qz]

Returns
alpha – Rotation angle of quaternion in rad

Return type
float

pynibs.util.util.**quat_to_rot**(*q*)

Computes the rotation matrix from quaternions.

Parameters
q (*nparray of float*) – Quaternion, either only the imaginary part (length=3) or the full quaternion (length=4)

Returns
rot – Rotation matrix, containing the x, y, z axis in the columns

Return type
nparray of float [3 x 3]

pynibs.util.util.**quaternion_conjugate**(*q*)

https://stackoverflow.com/questions/15425313/inverse-quaternion

Parameters
q –

Returns

Return type

pynibs.util.util.**quaternion_diff**(*q1*, *q2*)

https://math.stackexchange.com/questions/2581668/ error-measure-between-two-rotations-when-one-matrix-might-not-be-a-valid-rotatio

Parameters

- **q1** –
- **q2** –

Returns

Return type

pynibs.util.util.**quaternion_inverse**(*q*)

https://stackoverflow.com/questions/15425313/inverse-quaternion

Parameters
q –

Returns

Return type

pynibs.util.util.**rd**(*array*, *array_ref*)

Determine the relative difference between input data and reference data.

Parameters

- **array** (*np.ndarray*) – input data [(x), y0, y1, y2 ...]
- **array_ref** (*np.ndarray*) – reference data [(x_ref), y0_ref, y1_ref, y2_ref ...] if array_ref is 1D, all sizes have to match

Returns
rd – Relative difference between the columns of array and array_ref

Return type
ndarray of float [array.shape[1]]

pynibs.util.util.**recursive_len**(*item*)

Determine len of list of lists (recursively).

Parameters
item (*list of list*) – List of list

Returns
len – Total length of list of lists

Return type
int

pynibs.util.util.**rot_to_quat**(*rot*)

Computes the quaternions from rotation matrix. (see e.g. http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/)

Parameters
rot (*nparray of float [3 x 3]*) – Rotation matrix, containing the x, y, z axis in the columns

Returns
q – Quaternion, full (length=4)

Return type
nparray of float

pynibs.util.util.**rotation_matrix_to_euler_angles**(*r*)

Calculates the euler angles theta = [Psi, Theta, Phi] (in rad) from the rotation matrix R which, rotate the coordinate system in the order z, y', x''. (https://www.learnopencv.com/rotation-matrix-to-euler-angles/)

Parameters
r (*np.array [3 x 3]*) – Rotation matrix (z, y', x'')

Returns
theta – Euler angles in rad

Return type
np.array [3]

pynibs.util.util.**sigmoid_log_p**(*x*, *p*)

pynibs.util.util.**tal2mni**(*coords*, *direction='tal2mni'*, *style='nonlinear'*)

Transform Talairach coordinates into (SPM) MNI space and vice versa.

This is taken from https://imaging.mrc-cbu.cam.ac.uk/imaging/MniTalairach and http://gibms.mc.ntu.edu.tw/bmlab/tools/data-analysis-codes/mni2tal-m/

Parameters

- **coords** (*np.ndarray or list*) – x,y,z coordinates
- **direction** (*str, default: 'tal2mni'*) – Transformation direction. One of ('tal2mni', 'mni2tal')
- **style** (*str, default: 'nonlinear'*) – Transformation style. One of ('linear', 'nonlinear')

Returns
coords_trans

Return type
np.ndarray

pynibs.util.util.**unique_rows**(*a*)

Returns the unique rows of np.array(a).

Parameters
a (*nparray of float [m x n]*) – Array to search for double row entries

Returns
a_unique – array a with only unique rows

Return type
np.array [k x n]

1.1.2 Submodules

1.1.3 pynibs.coil module

pynibs.coil.**calc_coil_position_pdf**(*fn_rescon=None*, *fn_simpos=None*, *fn_exp=None*, *orientation='quaternions'*, *folder_pdfplots=None*)

Determines the probability density functions of the transformed coil position (x', y', z') and quaternions of the coil orientations (x'', y'', z'')

Parameters

- **fn_rescon** (*str*) – Filename of the results file from TMS experiments (results_conditions.csv)
- **fn_simpos** (*str*) – Filename of the positions and orientation from TMS experiments (simPos.csv)

- **fn_exp** (*str*) – Filename of experimental.csv file from experiments
- **orientation** (*str*) – Type of orientation estimation: 'quaternions' or 'euler'
- **folder_pdfplots** (*str*) – Folder, where the plots of the fitted pdfs are saved (omitted if not provided)

Returns

- **pdf_paras_location** (*list of list of np.ndarrays [n_conditions]*) –

  Pdf parameters (limits and shape) of the coil position for x', y', and z' for each:

  – beta_paras … [p, q, a, b] (2 shape parameters and limits)

  – moments … [data_mean, data_std, beta_mean, beta_std]

  – p_value … p-value of the Kolmogorov Smirnov test

  – uni_paras … [a, b] (limits)

- **pdf_paras_orientation_euler** (*list of np.ndarray [n_conditions]*) –

  Pdf parameters (limits and shape) of the coil orientation Psi, Theta, and Phi for each:

  – beta_paras … [p, q, a, b] (2 shape parameters and limits)

  – moments … [data_mean, data_std, beta_mean, beta_std]

  – p_value … p-value of the Kolmogorov Smirnov test

  – uni_paras … [a, b] (limits)

- **OP_mean** (*List of [3 x 4] np.ndarray [n_conditions]*) – List of mean coil position and orientation for different conditions (global coordinate system)

$$\begin{bmatrix} | & | & | & | \\ ori_x & ori_y & ori_z & pos \\ | & | & | & | \end{bmatrix}$$

- **OP_zeromean** (*list of [3 x 4 x n_con_each] np.ndarray [n_conditions]*) – List over conditions containing zero-mean coil orientations and positions
- **V** (*list of [3 x 3] np.ndarrays [n_conditions]*) – Transformation matrix of coil positions from global coordinate system to transformed coordinate system
- **P_transform** (*list of np.ndarray [n_conditions]*) – List over conditions containing transformed coil positions [x', y', z'] of all stimulations (zero-mean, rotated by SVD)
- **quaternions** (*list of np.ndarray [n_conditions]*) – List over conditions containing imaginary part of quaternions [x", y", z"] of all stimulations

pynibs.coil.**calc_coil_transformation_matrix**(*LOC_mean*, *ORI_mean*, *LOC_var*, *ORI_var*, *V*)

Calculate the modified coil transformation matrix needed for simnibs based on location and orientation variations observed in the framework of uncertainty analysis

Parameters

- **LOC_mean** (*np.ndarray of float*) – (3), Mean location of TMS coil
- **ORI_mean** (*np.ndarray of float*) – (3 x 3) Mean orientations of TMS coil

$$\begin{bmatrix} | & | & | \\ x & y & z \\ | & | & | \end{bmatrix}$$

- **LOC_var** (*np.ndarray of float*) –

(3) Location variation in normalized space (dx', dy', dz'), i.e. zero mean and projected on principal axes

- **ORI_var** (*np.ndarray of float*) –

(3) Orientation variation expressed in Euler angles [alpha, beta, gamma] in deg

- **V** (*np.ndarray of float*) – (3x3) V-matrix containing the eigenvectors from _,_,V = numpy.linalg.svd

Returns

- **mat** (*np.ndarray of float*)
- *(4, 4) Transformation matrix containing 3 axis and 1 location vector –*

$$\begin{bmatrix} | & | & | & | \\ x & y & z & pos \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

pynibs.coil.**check_coil_position**(*points*, *hull*)

Check if magnetic dipoles are lying inside head region

Parameters

- **points** (*np.ndarray of float*) – (N_points, 3) Coordinates (x,y,z) of magnetic dipoles
- **hull** (*Delaunay object or np.ndarray of float*) – (N_surface_points, 3) Head surface data

Returns

valid – Validity of coil position: TRUE: valid FALSE: unvalid

Return type

bool

pynibs.coil.**create_stimsite_from_exp_hdf5**(*fn_exp*, *fn_hdf*, *datanames=None*, *data=None*, *overwrite=False*)

This takes an experiment.hdf5 file and creates an .hdf5 + .xdmf tuple for all coil positions for visualization.

Parameters

- **fn_exp** (*str*) – Path to experiment.hdf5
- **fn_hdf** (*str*) – Filename for the resulting .hdf5 file. The .xdmf is saved with the same basename. Folder should already exist.
- **datanames** (*str or list of str*) – Dataset names for _data_. Default: None.
- **data** (*np.ndarray*) – Dataset array with (len(poslist.pos), len(datanames()). Default: None.
- **overwrite** (*boolean*) – Overwrite existing files. Default: False.

pynibs.coil.**create_stimsite_from_list**(*fn_hdf*, *poslist*, *datanames=None*, *data=None*, *overwrite=False*)

This takes a TMSLIST from simnibs and creates a .hdf5 + .xdmf tuple for all positions.

Centers and coil orientations are written so disk.

Parameters

- **fn_hdf** (*str*) – Filename for the .hdf5 file. The .xdmf is saved with the same basename. Folder should already exist.
- **datanames** (*str or list of str*) – Dataset names for _data_. Default: None.
- **data** (*np.ndarray*) – Dataset array with (len(poslist.pos), len(datanames()). Default: None.

- **poslist** (*TMSLIST object (simnibs.simulation.simstruct.TMSLIST)*) – poslist.pos[*].matsimnibs have to be set.
- **overwrite** (*boolean*) – Overwrite existing files. Default: False.

pynibs.coil.**create_stimsite_from_matsimnibs**(*fn_hdf*, *matsimnibs*, *datanames=None*, *data=None*, *overwrite=False*)

This takes a matsimnibs array and creates an .hdf5 + .xdmf tuple for all coil positions for visualization.

Centers and coil orientations are written disk.

Parameters

- **fn_hdf** (*str*) – Filename for the .hdf5 file. The .xdmf is saved with the same basename. Folder should already exist.
- **matsimnibs** (*np.ndarray*) – (4, 4, n_pos) Matsimnibs matrices containing the coil orientation (x,y,z) and position (p)

$$\begin{bmatrix} | & | & | & | \\ x & y & z & p \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **datanames** (*str or list of str*) – Dataset names for _data_. Default: None.
- **data** (*np.ndarray, optional*) – (len(poslist.pos), len(datanames).
- **overwrite** (*boolean, default: False*) – Overwrite existing files.

pynibs.coil.**create_stimsite_from_tmslist**(*fn_hdf*, *poslist*, *datanames=None*, *data=None*, *overwrite=False*)

This takes a TMSLIST from simnibs and creates a .hdf5 + .xdmf tuple for all positions.

Centers and coil orientations are written so disk.

Parameters

- **fn_hdf** (*str*) – Filename for the .hdf5 file. The .xdmf is saved with the same basename. Folder should already exist.
- **datanames** (*str or list of str*) – Dataset names for _data_. Default: None.
- **data** (*np.ndarray*) – Dataset array with (len(poslist.pos), len(datanames()). Default: None.
- **poslist** (*TMSLIST object (simnibs.simulation.simstruct.TMSLIST)*) – poslist.pos[*].matsimnibs have to be set.
- **overwrite** (*boolean*) – Overwrite existing files. Default: False.

pynibs.coil.**create_stimsite_hdf5**(*fn_exp*, *fn_hdf*, *conditions_selected=None*, *sep='_'*, *merge_sites=False*, *fix_angles=False*, *data_dict=None*, *conditions_ignored=None*)

Reads results_conditions and creates an hdf5/xdmf pair with condition-wise centers of stimulation sites and coil directions as data.

Parameters

- **fn_exp** (*str*) – Path to results.csv
- **fn_hdf** (*str*) – Path where to write file. Gets overridden if already existing
- **conditions_selected** (*str or list of str, Default=None*) – List of conditions returned by the function, the others are omitted, If None, all conditions are returned
- **sep** (*str, Default: "_"*) – Separator between condition label and angle (e.g. M1_0, or M1-0)

- **merge_sites** (*boolean*) – If true, only one coil center per site is generated.
- **fix_angles** (*boolean*) – rename 22.5 -> 0, 0 -> -45, 67.5 -> 90, 90 -> 135
- **data_dict** (*dict ofnp.ndarray of float [n_stimsites] (optional), default: None*) – Dictionary containing data corresponding to the stimulation sites (keys)
- **conditions_ignored** (*str or list of str, Default=None*) – Conditions, which are not going to be included in the plot

Returns
<Files> – Contains information about condition-wise stimulation sites and coil directions (fn_hdf)

Return type
hdf5/xdmf file pair

Example

```
pynibs.create_stimsite_hdf5('/exp/1/experiment_corrected.csv',
                            '/stimsite', True, True)
```

pynibs.coil.**get_coil_dipole_pos**(*coil_fn*, *matsimnibs*)

Apply transformation to coil dipoles and return position.

Parameters

- **coil_fn** (*str*) – Filename of coil .ccd file
- **matsimnibs** (*np.ndarray of float*) – Transformation matrix

Returns

- **dipoles_pos** (*np.ndarray*)
- *(N, 3) Cartesian coordinates (x, y, z) of coil magnetic dipoles*

pynibs.coil.**get_invalid_coil_parameters**(*param_dict*, *coil_position_mean*, *svd_v*, *del_obj*, *fn_coil*, *fn_hdf5_coilpos=None*)

Finds gpc parameter combinations, which place coil dipoles inside subjects head. Only endpoints (and midpoints) of the parameter ranges are examined.

get_invalid_coil_parameters(param_dict, pos_mean, v, del_obj, fn_coil, fn_hdf5_coilpos=None)

Parameters

- **param_dict** (*dict*) – Dictionary containing dictionary with 'limits' and 'pdfshape' keys: 'x', 'y', 'z', 'psi', 'theta', 'phi'
- **coil_position_mean** (*np.ndarray [3 x 4]*) – Mean coil positions and orientations
- **svd_v** (*np.ndarray [3 x 3]*) – SVD matrix V
- **del_obj** (*Delaunay object*) – Skin surface (created with scipy.spatial.Delaunay)
- **fn_coil** (*str*) – Filename of coil .ccd file
- **fn_hdf5_coilpos** (*str*) – Filename of .hdf5 file to save coil_pos in (incl. path and .hdf5 extension)

Returns
fail_params – Index and combination of failed parameter

Return type
list of int

pynibs.coil.**sort_opt_coil_positions**(*fn_coil_pos_opt*, *fn_coil_pos*, *fn_out_hdf5=None*, *root_path='/0/0/'*, *verbose=False*, *print_output=False*)

Sorts coil positions according to Traveling Salesman problem

Parameters

- **fn_coil_pos_opt** (*str*) – Name of .hdf5 file containing the optimal coil position indices
- **fn_coil_pos** (*str*) – Name of .hdf5 file containing the matsimnibs matrices of all coil positions
- **fn_out_hdf5** (*str*) – Name of output .hdf5 file (will be saved in the same format as fn_coil_pos_opt)
- **verbose** (*bool, optional, default: False*) – Print output messages
- **print_output** (*bool or str, optional, default: False*) – Print output image as .png file showing optimal path

Return type
<file> .hdf5 file containing the sorted optimal coil position indices

pynibs.coil.**test_coil_position_gpc**(*parameters*)

Testing valid coil positions for gPC analysis

pynibs.coil.**write_coil_pos_hdf5**(*fn_hdf*, *centers*, *m0*, *m1*, *m2*, *datanames=None*, *data=None*, *overwrite=False*)

Creates a .hdf5 + .xdmf file for all coil positions.

Centers and coil orientations are written to disk.

Parameters

- **fn_hdf** (*str*) – Filename for the .hdf5 file. The .xdmf is saved with the same basename. Folder should already exist.
- **centers** (*np.ndarray of float [n_pos x 3]*) – Coil positions
- **m0** (*np.ndarray of float [n_pos x 3]*) – Coil orientation x-axis (looking at the active (patient) side of the coil pointing to the right)
- **m1** (*np.ndarray of float [n_pos x 3]*) – Coil orientation y-axis (looking at the active (patient) side of the coil pointing up away from the handle)
- **m2** (*np.ndarray of float [n_pos x 3]*) – Coil orientation z-axis (looking at the active (patient) side of the coil pointing to the patient)
- **datanames** (*str or list of str [n_data]*) – Dataset names for _data_. Default: None.
- **data** (*np.ndarray [n_pos, n_data]*) – Dataset array with (len(poslist.pos), len(datanames()). Default: None.
- **overwrite** (*boolean*) – Overwrite existing files. Default: False.

1.1.4 pynibs.freesurfer module

This holds methods to interact with FreeSurfer ([1]), for example to translate FreeSurfer files into Paraview readable .vtk files.

References

pynibs.freesurfer.**data_sub2avg**(*fn_subject_obj*, *fn_average_obj*, *hemisphere*, *fn_in_hdf5_data*, *data_hdf5_path*, *data_label*, *fn_out_hdf5_geo*, *fn_out_hdf5_data*, *mesh_idx=0*, *roi_idx=0*, *subject_data_in_center=True*, *data_substitute=-1*, *verbose=True*, *replace=True*, *reg_fn='sphere.reg'*)

Maps the data from the subject space to the average template. If the data is given only in an ROI, the data is mapped to the whole brain surface.

Parameters

- **fn_subject_obj** (*str*) – Filename of subject object .hdf5 file (incl. path), e.g. `.../probands/subjectID/subjectID.hdf5`
- **fn_average_obj** (*str*) – Filename of average template object .pkl file (incl. path), e.g. `.../probands/avg_template/avg_template.hdf5`.
- **hemisphere** (*str*) – Define hemisphere to work on (`'lh'` or `'rh'` for left or right hemisphere, respectively).
- **fn_in_hdf5_data** (*str*) – Filename of .hdf5 data input file containing the subject data.
- **data_hdf5_path** (*str*) – Path in .hdf5 data file where data is stored (e.g. `'/data/tris/'`).
- **data_label** (*str or list of str*) – Label of datasets contained in hdf5 input file to map.
- **fn_out_hdf5_geo** (*str*) – Filename of .hdf5 geo output file containing the geometry information.
- **fn_out_hdf5_data** (*str*) – Filename of .hdf5 data output file containing the mapped data.
- **mesh_idx** (*int*) – Index of mesh used in the simulations.
- **roi_idx** (*int*) – Index of region of interest used in the simulations.
- **subject_data_in_center** (*bool, default: True*) – Specify if the data is given in the center of the triangles or in the nodes.
- **data_substitute** (*float*) – Data substitute with this number for all points outside the ROI mask
- **verbose** (*bool*) – Verbose output (Default: True)
- **replace** (*bool*) – Replace output files (Default: True)
- **reg_fn** (*str*) – Sphere.reg fn

Returns

<Files> – Geometry and corresponding data files to plot with Paraview:

- fn_out_hdf5_geo.hdf5: geometry file containing the geometry information of the average template
- fn_out_hdf5_data.hdf5: geometry file containing the data

Return type
.hdf5 files

pynibs.freesurfer.**freesurfer2vtk**(*in_fns*, *out_folder*, *hem='lh'*, *surf='pial'*, *prefix=None*, *fs_subject='fsaverage'*, *fs_subjects_dir=None*)

Transform multiple FreeSurfer .mgh files into one .vtk file. This can be read with Paraview and others.

Parameters

- **in_fns** (*list of str or str*) – Input filenames.
- **out_folder** (*str*) – Output folder.
- **hem** (*str, default: 'lh'*) – Which hemisphere: `'lh'` or `'rh'`.
- **surf** (*str, default: 'pial'*) – Which FreeSurfer surface: `'pial'`, `'inflated'`, …
- **prefix** (*str, optional*) – Prefix to add to each filename.
- **fs_subject** (*str, default: 'fsaverage'*) – FreeSurfer subject.
- **fs_subjects_dir** (*str, optional*) – FreeSurfer subjects directory. If not provided, read from environment.

Returns

<File> – One .vtk file with data as overlays from all .mgh files provided

Return type

out_folder/{prefix}_{hem}_{surf}.vtk

pynibs.freesurfer.**make_average_subject**(*subjects*, *subject_dir*, *average_dir*, *fn_reg='sphere.reg'*)

Generates the average template from a list of subjects using the FreeSurfer average.

Parameters

- **subjects** (*list of str*) – Paths of subjects directories, where the FreeSurfer files are located, e.g. for simnibs mri2mesh `.../fs_SUBJECT_ID`.
- **subject_dir** (*str*) – Temporary subject directory of FreeSurfer (symlinks of subjects will be generated in there and average template will be temporarily stored before it is copied to `average_dir`).
- **average_dir** (*str*) – path to directory where average template will be stored, e.g. probands/avg_template_15/mesh/0/fs_avg_template_15.
- **fn_reg** (*str, default: 'sphere.reg' --> ?h.sphere.reg*) – Filename suffix of FreeSurfer registration file containing registration information to template.

Returns

<Files> – Average template in average_dir and registered curvature files, `?h.sphere.reg` in subjects/surf folders.

Return type

.tif and .reg files

pynibs.freesurfer.**make_group_average**(*subjects=None*, *subject_dir=None*, *average=None*, *hemi='lh'*, *template='mytemplate'*, *steps=3*, *n_cpu=2*, *average_dir=None*)

Creates a group average from scratch, based on one subject. This prevents for example the fsaverage problems of large elements at M1, etc. This is an implemntation of [2] 'Creating a registration template from scratch (GW)'.

References

Parameters

- **subjects** (*list of str*) – List of FreeSurfer subjects names.
- **subject_dir** (*str*) – Temporary subject directory of FreeSurfer (symlinks of subjects will be generated in there and average template will be temporarily stored before it is copied to `average_dir`).
- **average** (str, default: `subjects[0]`) – Which subject to base new average template on.
- **hemi** (*str, default: 'lh'*) – Which hemisphere: `lh` or `rh`.

  Deprecated since version 0.0.1: Don't use any more.
- **template** (*str, default: 'mytemplate'*) – Basename of new template.
- **steps** (*int, default: 2*) – Number of iterations.
- **n_cpu** (*int, default: 4*) – How many cores for multithreading.
- **average_dir** (*str*) – Path to directory where average template will be stored, e.g. `probands/avg_template_15/mesh/0/fs_avg_template_15`.

Returns

- **<File>** (*.tif file*) – `SUBJECT_DIR/TEMPLATE*.tif`, TEMPLATE0.tif based on AVERAGE, rest on all subjects.
- **<File>** (*.myreg file*) – `SUBJECT_DIR/SUBJECT*/surf/HEMI.sphere.myreg*`.
- **<File>** (*.tif file*) – Subject wise sphere registration based on `TEMPLATE*.tif`.

`pynibs.freesurfer.`**read_curv_data**(*fname_curv*, *fname_inf*, *raw=False*)

Read curvature data provided by FreeSurfer with optional normalization.

Parameters

- **fname_curv** (*str*) – Filename of the FreeSurfer curvature file (e.g. ?h.curv), contains curvature data in nodes can be found in mri2mesh proband folder: `proband_ID/fs_ID/surf/?h.curv`.
- **fname_inf** (*str*) – Filename of inflated brain surface (e.g. `?h.inflated`), contains points and connectivity data of surface can be found in mri2mesh proband folder: `proband_ID/fs_ID/surf/?h.inflated`.
- **raw** (*bool*) – If raw-data is returned or if the data is normalized to -1 for neg. and +1 for pos. curvature.

Returns

curv – Curvature data in element centers.

Return type

np.ndarray of float or int

1.1.5 pynibs.hdf5_io module

pynibs.hdf5_io.**create_fibre_geo_hdf5**(*fn_fibres_hdf5*, *overwrite=True*)

Reformats geometrical fibre data and adds a /plot subfolder containing geometrical fibre data including connectivity

Parameters

- **fn_fibres_hdf5** (*str*) – Path to fibre.hdf5 file containing the original fibre data
- **overwrite** (*bool*) – Overwrites existing /plot subfolder in .hdf5 file

pynibs.hdf5_io.**create_fibre_xdmf**(*fn_fibre_geo_hdf5*, *fn_fibre_data_hdf5=None*, *overwrite=True*, *fibre_points_path='fibre_points'*, *fibre_con_path='fibre_con'*, *fibre_data_path=''*)

Creates .xdmf file to plot fibres in Paraview

Parameters

- **fn_fibre_geo_hdf5** (*str*) – Path to fibre_geo.hdf5 file containing the geometry (in /plot subfolder created with create_fibre_geo_hdf5())
- **fn_fibre_data_hdf5** (*str (optional) default: None*) – Path to fibre_data.hdf5 file containing the data to plot (in parent folder)
- **fibre_points_path** (*str (optional) default: fibre_points*) – Path to fibre point array in .hdf5 file
- **fibre_con_path** (*str (optional) default: fibre_con*) – Path to fibre connectivity array in .hdf5 file
- **fibre_data_path** (*str (optional) default: ""*) – Path to parent data folder in data.hdf5 file (Default: no parent folder)

Returns
<File>

Return type
.xdmf file for Paraview

pynibs.hdf5_io.**create_position_path_xdmf**(*sorted_fn*, *coil_pos_fn*, *output_xdmf*, *stim_intens=None*, *coil_sorted='/0/0/coil_seq'*)

Creates one .xdmf file that allows paraview plottings of coil position paths.

Parameters

- **sorted_fn** (*str*) – .hdf5 filename with position indices, values, intensities from pynibs.sort_opt_coil_positions()
- **coil_pos_fn** (*str*) – .hdf5 filename with original set of coil positions. Indices from sorted_fn are mapped to this. Either '/matsimnibs' or 'm1' and 'm2' datasets.
- **output_xdmf** (*str*) –
- **stim_intens** (*int, optional*) – Intensities are multiplied by this factor
- **coil_sorted** (*str*) – Path to coil positions in sorted_fn

Returns
output_xdmf

Return type
<file>

pynibs.hdf5_io.**data_superimpose**(*fn_in_hdf5_data*, *fn_in_geo_hdf5*, *fn_out_hdf5_data*, *data_hdf5_path='/data/tris/'*, *data_substitute=-1*, *normalize=False*)

Overlaying data stored in .hdf5 files except in regions where data_substitute is found. These points are omitted in the analysis and will be replaced by data_substitute instead.

Parameters

- **fn_in_hdf5_data** (*list of str*) – Filenames of .hdf5 data files with common geometry (e.g. generated by pynibs.data_sub2avg(...))
- **fn_in_geo_hdf5** (*str*) – Geometry .hdf5 file, which corresponds to the .hdf5 data files
- **fn_out_hdf5_data** (*str*) – Filename of .hdf5 data output file containing the superimposed data
- **data_hdf5_path** (*str*) – Path in .hdf5 data file where data is stored (e.g. '/data/tris/')
- **data_substitute** (*float or NaN*) – Data substitute with this number for all points in the inflated brain, which do not belong to the given data set (Default: -1)
- **normalize** (*boolean or str*) – Decide if individual datasets are normalized w.r.t. their maximum values before they are superimposed (Default: False)
  - 'global': global normalization w.r.t. maximum value over all datasets and subjects
  - 'dataset': dataset wise normalization w.r.t. maximum of each dataset individually (over subjects)
  - 'subject': subject wise normalization (over datasets)

Returns
<File> – Overlayed data

Return type
.hdf5 file

pynibs.hdf5_io.**hdf_2_ascii**(*hdf5_fn*)

Prints out structure of given .hdf5 file.

Parameters
hdf5_fn (*str*) – Filename of .hdf5 file.

Returns
h5 – Structure of .hdf5 file

Return type
items

pynibs.hdf5_io.**load_mesh_hdf5**(*fname*)

Loading mesh from .hdf5 file and setting up *TetrahedraLinear* class.

Parameters
fname (*str*) – Name of .hdf5 file (incl. path)

Returns
obj – *TetrahedraLinear* object

Return type
pynibs.mesh.mesh_struct.TetrahedraLinear

Example

.hdf5 file format and contained groups. The content of .hdf5 files can be shown using the tool HDFView (https://support.hdfgroup.org/products/java/hdfview/)

```
mesh
I---/elm
I    I--/elm_number          [1,2,3,...,N_ele]          Running index over all elements starting at 1,
                                                            triangles and tetrahedra
I    I--/elm_type            [2,2,2,...,4,4]            Element type: 2 triangles, 4 tetrahedra
I    I--/node_number_list    [1,5,6,0;... ;1,4,8,9]     Connectivity of triangles [X, X, X, 0] and tetrahedra
                                                                [X, X, X, X]
I    I--/tag1                [1001,1001, ..., 4,4,4]    Surface (100X) and domain (X) indices with 1000 offset
                                                                for surfaces
I    I--/tag2                [   1,   1, ..., 4,4,4]    Surface (X) and domain (X) indices w/o offset
I
I---/nodes
```

(continues on next page)

(continued from previous page)

```
I    I--/node_coord          [1.254, 1.762, 1.875;...]   Node coordinates in (mm)
I    I--/node_number         [1,2,3,...,N_nodes]         Running index over all points starting at 1
I    I--/units               ["mm"]                      .value is unit of geometry
I
I---/fields
I    I--/E/value             [E_x_1, E_y_1, E_z_1;...]   Electric field in all elms, triangles and tetrahedra
I    I--/J/value             [J_x_1, J_y_1, J_z_1;...]   Current density in all elms, triangles and tetrahedra
I    I--/normE/value         [normE_1,..., normE_N_ele]  Magnitude of electric field in all elements,
                                                                triangles and tetrahedra
I    I--/normJ/value         [normJ_1,..., normJ_N_ele]  Magnitude of current density in all elements,
                                                                triangles and tetrahedra


/data
I---/potential               [phi_1, ..., phi_N_nodes]   Scalar electric potential in nodes (size N_nodes)
I---/dAdt                    [A_x_1, A_y_1, A_z_1,...]   Magnetic vector potential (size 3xN_nodes)
```

pynibs.hdf5_io.**load_mesh_msh**(*fname*)

Loading mesh from .msh file and return *TetrahedraLinear* object.

Parameters
fname (*str*) – .msh filename (incl. path)

Returns
obj

Return type
pynibs.mesh.mesh_struct.TetrahedraLinear

pynibs.hdf5_io.**msh2hdf5**(*fn_msh=None*, *skip_roi=False*, *include_data=False*, *approach='mri2mesh'*, *subject=None*, *mesh_idx=None*)

Transforms mesh from .msh to .hdf5 format. Mesh is read from subject object or from fn_msh.

Parameters

- **fn_msh** (*str, optional, default: None*) – Filename of .msh file
- **skip_roi** (*bool, optional, default: False*) – Skip generating ROI in .hdf5
- **include_data** (*bool, optional, default: False*) – Also convert data in .msh file to .hdf5 file
- **subject** (*Subject object, optional, default: None*) – Subject object
- **mesh_idx** (*int or list of int, optional, default: None*) – Mesh index, the conversion from .msh to .hdf5 is conducted for
- **approach** (*str*) – Approach the headmodel was created with (“mri2mesh” or “headreco”)

  Deprecated since version 0.0.1: Not supported any more.

Returns
<File> – .hdf5 file with mesh information

Return type
.hdf5 file

pynibs.hdf5_io.**print_attrs**(*name*, *obj*)

Helper function for *hdf_2_ascii()*. To be called from `h5py.Group.visititems()`

Parameters

- **name** (*str*) – Name of structural element
- **obj** (*object*) – Structural element

Returns
<Print>

Return type
Structure of .hdf5 file

pynibs.hdf5_io.**read_arr_from_hdf5**(*fn_hdf5*, *folder*)

Reads array from and .hdf5 files and returns as list: Strings are returned as *np.bytes_ to str* and 'None' to None

Parameters

- **fn_hdf5** (*str*) – Filename of .hdf5 file
- **folder** (*str*) – Folder inside .hdf5 file to read

Returns
data_from_hdf5 – List containing data from .hdf5 file

Return type
list

pynibs.hdf5_io.**read_data_hdf5**(*fname*)

Reads phi and dA/dt data from .hdf5 file (phi and dAdt are given in the nodes).

Parameters
fname (*str*) – Filename of .hdf5 data file

Returns

- **phi** (*np.ndarray of float [N_nodes]*) – Electric potential in the nodes of the mesh
- **da_dt** (*np.ndarray of float [N_nodesx3]*) – Magnetic vector potential in the nodes of the mesh

pynibs.hdf5_io.**read_dict_from_hdf5**(*fn_hdf5*, *folder*)

Read all arrays from from hdf5 file and return them as dict

Parameters

- **fn_hdf5** (*str*) – Filename of .hdf5 file
- **folder** (*str*) – Folder inside .hdf5 file to read

Returns
d – Dictionary from .hdf5 file folder

Return type
dict

pynibs.hdf5_io.**simnibs_results_msh2hdf5**(*fn_msh*, *fn_hdf5*, *S*, *pos_tms_idx*, *pos_local_idx*, *subject*, *mesh_idx*, *mode_xdmf='r+'*, *n_cpu=4*, *verbose=False*, *overwrite=False*, *mid2roi=False*)

Converts simnibs .msh results file(s) to .hdf5 / .xdmf tuple.

Parameters

- **fn_msh** (*str list of str*) – Filenames (incl. path) of .msh results files from simnibs
- **fn_hdf5** (*str or list of str*) – Filenames (incl. path) of .hdf5 results files
- **S** (*Simnibs Session object*) – Simnibs Session object the simulations are conducted with
- **pos_tms_idx** (*list of int*) – Index of the simulation w.r.t. to the simnibs TMSList (inside Session object S) For every coil a separate TMSList exists, which contains multiple coil positions.
- **pos_local_idx** (*list of int*) – Index of the simulation w.r.t. to the simnibs POSlist in the TMSList (inside Session object S) For every coil a separate TMSList exists, which contains multiple coil positions.
- **subject** (*Subject object*) – Subject object loaded from .pkl file
- **mesh_idx** (*int*) – Mesh index

- **mode_xdmf** (*str, optional, default: "r+"*) – Mode to open hdf5_geo file to write xdmf. If hdf5_geo is already separated in tets and tris etc., the file is not changed, use "r" to avoid IOErrors in case of parallel computing.
- **n_cpu** (*int*) – Number of processes
- **verbose** (*bool, optional, default: False*) – Print output messages
- **overwrite** (*bool, optional, default: False*) – Overwrite .hdf5 file if existing
- **mid2roi** (*bool or string, optional, default: False*) – If the mesh contains ROIs and the e-field was calculated in the midlayer using simnibs (`S.map_to_surf = True`), the midlayer results will be mapped from the simnibs midlayer to the ROIs (takes some time for large ROIs)

Returns

<File> – .hdf5 file containing the results. An .xdmf file is also created to link the results with the mesh .hdf5 file of the subject

Return type

.hdf5 file

pynibs.hdf5_io.**simnibs_results_msh2hdf5_workhorse**(*fn_msh*, *fn_hdf5*, *S*, *pos_tms_idx*, *pos_local_idx*, *subject*, *mesh_idx*, *mode_xdmf='r+'*, *verbose=False*, *overwrite=False*, *mid2roi=False*)

Converts simnibs .msh results file to .hdf5 (including midlayer data if desired)

Parameters

- **fn_msh** (*list of str*) – Filenames (incl. path) of .msh results files from simnibs
- **fn_hdf5** (*str or list of str*) – Filenames (incl. path) of .hdf5 results files
- **S** (*Simnibs Session object*) – Simnibs Session object the simulations are conducted with
- **pos_tms_idx** (*list of int*) – Index of the simulation w.r.t. to the simnibs TMSList (inside Session object S) For every coil a separate TMSList exists, which contains multiple coil positions.
- **pos_local_idx** (*list of int*) – Index of the simulation w.r.t. to the simnibs POSlist in the TMSList (inside Session object S) For every coil a separate TMSList exists, which contains multiple coil positions.
- **subject** (*Subject object*) – Subject object loaded from .pkl file
- **mesh_idx** (*int*) – Mesh index
- **mode_xdmf** (*str, optional, default: "r+"*) – Mode to open hdf5_geo file to write xdmf. If hdf5_geo is already separated in tets and tris etc, the file is not changed, use "r" to avoid IOErrors in case of parallel computing.
- **verbose** (*bool, optional, default: False*) – Print output messages
- **overwrite** (*bool, optional, default: False*) – Overwrite .hdf5 file if existing
- **mid2roi** (*bool, list of string, or string, optional, default:False*) – If the mesh contains ROIs and the e-field was calculated in the midlayer using simnibs (`S.map_to_surf = True`), the midlayer results will be mapped from the simnibs midlayer to the ROIs (takes some time for large ROIs)

Returns

<File> – .hdf5 file containing the results. An .xdmf file is also created to link the results with the mesh .hdf5 file of the subject

Return type

.hdf5 file

pynibs.hdf5_io.**split_hdf5**(*hdf5_in_fn*, *hdf5_geo_out_fn=''*, *hdf5_data_out_fn=None*)

Splits one hdf5 into one with spatial data and one with statistical data. If coil data is present in `hdf5_in`, it is saved in `hdf5Data_out`. If new spatial data is added to file (curve, inflated, whatever), add this to the geogroups variable.

Parameters

- **hdf5_in_fn** (*str*) – Filename of .hdf5 input file
- **hdf5_geo_out_fn** (*str*) – Filename of .hdf5 .geo output file
- **hdf5_data_out_fn** (*str*) – Filename of .hdf5 .data output file (ff none, remove data from hdf5_in)

Returns

- **<File>** (*.hdf5 file*) – hdf5Geo_out_fn (spatial data)
- **<File>** (*.hdf5 file*) – hdf5Data_out_fn (data)

pynibs.hdf5_io.**write_arr_to_hdf5**(*fn_hdf5*, *arr_name*, *data*, *overwrite_arr=True*, *verbose=False*, *check_file_exist=False*)

Takes an array and adds it to an hdf5 file

If data is list of dict, `write_dict_to_hdf5()` is called for each dict with adapted hdf5-folder name Otherwise, data is casted to np.ndarray and dtype of unicode data casted to `'|S'`.

Parameters

- **fn_hdf5** (*str*) – Filename of .hdf5 file
- **arr_name** (*str*) – Complete path in .hdf5 file with array name
- **data** (*ndarray, list or dict*) – Data to write
- **overwrite_arr** (*bool, optional, default: True*) – Overwrite existing array
- **verbose** (*bool, optional, default: False*) – Print information

pynibs.hdf5_io.**write_data_hdf5**(*out_fn*, *data*, *data_names*, *hdf5_path='/data'*, *mode='a'*)

Creates a .hdf5 file with data.

Parameters

- **out_fn** (*str*) – Filename of output .hdf5 file containing the geometry information
- **data** (*np.ndarray or list of nparrays of float*) – Data to save in hdf5 data file
- **data_names** (*str or list of str*) – Labels of data
- **hdf5_path** (*str*) – Folder in .hdf5 geometry file, where the data is saved in (Default: /data)
- **mode** (*str, optional, default: "a"*) – Mode: “a” append, “w” write (overwrite)

Returns
<File> – File containing the stored data

Return type
.hdf5 file

Example

File structure of .hdf5 data file

```
data
|---/data_names[0]          [data[0]]          First dataset
|---/    ...                   ...                 ...
|---/data_names[N-1]        [data[N-1]]        Last dataset
```

pynibs.hdf5_io.**write_data_hdf5_surf**(*data*, *data_names*, *data_hdf_fn_out*, *geo_hdf_fn*, *replace=False*, *replace_array_in_file=True*)

Saves surface data to .hdf5 data file and generates corresponding .xdmf file linking both. The directory of data_hdf_fn_out and geo_hdf_fn should be the same, as only basenames of files are stored in the .xdmf file.

Parameters

- **data** (*np.ndarray or list [N_points_ROI x N_components]*) – Data to map on surfaces
- **data_names** (*str or list*) – Names for datasets
- **data_hdf_fn_out** (*str*) – Filename of .hdf5 data file
- **geo_hdf_fn** (*str*) – Filename of .hdf5 geo file containing the geometry information (has to exist)
- **replace** (*boolean, optional, default: False*) – Replace existing .hdf5 and .xdmf file completely
- **replace_array_in_file** (*boolean, optional, default: True*) – Replace existing array in file

Returns

- **<File>** (*.hdf5 file*) – data_hdf_fn_out.hdf5 containing data
- **<File>** (*.xdmf file*) – data_hdf_fn_out.xdmf containing information about .hdf5 file structure for Paraview

Example

File structure of .hdf5 data file

```
/data
|---/tris
|      |---dataset_0    [dataset_0]    (size: N_dataset_0 x M_dataset_0)
|      |---   ...
|      |---dataset_K    [dataset_K]    (size: N_dataset_K x M_dataset_K)
```

pynibs.hdf5_io.**write_dict_to_hdf5**(*fn_hdf5*, *data*, *folder*, *check_file_exist=False*, *verbose=False*)

Takes dict (from subject.py) and passes its keys to write_arr_to_hdf5()

```
fn_hdf5:folder/
             |--key1
             |--key2
             |...
```

Parameters

- **fn_hdf5** (*str*) –
- **data** (*dict or pynibs.Mesh*) –
- **folder** (*str*) –
- **verbose** (*bool*) –
- **check_file_exist** (*bool*) –

pynibs.hdf5_io.**write_geo_hdf5**(*out_fn*, *msh*, *roi_dict=None*, *hdf5_path='/mesh'*)

Creates a .hdf5 file with geometry data from mesh including region of interest(s).

Parameters

- **out_fn** (*str*) – Output hdf5 filename for mesh' geometry information.
- **msh** (pynibs.mesh.mesh_struct.TetrahedraLinear) – Mesh to write to file.
- **roi_dict** (dict of (*RegionOfInterestSurface* or RegionOfInterestVolume)) – Region of interest (surface and/or volume) information.
- **hdf5_path** (*str, default: '/mesh'*) – Path in output file to store geometry information.

Returns

<File> – File containing the geometry information

Return type

.hdf5 file

Example

File structure of .hdf5 geometry file

```
mesh
I---/elm
I    I--/elm_number             [1,2,3,...,N_ele]        Running index over all elements starting at 1
                                                            (triangles and tetrahedra)
I    I--/elm_type               [2,2,2,...,4,4]          Element type: 2 triangles, 4 tetrahedra
I    I--/tag1                [1001,1001, ..., 4,4,4]     Surface (100X) and domain (X) indices with 1000
                                                            offset for surfaces
I    I--/tag2                [   1,   1, ..., 4,4,4]     Surface (X) and domain (X) indices w/o offset
I    I--/triangle_number_list   [1,5,6;... ;1,4,8]        Connectivity of triangles [X, X, X]
I    I--/tri_tissue_type        [1,1, ..., 3,3,3]         Surface indices to differentiate between surfaces
I    I--/tetrahedra_number_list [1,5,6,7;... ;1,4,8,12]   Connectivity of tetrahedra [X, X, X, X]
I    I--/tet_tissue_type        [1,1, ..., 3,3,3]         Volume indices to differentiate between volumes
I    I--/node_number_list       [1,5,6,0;... ;1,4,8,9]    Connectivity of triangles [X, X, X, 0] and
                                                            tetrahedra [X, X, X, X]
I
I---/nodes
I    I--/node_coord          [1.254, 1.762, 1.875;...]    Node coordinates in (mm)
I    I--/node_number         [1,2,3,...,N_nodes]          Running index over all points starting at 1
I    I--/units               ['mm']                       .value is unit of geometry

roi_surface
I---/0                                                    Region of Interest number
I    I--/node_coord_up              [1.254, 1.762, 1.875;...]    Coordinates of upper surface points
I    I--/node_coord_mid             [1.254, 1.762, 1.875;...]    Coordinates of middle surface points
I    I--/node_coord_low             [1.254, 1.762, 1.875;...]    Coordinates of lower surface points
I    I--/tri_center_coord_up        [1.254, 1.762, 1.875;...]    Coordinates of upper triangle centers
I    I--/tri_center_coord_mid       [1.254, 1.762, 1.875;...]    Coordinates of middle triangle centers
I    I--/tri_center_coord_low       [1.254, 1.762, 1.875;...]    Coordinates of lower triangle centers
I    I--/node_number_list           [1,5,6,0;... ;1,4,8,9]       Connectivity of triangles [X, X, X]
I    I--/delta                      0.5                          Distance parameter between GM and WM surface
I    I--/tet_idx_tri_center_up      [183, 913, 56, ...]          Tetrahedra indices where triangle center of
                                                                    upper surface are lying in
I    I--/tet_idx_tri_center_mid     [185, 911, 58, ...]          Tetrahedra indices where triangle center of
                                                                    middle surface are lying in
I    I--/tet_idx_tri_center_low     [191, 912, 59, ...]          Tetrahedra indices where triangle center of
                                                                    lower surface are lying in
I    I--/tet_idx_node_coord_mid     [12, 15, 43, ...]            Tetrahedra indices where the node_coords_mid
                                                                    are lying in
I    I--/gm_surf_fname              .../surf/lh.pial             Filename of GM surface from segmentation
I    I--/wm_surf_fname              .../surf/lh.white            Filename of WM surface from segmentation
I    I--/layer                      3                            Number of layers
I    I--/fn_mask                    .../simnibs/mask.mgh         Filename of region of interest mask
I    I--/X_ROI                      [-10, 15]                    X limits of region of interest box
I    I--/Y_ROI                      [-10, 15]                    Y limits of region of interest box
I    I--/Z_ROI                      [-10, 15]                    Z limits of region of interest box
I
I---/1
I    I ...

roi_volume
I---/0                                                    Region of Interest number
I    I--/node_coord                 [1.254, 1.762, 1.875;...]    Coordinates (x,y,z) of ROI nodes
I    I--/tet_node_number_list       [1,5,6,7;... ;1,4,8,9]       Connectivity matrix of ROI tetrahedra
I    I--/tri_node_number_list       [1,5,6;... ;1,4,8]           Connectivity matrix of ROI triangles
I    I--/tet_idx_node_coord         [183, 913, 56, ...]          Tetrahedra indices where ROI nodes are
I    I--/tet_idx_tetrahedra_center  [12, 15, 43, ...]            Tetrahedra indices where center points of
                                                                    ROI tetrahedra are
I    I--/tet_idx_triangle_center    [12, 15, 43, ...]            Tetrahedra indices where center points of
                                                                    ROI triangles are


I---/1
I    I ...
```

pynibs.hdf5_io.**write_geo_hdf5_surf**(*out_fn*, *points*, *con*, *replace=False*, *hdf5_path='/mesh'*)

Creates a .hdf5 file with geometry data from midlayer.

Parameters

- **out_fn** (*str*) – Filename of output .hdf5 file containing the geometry information
- **points** (*np.ndarray [N_points x 3]*) – Coordinates of nodes (x,y,z)
- **con** (*np.ndarray [N_tri x 3]*) – Connectivity list of triangles
- **replace** (*boolean*) – Replace .hdf5 geometry file (True / False)
- **hdf5_path** (*str*) – Folder in .hdf5 geometry file, where the geometry information is saved in (Default: /mesh)

Returns
<File> – File containing the geometry information.

Return type
.hdf5 file

Example

File structure of .hdf5 geometry file:

```
mesh
|---/elm
|    |--/triangle_number_list   [1,5,6;... ;1,4,8]      Connectivity of triangles [X, X, X]
|    |--/tri_tissue_type        [1,1, ..., 3,3,3]       Surface indices to differentiate between surfaces
|
|---/nodes
|    |--/node_coord             [1.2, 1.7, 1.8; ...]    Node coordinates in (mm)
```

pynibs.hdf5_io.**write_temporal_xdmf**(*hdf5_fn*, *data_folder='c'*, *coil_center_folder=None*, *coil_ori_0_folder=None*, *coil_ori_1_folder=None*, *coil_ori_2_folder=None*, *coil_current_folder=None*, *hdf5_geo_fn=None*, *overwrite_xdmf=False*, *verbose=False*)

Creates .xdmf markup file for given ROI hdf5 data file with 4D data. This was written to be able to visualize data from the permutation analysis of the regression approach It expects an .hdf5 with a data group with (many) subarrays. The N subarrays name should be named from 0 to N-1 Each subarray has shape `(N_elemns, 1)`

Not tested for whole brain.

```
hdf5:/data_folder/0
                 /1
                 /2
                 /3
                 /4
                 ...
```

Parameters

- **hdf5_fn** (*str*) – Filename of hdf5 file containing the data
- **data_folder** (*str*) – Path within hdf5 to group of dataframes
- **hdf5_geo_fn** (*str (optional)*) – Filename of hdf5 file containing the geometry
- **overwrite_xdmf** (*boolean*) – Overwrite existing .xdmf file if present
- **coil_center_folder** (*str*) –
- **coil_ori_0_folder** (*str*) –
- **coil_ori_1_folder** (*str*) –
- **coil_ori_2_folder** (*str*) –
- **coil_current_folder** (*str*) –
- **verbose** (*boolean*) – Print output or not

Returns
<File> – hdf5_fn[-4].xdmf

Return type
.xdmf file

pynibs.hdf5_io.**write_xdmf**(*hdf5_fn*, *hdf5_geo_fn=None*, *overwrite_xdmf=False*, *overwrite_array=False*, *verbose=False*, *mode='r+'*)

Creates .xdmf markup file for given hdf5 file, mainly for paraview visualization. Checks if triangles and tetrahedra already exists as distinct arrays in `hdf5_fn`. If not, these are added to the .hdf5 file and rebased to 0 (from 1). If only `hdf5_fn` is provided, spatial information has to be present as arrays for tris and tets in this dataset.

Parameters

- **hdf5_fn** (*str*) – Filename of hdf5 file containing the data
- **hdf5_geo_fn** (*str*) – Filename of hdf5 file containing the geometry. Optional.
- **overwrite_xdmf** (*bool*) – Overwrite existing xdmf file if present. Default: False.
- **overwrite_array** (*bool*) – Overwrite existing arrays if present. Default: False.
- **verbose** (*boolean*) – Print output or not
- **mode** (*str, optional, default: "r+"*) – Mode to open hdf5_geo file. If hdf5_geo is already separated in tets and tris etc., nothing has to be written, use "r" to avoid IOErrors in case of parallel computing.

Returns

- **fn_xml** (*str*) – Filename of the created .xml file
- **<File>** (*.xdmf file*) – hdf5_fn[-4].xdmf (only data if hdf5Geo_fn provided)
- **<File>** (*.hdf5 file*) – hdf5_fn changed if neccessary
- **<File>** (*.hdf5 file*) – hdf5geo_fn containing spatial data

1.1.6 pynibs.muap module

pynibs.muap.**calc_mep_wilson**(*firing_rate_in*, *t*, *Qvmax=900*, *Qmmax=300*, *q=8*, *Tmin=14*, *N=100*, *M0=42*, *lam=0.002*, *tau0=0.006*)

Determine motor evoked potential from incoming firing rate

Parameters

- **firing_rate_in** (*ndarray of float [n_t]*) – Input firing rate from alpha motor neurons
- **t** (*ndarray of float [n_t]*) – Time axis in s
- **Qvmax** (*float, optional, default: 900*) – Max of incoming firing rate [1/s]
- **Qmmax** (*float, optional, default: 300*) – Max of MU firing rate [1/s]
- **q** (*float, optional, default: 8*) – Min firing rate of MU [1/s]
- **Tmin** (*float, optional, default: 14*) – Min MU threshold [1/s]
- **N** (*float, optional, default: 100*) – Number of MU
- **M0** (*float, optional, default: 42*) – Scaling constant of MU amplitude [mV/s]
- **lam** (*float, optional, default: 0.002*) – MUAP timescale of first order Hermite Rodriguez function [s]
- **tau0** (*float, optional, default: 0.006*) – Standard shift of MUAP to ensure causality [s]

Returns
mep – Motor evoked potential at surface electrode

Return type
ndarray of float [n_t]

pynibs.muap.**compute_signal**(*signal_matrix*, *sensor_matrix*)

Determine average signal from one single muscle fibre on all point electrodes

Parameters

- **signal_matrix** (*ndarray of float [n_time x n_fibre]*) – Signal matrix containing the action potential values for each time step in the rows
- **sensor_matrix** (*ndarray of float [n_fibre x n_ele]*) – Sensor matrix containing the inverse distances weighted with the anisotropy of muscle tissue

Returns
signal – Average signal detected all point electrodes

Return type
ndarray of float [n_time]

pynibs.muap.**create_electrode**(*l_x*, *l_z*, *n_x*, *n_z*)

Creates electrode coordinates

Parameters

- **l_x** (*float*) – X-extension of electrode in mm
- **l_z** (*float*) – Z-extension of electrode in mm
- **n_x** (*int*) – Number of point electrode in x-direction
- **n_z** (*int*) – Number of point electrodes in z-direction

Returns
electrode_coords – Coordinates of point electrodes (x, y, z)

Return type
ndarray of float [n_ele x 3]

pynibs.muap.**create_muscle_coords**(*l_x*, *l_y*, *n_x*, *n_y*, *h*)

Create x and y coordinates of muscle fibres in muscle

Parameters

- **l_x** (*float*) – X-extension of muscle in mm
- **l_y** (*float*) – Y-extension of muscle in mm
- **n_x** (*int*) – Number of muscle fibres in x-direction
- **n_y** (*int*) – Number of muscle fibres in y-direction
- **h** (*float*) – Offset of muscle from electrode plane in mm

Returns
muscle_coords – Coordinates of muscle fibres in x-y plane (x, y, z)

Return type
ndarray of float [n_muscle x 3]

pynibs.muap.**create_muscle_fibre**(*x0*, *y0*, *L*, *n_fibre*)

Creates muscle fibre coordinates (in z-direction)

Parameters

- **x0** (*float*) – X-location of muscle fibre
- **y0** (*float*) – Y-location of muscle fibre

- **L** (*float*) – Length of muscle fibre
- **n_fibre** (*float*) – Number of discrete fibre elements

Returns
fibre_coords – Coordinates of muscle fibre in z-direction (x, y, z)

Return type
ndarray of float [n_fibre x 3]

pynibs.muap.**create_sensor_matrix**(*electrode_coords*, *fibre_coords*, *sigma_r=1*, *sigma_z=1*)

Create sensor matrix containing the inverse distances from the point electrodes to the fibre elements weighted by the anisotropy factor of the muscle tissue.

Parameters

- **electrode_coords** (*ndarray of float [n_ele x 3]*) – Coordinates of point electrodes (x, y, z)
- **fibre_coords** (*ndarray of float [n_fibre x 3]*) – Coordinates of muscle fibre in z-direction (x, y, z)
- **sigma_r** (*float, optional, default: 1*) – Radial conductivity of muscle
- **sigma_z** (*float, optional, default: 1*) – Axial conductivity of muscle along fibre

Returns
sensor_matrix – Sensor matrix containing the inverse distances weighted with the anisotropy of muscle tissue

Return type
ndarray of float [n_fibre x n_ele]

pynibs.muap.**create_signal_matrix**(*T*, *dt*, *fibre_coords*, *z_e*, *v*)

Create signal matrix containing the travelling action potential on the fibre

Parameters

- **T** (*float*) – Total time
- **dt** (*float*) – Time step
- **fibre_coords** (*ndarray of float [n_fibre x 3]*) – Coordinates of muscle fibre in z-direction (x, y, z)
- **z_e** (*float*) – Location of action potential generation
- **v** (*float*) – Velocity of action potential

Returns
signal_matrix – Signal matrix containing the action potential values for each time step in the rows

Return type
ndarray of float [n_time x n_fibre]

pynibs.muap.**dipole_potential**(*z*, *loc*, *response*)

Returns dipole potential at given coordinates z (interpolates given dipole potential)

pynibs.muap.**hermite_rodriguez_1st**(*t*, *tau0=0*, *tau=0*, *lam=0.002*)

First order Hermite Rodriguez function to model surface MUAPs

Parameters

- **t** (*ndarray of float [n_t]*) – Time axis in s
- **tau0** (*float, optional, default: 0*) – initial shift to ensure causality in s
- **tau** (*float, optional, default: 0*) – shift (firing time) in s

- **lam** (*float, optional, default: 2*) – Timescale in s

Returns
y – Surface MUAP

Return type
ndarray of float [n_t]

pynibs.muap.**sfap**(*z*, *sigma_i=1.01*, *d=5.4999999999999995e-05*, *alpha=0.5*)

Single fibre propagating transmembrane current (second spatial derivative of transmembrane potential).

S. D. Nandedkar and E. V. Stalberg,"Simulation of single musclefiber action potentials" Med. Biol. Eng. Comput., vol. 21, pp. 158–165, Mar.1983.

J. Duchene and J.-Y. Hogrel,"A model of EMG generation," IEEETrans. Biomed. Eng., vol. 47, no. 2, pp. 192–200, Feb. 2000

Hamilton-Wright, A., & Stashuk, D. W. (2005). Physiologically based simulation of clinical EMG signals. IEEE Transactions on biomedical engineering, 52(2), 171-183.

Parameters

- **t** (*ndarray of float [n_t]*) – Time in (ms)
- **sigma_i** (*float, optional, default: 1.01*) – Intracellular conductivity in (S/m)
- **d** (*float, optional, default: 55*1e-6*) – Diameter of muscle fibre in (m)
- **v** (*float, optional, default: 1*) – Conduction velocity in (m/s)
- **alpha** (*float, optional, default: 0.5*) – Scaling factor to adjust length of AP

Returns
i – Transmembrane current of muscle fibre

Return type
ndarray of float [n_t]

pynibs.muap.**sfap_dip**(*z*)

pynibs.muap.**weight_signal_matrix**(*signal_matrix*, *fn_imp*, *t*, *z*)

Weight signal matrix with impulse response from single dipole at every location

1.1.7 pynibs.opt module

pynibs.opt.**get_det_fim**(*x*, *fun*, *p*, *fim_matrix*)

Updates the Fisher Information Matrix and returns the negative determinant based on the sample x. It is a score how much information the additional sample yields.

Parameters

- **fun** (*function object*) – Function object defined in interval [0, 1].
- **x** (*float*) – Single sample location (interval [0, 1])
- **p** (*dict*) – Dictionary containing the parameter estimates. The keys are the parameter names of fun.
- **fim_matrix** (*ndarray of float [n_params x n_params]*) – Fisher Information Matrix

Returns
det – Determinant of the Fisher Information Matrix after adding sample x

Return type
float

pynibs.opt.**get_fim_sample**(*fun*, *x*, *p*)

Get Fisher Information Matrix of one single sample.

Parameters

- **fun** (*function object*) – Function object the fisher information matrix is calculated for. The sample is passed as the first argument.
- **x** (*float*) – Sample passed to function
- **p** (*dict*) – Dictionary containing the parameter estimates. The keys are the parameter names of fun.

Returns

fim_matrix – Fisher information matrix

Return type

ndarray of float [n_params x n_params]

pynibs.opt.**get_optimal_coil_positions**(*e_matrix*, *criterion*, *n_stim*, *ele_idx_1=None*, *ele_idx_2=None*, *fn_out_hdf5=None*, *n_cpu=4*, *zap_idx_opt=None*, *fun=None*, *p=None*, *c=None*, *weights=None*, *overwrite=True*, *verbose=True*, *fn_coilpos_hdf5=None*, *start_zap_idx=0*)

Determine set of optimal coil positions for TMS regression analysis.

Parameters

- **e_matrix** (*ndarray of float [n_stim, n_ele]*) – Matrix containing the electric field values in the ROI
- **criterion** (*str*) – Optimization criterion: - "mc_cols": Minimization of mutual coherence between columns - "mc_rows": Minimization of mutual coherence between rows - "svd": Minimization of condition number - "dist": Equal distant sampling - "dist_svd": Minimization of condition number and equidistant sampling - "dist_mc_cols": Minimization of mutual coherence between columns and equidistant sampling - "dist_mc_rows": Minimization of mutual coherence between rows and equidistant sampling - "coverage": Maximizes the electric field coverage - "variability": Maximizes variability between elements
- **n_stim** (*int*) – Maximum number of stimulations
- **ele_idx_1** (*ndarray of int, optional, default: None*) – Element indices the first optimization goal is performed for, If None, all elements are consiered
- **ele_idx_2** (*ndarray of int, optional, default: None*) – Element indices the first optimization goal is performed for. If None, all elements are consiered
- **n_cpu** (*int*) – Number of threads
- **fn_out_hdf5** (*str, optional, default: None*) – Returns the list of optimal zap indices if fn_out_hdf5 is None, otherwise, save the results in .hdf5 file. Filename of output .hdf5 file where the zap index lists are saved in subfolder "zap_index_lists" - "zap_index_lists/0": [213] - "zap_index_lists/1": [213, 5] - etc
- **zap_idx_opt** (*list of int, optional, default: None*) – List of already selected optimal coil positions (those are ignored in the optimization and will not be picked again)
- **fun** (*function object*) – Function object defined in interval [0, 1]. (only needed for fim optimization)
- **p** (*list of dict [n_ele], optional, default: None*) – List of dicts containing the parameter estimates (whole ROI). The keys are the parameter names of fun. (only needed for fim and dist optimization)

- **c** (*ndarray of float [n_ele], optional, default: None*) – Congruence factor in each ROI element. Used to weight fim and dist optimization. (only needed for fim and dist optimization)
- **weights** (*list of float [2], optional, default: [0.5, 0.5]*) – Weights of optimization criteria in case of multiple goal functions (e.g. fim_svd). Higher weight means higher importance for the respective criteria. By default both optimization criteria are weighted equally [0.5, 0.5].
- **overwrite** (*bool, optional, default: True*) – Overwrite existing solutions or read existing hdf5 file and continue optimization
- **verbose** (*bool, optional, default: True*) – Print output messages
- **fn_coilpos_hdf5** (*str*) – File containing the corresponding coil positions and orientations (centers, m0, m1, m2)
- **start_zap_idx** (*int, optional, default: 0*) – First zap index to start greedy search

Returns

- **zap_idx_e_opt** (*list of int [n_stim]*) – Optimal zap indices
- *<File> .hdf5 file* – Output file containing the zap index lists

pynibs.opt.**get_optimal_sample_fim**(*fun*, *p*, *x=None*)

Determines optimal location of next sample by maximizing the determinant of the Fisher Information Matrix.

Parameters

- **fun** (*function object*) – Function object (interval [0, 1]).
- **x** (*ndarray of float, optional, default: None*) – Previous sample locations (interval [0, 1]).
- **p** (*dict*) – Dictionary containing the parameter estimates. The keys are the parameter names of fun.

Returns

x_opt – Optimal location of next sample (interval [0, 1]).

Return type

float

pynibs.opt.**init_fim_matrix**(*fun*, *x*, *p*)

Initializes the Fisher Information Matrix based on the samples given in x.

Parameters

- **fun** (*function object*) – Function object defined in interval [0, 1].
- **x** (*ndarray of float*) – Initial sample locations (interval [0, 1])
- **p** (*dict*) – Dictionary containing the parameter estimates. The keys are the parameter names of fun.

Returns

fim_matrix – Fisher Information Matrix

Return type

ndarray of float [n_params x n_params]

pynibs.opt.**online_optimization**(*fn_subject_hdf5*, *fn_roi_ss_indices_hdf5*, *fn_out_hdf5*, *fn_stimsites_hdf5*, *e_matrix*, *mep*, *mesh_idx*, *roi_idx*, *n_zaps_init=3*, *criterion_init='mc_rows'*, *criterion='coverage'*, *n_cpu=4*, *threshold=0.8*, *weights=None*, *eps0=0.01*, *eps0_dist=1*, *exponent=5*, *perc=99*, *n_refit=0*, *fun=<function sigmoid>*, *verbose=True*)

Performs virtual online optimization to determine the congruence factor. After an initial set of coil positions, the algorithm iteratively optimizes the next coil position based on the virtually measured MEP data.

Parameters

- **fn_subject_hdf5** (*str*) – Filename of subject .hdf5 file
- **fn_roi_ss_indices_hdf5** (*str*) – Filename of .hdf5 file containing the element indices of the subsampled ROI in f["roi_indices"]
- **e_matrix** (*ndarray of float [n_zaps x n_ele]*) – Electric field matrix
- **mep** (*ndarray of float [n_zaps]*) – Motor evoked potentials for every stimulation
- **fn_out_hdf5** (*str*) – Filename of .hdf5 output file containing the coil positions and the congruence factor maps for every iteration
- **fn_stimsites_hdf5** (*str*) – Filename of the .hdf5 file containing the stimulation sites in "centers", "m0", "m1", "m2"
- **mesh_idx** (*int*) – Mesh index
- **roi_idx** (*int*) – ROI index
- **n_zaps_init** (*int, optional, default: 3*) – Number of initial samples optimized using optimization criterion specified in "criterion_init"
- **criterion_init** (*str, optional, default: "mc_rows"*) – Optimization criterion for which the initial samples are optimized (e.g. "mc_rows", "svd", ...)
- **criterion** (*str, optional, default: "coverage"*) – Optimization criterion for which the online optimization is performed (e.g. "coverage", "mc_rows", "svd", ...)
- **n_cpu** (*int, optional, dfault: 4*) – Number of CPU cores to use
- **threshold** (*float, optional, default: 0.1*) – Threshold between [0 ... 1] of the maximal congruence factor. Elements where c > threshold * max(c) are included in the online optimization to select the next optimal coil position.
- **weights** (*list of float [2], optional, default: [0.5, 0.5]*) – Weights of optimization criteria in case of multiple goal functions (e.g. fim_svd). Higher weight means higher importance for the respective criteria. By default both optimization criteria are weighted equally [0.5, 0.5].
- **eps0** (*float, optional, default: 0.01*) – First error threshold to terminate the online optimization. The normalized root mean square deviation is calculated between the current and the previous solution. If the error is lower than eps0 for 3 times in a row, the online optimization terminates and returns the results.
- **eps0_dist** (*float, optional, default: 1*) – Second error threshold to terminate the online optimization. The geodesic distance in mm of the hotspot is calculated between the current and the previous solution. If the error is lower than eps0_dist for 3 times in a row, the online optimization terminates and returns the results.
- **exponent** (*float, optional, default: 5*) – Exponent the congruence factor map is scaled c**exponent
- **perc** (*float, optional, default: 99*) – Percentile the congruence factor map is normalized (between 0 and 100)
- **n_refit** (*int, optional, default: 0*) – Number of refit iterations. No refit is applied if n_refit=0.
- **fun** (*function object, optional, default: pynibs.linear*) – Function to use to determine the congruence factor (e.g. pynibs.linear, pynibs.sigmoid, ...)
- **verbose** (*bool, optional, default: True*) – Plot output messages

Returns
Results output file containing the coil positions and the congruence factor maps for every iteration

Return type
<file> .hdf5 file

pynibs.opt.**workhorse_corr**(*idx_list*, *array*, *ele_idx_1*)

Parameters

- **idx_list** –
- **array** –
- **ele_idx_1** –

Returns

pynibs.opt.**workhorse_coverage**(*idx_list*, *array*, *x*, *y*, *ele_idx_1*)

Determine coverage score (likelihood) for given zap indices in idx_list

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **x** (*ndarray of float [200 x n_ele]*) – x-values of coverage distributions, defined in interval [0, 1] (element wise normalized electric field)
- **y** (*ndarray of float [200 x n_ele]*) – y-values of coverage distributions (element wise probability of already included e-fields)
- **ele_idx_1** (*ndarray of float [n_roi]*) – Element indices for which the coverage optimization is performed for

Returns
res – Coverage score (likelihood) for given electric field combinations. Lower values indicate that the new zap fills a gap which was not covered before.

Return type
ndarray of float [n_combs]

pynibs.opt.**workhorse_coverage_prepare**(*idx_list*, *array*, *zap_idx*)

Prepares coverage calculation. Determines coverage distributions for elements in idx_list given the zaps in zap_idx

Parameters

- **idx_list** (*list [n_ele]*) – Index lists of elements.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **zap_idx** (*ndarray of int*) – Included zaps in coverage distribution.

Returns

- **x** (*ndarray of float [200 x n_ele]*) – x-values of coverage distributions, defined in interval [0, 1] (element wise normalized electric field)
- **y** (*ndarray of float [200 x n_ele]*) – y-values of coverage distributions (element wise probability of already included e-fields)

pynibs.opt.**workhorse_dist**(*idx_list*, *array*, *ele_idx_1*)

Determines distance score for given zap indices in idx_list.

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **ele_idx_1** (*ndarray of float [n_ele]*) – Element indices for which the optimization is performed

Returns

res – Distance based score. Lower values indicate more equidistant sampling (better)

Return type

ndarray of float [n_combs]

pynibs.opt.**workhorse_dist_mc**(*idx_list*, *array*, *ele_idx_1*, *ele_idx_2*, *mode='cols'*)

Determines distance score and mutual coherence for given zap indices in idx_list. If c_max_idx is given, the distance based score is calculated only for this element. The condition number however is optimized for all elements in array

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **mode** (*str, optional, default: "cols"*) – Set if the mutual coherence is calculated w.r.t. columns or rows ("cols", "rows")
- **ele_idx_1** (*ndarray of float [n_ele]*) – Element indices for which the dist optimization is performed for
- **ele_idx_2** (*ndarray of float [n_ele]*) – Element indices for which the mc optimization is performed for

Returns

- **res_dist** (*ndarray of float [n_combs]*) – Distance based score. Lower values indicate more equidistant sampling (better)
- **res_mc** (*ndarray of float [n_combs]*) – Mutual coherence. Lower values indicate more orthogonal e-field combinations (better)

pynibs.opt.**workhorse_dist_svd**(*idx_list*, *array*, *ele_idx_1*, *ele_idx_2*)

Determines distance score and condition number for given zap indices in idx_list. If c_max_idx is given, the distance based score is calculated only for this element. The condition number however is optimized for all elements in array

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **ele_idx_1** (*ndarray of float [n_ele]*) – Element indices for which the dist optimization is performed for
- **ele_idx_2** (*ndarray of float [n_ele]*) – Element indices for which the svd optimization is performed for

Returns

- **res_dist** (*ndarray of float [n_combs]*) – Distance based score. Lower values indicate more equidistant sampling (better)
- **res_svd** (*ndarray of float [n_combs]*) – Condition number. Lower values indicate more orthogonal e-field combinations (better)

pynibs.opt.**workhorse_fim**(*idx_list*, *array*, *ele_idx_1*, *e_opt*, *c=None*)

Determine difference between e-fields and optimal e-field determined using the Fisher Information Matrix.

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **ele_idx_1** (*ndarray of float [n_roi]*) – Element indices for which the fim optimization is performed for
- **e_opt** (*ndarray of float [n_roi]*) – Optimal electric field value(s) (target) determined by FIM method
- **c** (*ndarray of float [n_ele], optional, default: None*) – Congruence factor map normalized to 1 (whole ROI) used to weight the difference between the optimal e-field and the candidate e-field. If None, no weighting is applied.

Returns

res – Difference between e-fields and optimal e-field.

Return type

ndarray of float [n_combs]

pynibs.opt.**workhorse_fim_mc**(*idx_list*, *array*, *ele_idx_1*, *ele_idx_2*, *e_opt*, *c=None*, *mode='rows'*)

Determine difference between e-fields and optimal e-field determined using the Fisher Information Matrix and mutual coherence.

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **ele_idx_1** (*ndarray of float [n_roi_1]*) – Element indices for which the fim optimization is performed for
- **ele_idx_2** (*ndarray of float [n_roi_2]*) – Element indices for which the mc optimization is performed for
- **e_opt** (*float*) – Optimal electric field value (target) determined by FIM method
- **c** (*ndarray of float [n_ele], optional, default: None*) – Congruence factor map normalized to 1 (whole ROI) used to weight the difference between the optimal e-field and the candidate e-field. If None, no weighting is applied.

Returns

- **res_fim** (*ndarray of float [n_combs]*) – Difference between e-fields and optimal e-field.
- **res_mc** (*ndarray of float [n_combs]*) – Mutual coherence. Lower values indicate more orthogonal e-field combinations (better)

pynibs.opt.**workhorse_fim_svd**(*idx_list*, *array*, *ele_idx_1*, *ele_idx_2*, *e_opt*, *c=None*)

Determine difference between e-fields and optimal e-field determined using the Fisher Information Matrix and condition number.

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **ele_idx_1** (*ndarray of float [n_roi_1]*) – Element indices for which the fim optimization is performed for
- **ele_idx_2** (*ndarray of float [n_roi_2]*) – Element indices for which the svd optimization is performed for
- **e_opt** (*float*) – Optimal electric field value (target) determined by FIM method
- **c** (*ndarray of float [n_ele], optional, default: None*) – Congruence factor map normalized to 1 (whole ROI) used to weight the difference between the optimal e-field and the candidate e-field. If None, no weighting is applied.

Returns

- **res_fim** (*ndarray of float [n_combs]*) – Difference between e-fields and optimal e-field.
- **res_svd** (*ndarray of float [n_combs]*) – Condition number. Lower values indicate more orthogonal e-field combinations (better)

pynibs.opt.**workhorse_mc**(*idx_list*, *array*, *ele_idx_1*, *mode='cols'*)

Determines mutual coherence for given zap indices in idx_list.

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **ele_idx_1** (*ndarray of float [n_ele]*) – Element indices for which the optimization is performed
- **mode** (*str, optional, default: "cols"*) – Set if the mutual coherence is calculated w.r.t. columns or rows ("cols", "rows")

Returns
res – Mutual coherence. Lower values indicate more orthogonal e-field combinations (better)

Return type
ndarray of float [n_combs]

pynibs.opt.**workhorse_smooth**(*idx_list*, *array*, *ele_idx_1*)

pynibs.opt.**workhorse_svd**(*idx_list*, *array*, *ele_idx_1*)

Determines condition number for given zap indices in idx_list.

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **ele_idx_1** (*ndarray of float [n_ele]*) – Element indices for which the optimization is performed

Returns
res – Condition number. Lower values indicate more orthogonal e-field combinations (better)

Return type
ndarray of float [n_combs]

pynibs.opt.**workhorse_var**(*idx_list*, *array*, *ele_idx_1*)

pynibs.opt.**workhorse_variability**(*idx_list*, *array*, *ele_idx_1*)

Determines variability score for given zap indices in idx_list.

Parameters

- **idx_list** (*list of lists [n_combs][n_zaps]*) – Index lists of zaps containing different possible combinations. Usually only the last index changes.
- **array** (*ndarray of float [n_zaps x n_ele]*) – Electric field for different coil positions and elements
- **ele_idx_1** (*ndarray of float [n_ele]*) – Element indices for which the optimization is performed

Returns
res – Condition number. Lower values indicate more orthogonal e-field combinations (better)

Return type
ndarray of float [n_combs]

1.1.8 pynibs.para module

pynibs.para.**ResetSession**()

Resets Paraview session (needed if multiple plots are generated successively)

pynibs.para.**b2rcw**(*cmin_input*, *cmax_input*)

BLUEWHITERED Blue, white, and red color map. This function is designed to generate a blue to red colormap. The color of the colorbar is from blue to white and then to red, corresponding to the data values from negative to zero to positive, respectively. The color white always correspondes to value zero. The brightness of blue and red will change according to your setting, so that the brightness of the color corresponded to the color of his opposite number. e.g. b2rcw(-3,6) is from light blue to deep red e.g. b2rcw(-3,3) is from deep blue to deep red

Parameters

- **cmin_input** (*float*) – Minimum value of data
- **cmax_input** (*float*) – Maximum value of data

Returns
newmap

Return type
nparray of float [N_RGB x 3]

pynibs.para.**create_plot_settings_dict**(*plotfunction_type*)

Creates a dictionary with default plotsettings.

Parameters
plotfunction_type (*str*) – Plot function the dictionary is generated for:

- 'surface_vector_plot'
- 'surface_vector_plot_vtu'
- 'volume_plot'
- 'volume_plot_vtu'

Returns

- **ps** (*dict*) – Dictionary containing the plotsettings:
- **axes** (*boolean*) – Show orientation axes (TRUE / FALSE)
- **background_color** (*nparray [1 x 3]*) – Set background color of exported image RGB (0…1)
- **calculator** (*str*) – Format string with placeholder of the calculator expression the quantity to plot is modified with (e.g.: "{}^5")
- **clip_coords** (*nparray of float [N_clips x 3]*) – Coordinates of clip surface origins (x,y,z)
- **clip_normals** (*nparray of float [N_clips x 3]*) – Surface normals of clip surfaces pointing in the direction where the volume is kept for clip_type = ['clip' …] (x,y,z)
- **clip_type** (*list of str*) – Type of clipping:
  - 'clip': cut geometry but keep volume behind
  - 'slice': cut geometry and keep only the slice
- **coil_dipole_scaling** (*list [1 x 2]*) – Specify the scaling type of the dipoles (2 entries):

  coil_dipole_scaling[0]:
  - 'uniform': uniform scaling, i.e. all dipoles have the same size
  - 'scaled': size scaled according to dipole magnitude

  coil_dipole_scaling[1]:
  - scalar scale parameter of dipole size
- **coil_dipole_color** (*str or list*) – Color of the dipoles; either str to specify colormap (e.g. 'jet') or list of RGB values [1 x 3] (0…1)
- **coil_axes** (*boolean*) – Plot coil axes visualizing the principle direction and orientation of the coil (Default: True)
- **colorbar_label** (*str*) – Label of plotted data close to colorbar
- **colorbar_position** (*list of float [1 x 2]*) – Position of colorbar (lower left corner) 0…1 [x_pos, y_pos]
- **colorbar_orientation** (*str*) – Orientation of colorbar ('Vertical', 'Horizontal')
- **colorbar_aspectratio** (*int*) – Aspectratio of colorbar (higher values make it thicker)
- **colorbar_titlefontsize** (*float*) – Fontsize of colorbar title
- **colorbar_labelfontsize** (*float*) – Fontsize of colorbar labels (numbers)
- **colorbar_labelformat** (*str*) – Format of colorbar labels (e.g.: '%-#6.3g')
- **colorbar_numberoflabels** (*int*) – maximum number of colorbar labels
- **colorbar_labelcolor** (*list of float [1 x 3]*) – Color of colorbar labels in RGB (0…1)
- **colormap** (*str or nparray*) – if nparray [1 x 4*N]: custom colormap providing data and corresponding RGB values

$$\begin{bmatrix} data_1 & R_1 & G_1 & B_1 \\ data_2 & R_2 & G_2 & B_2 \\ ... & ... & ... & ... \\ data_N & R_N & G_N & B_N \end{bmatrix}$$

  if str: colormap of plotted data chosen from included presets:
  - 'Cool to Warm',
  - 'Cool to Warm (Extended)',
  - 'Blue to Red Rainbow',

  - 'X Ray',
  - 'Grayscale',
  - 'jet',
  - 'hsv',
  - 'erdc_iceFire_L',
  - 'Plasma (matplotlib)',
  - 'Viridis (matplotlib)',
  - 'gray_Matlab',
  - 'Spectral_lowBlue',
  - 'BuRd'
  - 'Rainbow Blended White'
  - 'b2rcw'
- **colormap_categories** (*boolean*) – Use categorized (discrete) colormap
- **datarange** (*list [1 x 2]*) – Minimum and Maximum of plotted datarange [MIN, MAX] (default: automatic)
- **domain_IDs** (*int or list of int*) – Domain IDs

  surface plot: Index of surface where the data is plotted on (Default: 0)

  volume plot: Specify the domains IDs to show in plot (default: all) Attention! Has to be included in the dataset under the name 'tissue'! e.g. for SimNIBS:

  - 1 -> white matter (WM)
  - 2 -> grey matter (GM)
  - 3 -> cerebrospinal fluid (CSF)
  - 4 -> skull
  - 5 -> skin
- **domain_label** (*str*) – Label of the dataset which contains the domain IDs (default: 'tissue_type')
- **edges** (*boolean*) – Show edges of mesh (TRUE / FALSE)
- **fname_in** (*str or list of str*) – Filenames of input files, 2 possibilities:
  - .xdmf-file: filename of .xmdf (needs the corresponding .hdf5 file(s) in the same folder)
  - .hdf5-file(s): filename(s) of .hdf5 file(s) containing the data and the geometry. The data can be provided in the first hdf5 file and the geometry can be provided in the second file. However, both can be also provided in a single hdf5 file.
- **fname_png** (*str*) – Name of output .png file (incl. path)
- **fname_vtu_volume** (*str*) – Name of .vtu volume file containing volume data (incl. path)
- **fname_vtu_surface** (*str*) – Name of .vtu surface file containing surface data (incl. path) (to distinguish tissues)
- **fname_vtu_coil** (*str*) – Name of coil .vtu file (incl. path) (optional)
- **info** (*str*) – Information about the plot the settings are belonging to
- **interpolate** (*boolean*) – Interpolate data for visual smoothness (TRUE / FALSE)
- **NanColor** (*list of float [3]*) – RGB color values for "Not a Number" values (range 0 … 1)

- **opacitymap** (*nparray*) – Points defining the piecewise linear opacity transfer function (transparency) (default: no transparency) connecting data values with opacity (alpha) values ranging from 0 (max. transparency) to 1 (no transparency)

$$\begin{bmatrix} data_1 & opac_1 & 0.5 & 0 \\ data_2 & opac_2 & 0.5 & 0 \\ ... & ... & ... & ... \\ data_N & opac_N & 0.5 & 0 \end{bmatrix}$$

- **plot_function** (*str*) – Function the plot is generated with:
  - 'surface_vector_plot'
  - 'surface_vector_plot_vtu'
  - 'volume_plot'
  - 'volume_plot_vtu'
- **png_resolution** (*float*) – Resolution parameter of output image (1...5)
- **quantity** (*str*) – Label of magnitude dataset to plot
- **surface_color** (*nparray [1 x 3]*) – Color of brain surface in RGB (0...1) for better visability of tissue borders
- **surface_smoothing** (*bool*) – Smooth the plotted surface (True/False)
- **show_coil** (*boolean*) – show coil if present in dataset as block termed 'coil' (Default: True)
- **vcolor** (*nparray of float [N_vecs x 3]*) – Array containing the RGB values between 0...1 of the vector groups in dataset to plot
- **vector_mode** (*dict*) – dict key determines the type how many vectors are shown: - 'All Points' - 'Every Nth Point' - 'Uniform Spatial Distribution'

  dict value (int) is the corresponding number of vectors
  - 'All Points' (not set)
  - 'Every Nth Point' (every Nth vector is shown in the grid)
  - 'Uniform Spatial Distribution' (not set)
- **view** (*list*) – Camera position and angle [[3 x CameraPosition], [3 x CameraFocalPoint], [3 x CameraViewUp], 1 x CameraParallelScale]
- **viewsize** (*nparray [1 x 2]*) – Set size of exported image in pixel [width x height] will be extra scaled by parameter png_resolution
- **vlabels** (*list of str*) – Labels of vector datasets to plot (other present datasets are ignored)
- **vscales** (*list of float*) – Scale parameters of vector groups to plot
- **vscale_mode** (*list of str [N_vecs x 1]*) – List containing the type of vector scaling:
  - 'off': all vectors are normalized
  - 'vector': vectors are scaled according to their magnitudeeee

pynibs.para.**crop_data_hdf5_to_datarange**(*ps*)

Crops the data (quantity) in .hdf5 data file to datarange and overwrites the original .hdf5 data file pointed by the .xdmf file.

Parameters

ps (*dict*) – Plot settings dictionary created with create_plotsettings_dict(plot_function)

Returns

- **fn_hdf5** (*str*) – Filename (incl. path) of data .hdf5 file (read from .xmdf file)

- **<File>** (*.hdf5 file*) – *_backup.hdf5 backup file of original .hdf5 data file
- *<File> .hdf5 file* – Cropped data

pynibs.para.**crop_image**(*fname_image*, *fname_image_cropped*)

Remove surrounding empty space around an image. This implemenation assumes that the surrounding space has the same colour as the top leftmost pixel.

Parameters
fname_image (*str*) – Filename of image to be cropped

Returns
<File> – Cropped image file saved as "fname_image_cropped"

Return type
.png file

pynibs.para.**surface_vector_plot**(*ps*)

Generate plot with Paraview from data in .hdf5 file(s).

Parameters
ps (*dict*) – Plot settings dict initialized with create_plot_settings_dict(plotfunction_type='surface_vector_plot')

Returns
<File> – Generated plot

Return type
.png file

pynibs.para.**surface_vector_plot_vtu**(*ps*)

Generate plot with Paraview from data in .vtu file.

Parameters
ps (*dict*) – Plot settings dict initialized with create_plot_settings_dict(plotfunction_type='surface_vector_plot_vtu')

Returns
<File> – Generated plot

Return type
.png file

pynibs.para.**volume_plot**(*ps*)

Generate plot with Paraview from data in .hdf5 file.

Parameters
ps (*dict*) – Plot settings dict initialized with create_plot_settings_dict(plotfunction_type="volume_plot")

Returns
<File> – Generated plot

Return type
.png file

pynibs.para.**volume_plot_vtu**(*ps*)

Generate plot with Paraview from data in .vtu file.

Parameters
ps (*dict*) – Plot settings dict initialized with create_plot_settings_dict(plotfunction_type="volume_plot_vtu")

Returns
<File> – Generated plot

Return type
.png file

pynibs.para.**write_vtu**(*fname*, *data_labels*, *points*, *connectivity*, *idx_start*, *data*)

Writes data in tetrahedra centers into .vtu file, which can be loaded with Paraview.

Parameters

- **fname** (*str*) – Name of .vtu file (incl. path)
- **data_labels** (*list with N_data str*) – Label of each dataset
- **points** (*array of float [N_points x 3]*) – Coordinates of vertices
- **connectivity** (*array of int [N_tet x 4]*) – Connectivity of points forming tetrahedra
- **idx_start** (*int*) – Smallest index in connectivity matrix, defines offset w.r.t Python indexing, which starts at '0'
- ***data** (*array(s) [N_tet x N_comp(N_data)]*) – Arrays containing data in tetrahedra center multiple components per dataset possible e.g. [Ex, Ey, Ez]

Returns

<File> – Geometry and data information

Return type

.vtu file

pynibs.para.**write_vtu_coilpos**(*fname_geo*, *fname_vtu*)

Read dipole data of coil (position and magnitude of each dipole) from geo file and store it as vtu file.

Parameters

- **fname_geo** (*str*) – .geo file from SimNIBS.
- **fname_vtu** (*str*) – .vtu output file. Nodes and nodedata.

Returns

<File> – Magnetic dipoles of the TMS coil

Return type

.vtu file

pynibs.para.**write_vtu_mult**(*fname*, *data_labels*, *points*, *triangles*, *tetrahedras*, *idx_start*, **data*)

Writes data in triangles and tetrahedra centers into .vtu file, which can be loaded with Paraview.

Parameters

- **fname** (*str*) – Name of .vtu file (incl. path)
- **data_labels** (*list of str [N_data]*) – Label of each dataset
- **points** (*nparray of float [N_points x 3]*) – Coordinates of vertices
- **triangles** (*nparray of int [N_tri x 3]*) – Connectivity of points forming triangles
- **tetrahedras** (*nparray of int [N_tri x 4]*) – Connectivity of points forming tetrahedra idx_start: int smallest index in connectivity matrix, defines offset w.r.t python indexing, which starts at '0'
- ***data** (*nparray(s) [N_tet x N_comp(N_data)]*) – Arrays containing data in tetrahedra center multiple components per dataset possible e.g. [Ex, Ey, Ez]

Returns

<File> – Geometry and data information

Return type

.vtu file

1.1.9 pynibs.roi module

class pynibs.roi.**CorticalLayer**(*create_key*, *id*, *volumetric_mesh=None*, *roi=None*, *depth=None*, *path=None*, *surface=None*)

Bases: object

class Settings

Bases: object

GRID_POINTS_PER_MM = 1.5

NUM_TRIANGLE_SMOOTHING_STEPS = 30

ROI_SIZE_OFFSET = 5

TAG_GRAY_MATTER_SURF = 1002

TAG_GRAY_MATTER_VOL = 2

TAG_WHITE_MATTER_SURF = 1001

TAG_WHITE_MATTER_VOL = 1

classmethod create_in_bbox(*id*, *bbox*, *depth*, *volmesh*)

Factory method for constructing a CorticalLayer-object within a region-of-interest and a specified cortical depth.

Parameters

- **id** (*str*) – Identifier of the layer.
- **bbox** (*List[float]*) – List of bounding values around the ROI box: [x_min, x_max, y_min, y_max, z_min, z_max].
- **depth** (*float*) – Normalized distance of the layer from gray matter surface. Provide values in the open interval (0,1)
- **volmesh** (*simnibs.Msh*) – The tetrahedral volume mesh, in which the layer should be generated.

classmethod create_in_roi(*id*, *roi*, *depth*, *volmesh*)

Factory method for constructing a CorticalLayer-object within a region-of-interest and a specified cortical depth.

Parameters

- **id** (*str*) – Identifier of the layer.
- **roi** (*RegionOfInterestSurface instance*) – RegionOfInterestSurface
- **depth** (*float*) – Normalized distance of the layer from gray matter surface. Provide values in the open interval (0,1)
- **volmesh** (*simnibs.Msh*) – The tetrahedral volume mesh, in which the layer should be generated.

static crop_mesh_with_box(*mesh*, *roi*, *keep_elements=False*)

Returns the cropped mesh with all points that are inside the region of interest

Parameters

- **keep_elements** (*bool, default = False*) – If True, keeps elements with at least one point in roi, else removes them.
- **mesh** (*simnibs.Msh*) – The mesh that is supposed to be cropped.
- **roi** (*List[float]*) – The bounding box of the region of interest which the mesh should be cropped to. [x-min, x-max, y-min, y-max, z-min, z-max]

Returns
mesh_cropped – The cropped mesh.

Return type
simnibs.Msh

`static crop_mesh_with_surface`(*mesh*, *roi*, *keep_elements=False*, *radius=3*)

Returns the cropped mesh with all points that are close to the surface of interest

Parameters

- **keep_elements** (*bool, default = False*) – If True, keeps elements with at least one point in roi, else removes them.
- **mesh** (*simnibs.Msh*) – The mesh that is supposed to be cropped.
- **roi** (*RegionOfInterestSurface instance*) – RegionOfInterestSurface
- **radius** (*float, default = 3*) – Search radius of mesh elements around ROI nodes

Returns
mesh_cropped – The cropped mesh.

Return type
simnibs.Msh

`generate_layer`(*depth*, *roi*)

Create the geometry of the layer at the specified depth using marching cubes.

Parameters

- **depth** (*float*) – The depth below the GM surface at which the layer should be generated; in [0,1].
- **roi** (*RegionOfInterestSurface instance*) – RegionOfInterestSurface

`get_evenly_spaced_element_subset`(*elements_per_square_mm*)

Subsample the surface representation of the ayer.

Parameters
elements_per_square_mm (*float*) – Number of triangles per mm^2 in the layer.

Returns
selected elements – List of indices of selected elements as a result of the subsampling.

Return type
Typing.List[int]

`get_smoothed_normals`()

Computed the smoothed normals of the surface representation of this layer. Note: For the later stages, we don't want a smoothed surface, but smooth

normals in order to maintain the location of the cells, but orient them more smoothly. Therefore, we use smoothed normals, e.g. for the computation of the theta angle, but do not smooth the entire layer surface.

Returns
normals – The tetrahedral volume mesh, in which the layer should be generated.

Return type
np.ndarray

`classmethod init_from_file`(*id*, *fn*)

Factory method for constructing a CorticalLayer-object from a file.

Parameters

- **id** (*str*) – Identifier of the layer.

- **fn** (*str*) – File path to a region of interest surfe (e.g. midlayer).

classmethod init_from_surface(*id*, *surf*)

Factory method for constructing a CorticalLayer-object from a Simnibs-surface object.

Parameters

- **id** (*str*) – Identifier of the layer.
- **surf** (*simnibs.Msh*) – The surface representation of an already existing layer (e.g. midlayer).

remove_unconnected_surfaces()

Remove elements small unconnected element-clusters from this layer.

static roi_bbox_from_points(*points*, *offset=0*)

Find the minimal bounding box around the provided points.

Parameters

- **points** (*simnibs.Msh*) – The tetrahedral volume mesh, in which the layer should be generated.
- **offset** (*float*) – Normalized distance of the layer from gray matter surface. Provide values in the open interval (0,1)

Returns

bounding_box – List of bounding values around the provided points: [x_min, x_max, y_min, y_max, z_min, z_max].

Return type

List[float]

save(*fn*)

Save the current surface representation of this CorticalLayer instance at the specified location.

Parameters

fn (*str*) – Target file name of the surface-file of this layer.

class pynibs.roi.**RegionOfInterestSurface**

Bases: object

Region of interest (surface).

node_coord_up

(N_points, 3) Coordinates (x,y,z) of upper surface nodes

Type

np.ndarray

node_coord_mid

(N_points, 3) Coordinates (x,y,z) of middle surface nodes

Type

np.ndarray

node_coord_low

(N_points, 3) Coordinates (x,y,z) of lower surface nodes

Type

np.ndarray

node_number_list

(N_points, 3) Connectivity matrix of triangles

Type

np.ndarray

delta

Distance parameter between WM and GM (0 -> WM, 1 -> GM)

Type
float

tet_idx_tri_center_up

Tetrahedra indices of TetrahedraLinear object instance where the center points of the triangles of the upper surface are lying in

Type
np.ndarray [N_points]

tet_idx_tri_center_mid

Tetrahedra indices of TetrahedraLinear object instance where the center points of the triangles of the middle surface are lying in

Type
np.ndarray [N_points]

tet_idx_tri_center_low

Tetrahedra indices of TetrahedraLinear object instance where the center points of the triangles of the lower surface are lying in

Type
np.ndarray [N_points]

tet_idx_node_coord_mid

(N_tri,) Tetrahedra indices of TetrahedraLinear object instance where the nodes of the middle surface are lying in

Type
np.ndarray

tri_center_coord_up

(N_tri, 3) Coordinates of roi triangle center of upper surface

Type
np.ndarray

tri_center_coord_mid

(N_tri, 3) Coordinates of roi triangle center of middle surface

Type
np.ndarray

tri_center_coord_low

(N_tri, 3) Coordinates of roi triangle center of lower surface

Type
np.ndarray

fn_mask

Filename for surface mask in subject space. .mgh file or freesurfer surface file.

Type
string

fn_mask_avg

Filename for .mgh mask in fsaverage space. Absolute path or relative to mesh folder.

Type
string

fn_mask_nii

Filename for .nii or .nii.gz mask. Absolute path or relative to mesh folder.

Type
string

X_ROI

Region of interest [Xmin, Xmax], whole X range if empty [0,0] or None (left - right)

Type
list of float

Y_ROI

Region of interest [Ymin, Ymax], whole Y range if empty [0,0] or None (anterior - posterior)

Type
list of float

Z_ROI

Region of interest [Zmin, Zmax], whole Z range if empty [0,0] or None (inferior - superior)

Type
list of float

template

'MNI', 'fsaverage', 'subject'

Type
str

center

Center coordinates for spherical ROI in self.template space

Type
list of float

radius

Radius in [mm] for spherical ROI

Type
float

gm_surf_fname

Filename(s) of GM surface generated by freesurfer (lh and/or rh) (e.g. in mri2msh: .../fs_ID/surf/lh.pial)

Type
str or list of str

wm_surf_fname

Filename(s) of WM surface generated by freesurfer (lh and/or rh) (e.g. in mri2msh: .../fs_ID/surf/lh.white)

Type
str or list of str

layer

Define the number of layers:

- 1: one layer
- 3: additionally upper and lower layers are generated around the central midlayer

Type
int

determine_element_idx_in_mesh(*msh*)

Determines tetrahedra indices of msh where the triangle center points of upper, middle and lower surface and the nodes of middle surface are lying in.

Parameters

msh (*pynibs.TetrahedraLinear*) – Object instance of TetrahedraLinear class

Returns

- **RegionOfInterestSurface.tet_idx_tri_center_up** (*np.ndarray [N_points]*) – Tetrahedra indices of TetrahedraLinear object instance where the center points of the triangles of the upper surface are lying in
- **RegionOfInterestSurface.tet_idx_tri_center_mid** (*np.ndarray [N_points]*) – Tetrahedra indices of TetrahedraLinear object instance where the center points of the triangles of the middle surface are lying in
- **RegionOfInterestSurface.tet_idx_tri_center_low** (*np.ndarray [N_points]*) – Tetrahedra indices of TetrahedraLinear object instance where the center points of the triangles of the lower surface are lying in
- **RegionOfInterestSurface.tet_idx_node_coord_mid** (*np.ndarray [N_tri]*) – Tetrahedra indices of TetrahedraLinear object instance where the nodes of the middle surface are lying in

generate_cortical_laminae(*head_model_mesh*, *bbox=None*, *laminae=(0.06, 0.4, 0.55, 0.65, 0.85)*, *layer_ids=('L1', 'L23', 'L4', 'L5', 'L6')*)

Create the cortical layering with the provided laminar depths. Defaults to the standard depths of the laminae in the neo-cortex from layer I to VI

from "Simulation of transcranial magnetic stimulation in head model with morphologically-realistic cortical neurons", Aberra et al., https://doi.org/10.1016/j.brs.2019.10.002

Parameters

- **head_model_mesh** (*simnibs.Msh*) – The head model volume mesh. Inside the GM compartment of this mesh, the layering will be generated.
- **bbox** (*np.ndarray (optional)*) – Bounding coordinates of the region of interest. Optional, if the mid-layer surface is already existing (and can thus be used to determine the bounding coordinates).
- **laminae** (*Dict[str,float] | Tuple*) – List of depths of the individual to-be created lamiae.

make_GM_WM_surface(*gm_surf_fname=None*, *wm_surf_fname=None*, *midlayer_surf_fname=None*, *mesh_folder=None*, *delta=0.5*, *x_roi=None*, *y_roi=None*, *z_roi=None*, *layer=1*, *fn_mask=None*, *refine=False*)

Generating a surface between WM and GM in a distance of delta 0…1 for ROI, given by Freesurfer mask or coordinates.

Parameters

- **gm_surf_fname** (*str or list of str*) – Filename(s) of GM FreeSurfer surface(s) (lh and/or rh). Either relative to mesh_folder (fs_ID/surf/lh.pial) or absolute (/full/path/to/lh.pial)
- **wm_surf_fname** (*str or list of str*) – Filename(s) of WM FreeSurfer surface(s) (lh and/or rh) Either relative to mesh_folder (fs_ID/surf/lh.white) or absolute (/full/path/to/lh.white)
- **midlayer_surf_fname** (*str or list of str*) – Filename(s) of midlayer surface (lh and/or rh) Either relative to mesh_folder (fs_ID/surf/lh.central) or absolute (/full/path/to/lh.central)

- **mesh_folder** (*str*) – Root folder of mesh, Needed if paths above are given relative, or refine=True
- **m2m_mat_fname** (*[defunct]*) – Filename of mri2msh transformation matrix (e.g. in mri2msh: .../m2m_ProbandID/MNI2conform_6DOF.mat)
- **delta** (*float*) – Distance parameter where surface is generated 0...1 (default: 0.5)
  - 0 -> WM surface
  - 1 -> GM surface
- **x_roi** (*list of float*) – Region of interest [Xmin, Xmax], whole X range if empty [0,0] or None (left - right)
- **y_roi** (*list of float*) – Region of interest [Ymin, Ymax], whole Y range if empty [0,0] or None (anterior - posterior)
- **z_roi** (*list of float*) – Region of interest [Zmin, Zmax], whole Z range if empty [0,0] or None (inferior - superior)
- **layer** (*int*) – Define the number of layers:
  - 1: one layer
  - 3: additionally upper and lower layers are generated around the central midlayer
- **fn_mask** (*str*) – Filename for FreeSurfer .mgh mask.
- **refine** (*bool, optional, default: False*) – Refine ROI by splitting elements

Returns

- **node_coord_up** (*np.ndarray of float [N_roi_points x 3]*) – Node coordinates (x, y, z) of upper epsilon layer of ROI surface
- **node_coord_mid** (*np.ndarray of float [N_roi_points x 3]*) – Node coordinates (x, y, z) of ROI surface
- **node_coord_low** (*np.ndarray of float [N_roi_points x 3]*) – Node coordinates (x, y, z) of lower epsilon layer of ROI surface
- **node_number_list** (*np.ndarray of int [N_roi_tri x 3]*) – Connectivity matrix of intermediate surface layer triangles
- **delta** (*float*) – Distance parameter where surface is generated 0...1 (default: 0.5)
  - 0 -> WM surface
  - 1 -> GM surface
- **tri_center_coord_up** (*np.ndarray of float [N_roi_tri x 3]*) – Coordinates (x, y, z) of triangle center of upper epsilon layer of ROI surface
- **tri_center_coord_mid** (*np.ndarray of float [N_roi_tri x 3]*) – Coordinates (x, y, z) of triangle center of ROI surface
- **tri_center_coord_low** (*np.ndarray of float [N_roi_tri x 3]*) – Coordinates (x, y, z) of triangle center of lower epsilon layer of ROI surface
- **fn_mask** (*str*) – Filename for freesurfer mask. If given, this is used instead of *_ROIs
- **X_ROI** (*list of float*) – Region of interest [Xmin, Xmax], whole X range if empty [0,0] or None (left - right)
- **Y_ROI** (*list of float*) – Region of interest [Ymin, Ymax], whole Y range if empty [0,0] or None (anterior - posterior)
- **Z_ROI** (*list of float*) – Region of interest [Zmin, Zmax], whole Z range if empty [0,0] or None (inferior - superior)

Example

```
make_GM_WM_surface(self, gm_surf_fname, wm_surf_fname, delta, X_ROI, Y_ROI, Z_ROI)
make_GM_WM_surface(self, gm_surf_fname, wm_surf_fname, delta, mask_fn, layer=3)
```

project_on_midlayer(*target*, *verbose=False*)

Project a coordinate on the nearest midlayer node

Parameters

- **target** (*np.ndarray*) – Coordinate to project as (3,) array
- **verbose** (*bool*) – Print some verbosity information. Default: False

Returns

target_proj – Node coordinate of nearest midlayer node.

Return type

np.ndarray

subsample(*dist=10*, *fn_sphere=''*)

Subsample ROI surface and return element indices

Parameters

- **dist** (*float*) – Distance in mm the subsampled points lie apart
- **fn_sphere** (*str*) – Name of ?.sphere file (freesurfer)

Returns

ele_idx – Element indices of the subsampled surface

Return type

ndarray of float [n_ele]

class pynibs.roi.**RegionOfInterestVolume**

Bases: object

Region of interest (volume) class

node_coord

Coordinates (x,y,z) of ROI tetrahedra nodes

Type

np.ndarray [N_points x 3]

tet_node_number_list

Connectivity matrix of ROI tetrahedra

Type

np.ndarray [N_tet_roi x 3]

tri_node_number_list

Connectivity matrix of ROI tetrahedra

Type

np.ndarray [N_tri_roi x 3]

tet_idx_node_coord

Tetrahedra indices of TetrahedraLinear object instance where the ROI nodes are lying in

Type

np.ndarray [N_points]

tet_idx_tetrahedra_center

Tetrahedra indices of TetrahedraLinear object instance where the center points of the ROI tetrahedra are lying in

Type
np.ndarray [N_tet_roi]

`tet_idx_triangle_center`

Tetrahedra indices of TetrahedraLinear object instance where the center points of the ROI triangle are lying in. If the ROI is directly generated from the msh instance using "make_roi_volume_from_msh", these indices are the triangle indices of the head mesh since the ROI mesh and the head mesh are overlapping. If the ROI mesh is not the same as the head mesh, the triangle center of the ROI mesh are always lying in a tetrahedra of the head mesh (these indices are given in this case)

Type
np.ndarray [N_tri_roi]

`make_roi_volume_from_msh`(*msh*, *volume_type='box'*, *x_roi=None*, *y_roi=None*, *z_roi=None*)

Generate region of interest (volume) and extract nodes, triangles and tetrahedra from msh instance.

Parameters

- **msh** (*pynibs.TetrahedraLinear*) – Mesh object instance of type TetrahedraLinear (see main.py)
- **volume_type** (*str*) – Type of ROI ('box' or 'sphere')
- **x_roi** (*list of float*) –
  - type = 'box': [Xmin, Xmax] (in mm), whole X range if empty [0,0] or None (left - right)
  - type = 'sphere': origin [x,y,z]
- **y_roi** (*list of float*) –
  - type = 'box': [Ymin, Ymax] (in mm), whole Y range if empty [0,0] or None (anterior - posterior)
  - type = 'sphere': radius (in mm)
- **z_roi** (*list of float*) –
  - type = 'box': [Zmin, Zmax] (in mm), whole Z range if empty [0,0] or None (inferior - superior)
  - type = 'sphere': None

Returns

- **RegionOfInterestVolume.node_coord** (*np.ndarray [N_points x 3]*) – Coordinates (x,y,z) of ROI tetrahedra nodes
- **RegionOfInterestVolume.tet_node_number_list** (*np.ndarray [N_tet_roi x 3]*) – Connectivity matrix of ROI tetrahedra
- **RegionOfInterestVolume.tri_node_number_list** (*np.ndarray [N_tri_roi x 3]*) – Connectivity matrix of ROI tetrahedra
- **RegionOfInterestVolume.tet_idx_node_coord** (*np.ndarray [N_points]*) – Tetrahedra indices of TetrahedraLinear object instance where the ROI nodes are lying in
- **RegionOfInterestVolume.tet_idx_tetrahedra_center** (*np.ndarray [N_tet_roi]*) – Tetrahedra indices of TetrahedraLinear object instance where the center points of the ROI tetrahedra are lying in
- **RegionOfInterestVolume.tet_idx_triangle_center** (*np.ndarray [N_tri_roi]*) – Tetrahedra indices of TetrahedraLinear object instance where the center points of the ROI triangle are lying in. If the ROI is directly generated from the msh instance using "make_roi_volume_from_msh", these indices are the triangle indices of the head mesh since the ROI mesh and the head mesh are overlapping. If the ROI mesh is not the same as the head mesh, the triangle center of the ROI mesh are always lying in a tetrahedra of the head mesh (these indices are given in this case)

pynibs.roi.**clean_roi**(*img*, *vox_thres=0.5*, *fn_out=None*)

Remove values < vox thres from image.

Parameters

- **img** (*str or nibabel.nifti1.Nifti1Image*) –
- **vox_thres** (*float, optional*) –
- **fn_out** (*str*) –

Returns

- **img_thres** (*nibabel.nifti1.Nifti1Image*)
- **img_thres** (*<file>*) – If fn_out is specified, thresholded image is saved here

pynibs.roi.**create_refine_spherical_roi**(*center*, *radius*, *final_tissues_nii*, *out_fn*, *target_size=0.5*, *outside_size=None*, *outside_factor=3*, *out_spher_fn=None*, *tissue_types=None*, *verbose=False*)

Create a spherical roi nifti for simnibs 4 refinement. Only tissue types accoring to _tissue_types will be refined.

Use the resulting output file as input for –sizing_field in SimNIBS-4/simnibs/cli/meshmesh.py

Parameters

- **center** (*list of float*) – Center of spherical ROI in mm
- **radius** (*float*) – Radius of spherical ROI in mm
- **final_tissues_nii** (*string or nib.nifti1.Nifti1Image*) – final_tissues.nii.gz to create roi for.
- **out_fn** (*str*) – Final output filename
- **target_size** (*float, default = 0.5*) – Target element size of refined areas in mm (?)
- **outside_size** (*float, default = None*) – Element size outside of target size.
- **outside_factor** (*float, default = None*) – Distance factor to define the 'outside' area: oudsidefactor * radius -> outside
- **out_spher_fn** (*str, optional*) – Output filename of orignal, raw spherical ROI
- **tissue_types** (*list of float, default = [1,2,3]*) – Which tissue types to refine. Defaults to WM, GM, CSF
- **verbose** (*bool, optional, default=False*) – Print additional information

pynibs.roi.**determine_element_idx_in_mesh**(*fname*, *msh*, *points*, *compute_baricentric=False*)

Finds the tetrahedron that contains each of the described points using a stochastic walk algorithm. Implemented from Devillers et al. (2002) [1]

Parameters

- **msh** (pynibs.mesh.mesh_struct.TetrahedraLinear) –
- **fname** (*str or None*) – Filename of saved .txt file containing the element indices (no data is saved when *fname=None* or *fname=''*)
- **points** (*np.ndarray (N, 3) or list of np.ndarray*) – List of points to be queried
- **compute_baricentric** (*bool*) – Wether or not to compute baricentric coordinates of the points

Returns

- **th_with_points** (*np.ndarray*) – List with the tetrahedron that contains each point. If the point is outside the mesh, the value will be -1
- **baricentric** (*np.ndarray [n, 4](if compute_baricentric == True)*) – Baricentric coordinates of point. If the point is outside, a list of zeros

Notes

pynibs.roi.**elem_workhorse**(*chunk*, *points_out*, *P1_all*, *P2_all*, *P3_all*, *P4_all*, *N_points_total*, *N_CPU*)

Parameters

- **chunk** (*np.ndarray*) – Indices of points the CPU thread is computing the element indices for
- **points_out** (*np.ndarray of float*) – (N_points, 3) Coordinates of points, the tetrahedra indices are computed for
- **P1_all** (*np.ndarray of float*) –
- **(N_tet** –
- **tetrahedra** (*3) Coordinates of first point of*) –
- **P2_all** (*np.ndarray of float*) – (N_tet, 3) Coordinates of second point of tetrahedra
- **P3_all** (*np.ndarray of float*) – (N_tet, 3) Coordinates of third point of tetrahedra
- **P4_all** (*np.ndarray of float*) – (N_tet, 3) Coordinates of fourth point of tetrahedra
- **N_points_total** (*int*) – Total number of points
- **N_CPU** (*int*) – Number of CPU cores to use

Returns
tet_idx_local

Return type
np.ndarray of int (N_points,)

pynibs.roi.**get_mask**(*areas*, *fn_annot*, *fn_inflated_fs*, *fn_out*)

Determine freesurfer average mask .overlay file, which is needed to generate subject specific ROIs.

Parameters

- **areas** (*list of str*) – Brodmann areas (e.g. ['Brodmann.6', 'Brodmann.4', 'Brodmann.3', 'Brodmann.1'])
- **fn_annot** (*str*) – Annotation file of freesurfer (e.g. 'FREESURFER_DIR/fsaverage/label/lh.PALS_B12_Brodmann.annot')
- **fn_inflated_fs** (*str*) – Inflated surface of freesurfer average (e.g. 'FREESURFER_DIR/fsaverage/surf/lh.inflated')
- **fn_out** (*str*) – Filename of .overlay file of freesurfer mask

Returns
<File> – fn_out.overlay file of freesurfer mask

Return type
.overlay file

pynibs.roi.**get_sphere_in_nii**(*center*, *radius*, *nii=None*, *out_fn=None*, *thresh_by_nii=True*, *val_in=1*, *val_out=0*, *outside_val=0*, *outside_radius=inf*)

Computes a spherical ROI for a given Nifti image (defaults to SimNIBS MNI T1 tissue). The ROI area is defined in nifti coordinates. By default, everything inside the ROI is set to 1, areas outside = 0. The ROI is further thresholded by the nifti. A nib.Nifti image is returned and optionally saved.

Parameters

- **center** (*array-like*) – X, Y, Z coordinates in nifti space
- **radius** (*float*) – radius of sphere
- **nii** (*string or nib.nifti1.Nifti1Image, optional*) – The nifti image to work with.
- **out_fn** (*string, optional*) – If provided, sphere ROI image is saved here
- **outside_val** (*float, default = None*) – Value outside of outside_radius.
- **outside_radius** (*float, default = None*) – Distance factor to define the 'outside' area: oudsidefactor * radius -> outside
- **thresh_by_nii** (*bool, optional*) – Mask sphere by nii != 0
- **val_in** (*float, optional*) – Value within ROI
- **val_out** (*float, optional*) – Value outside ROI

Returns

- **sphere_img** (*nib.nifti1.Nifti1Image*)
- **sphere_img** (*<file>, optional*)

Raises

ValueError – If the final ROI is empty.

pynibs.roi.**load_roi_surface_obj_from_hdf5**(*fname*)

Loading and initializing RegionOfInterestSurface object/s from .hdf5 mesh file.

Parameters

fname (*str*) – Filename (incl. path) of .hdf5 mesh file, e.g. from subject.fn_mesh_hdf5

Returns

RegionOfInterestSurface – RegionOfInterestSurface

Return type

pynibs.roi.RegionOfInterestSurface or list of *pynibs.roi.RegionOfInterestSurface*

pynibs.roi.**make_GM_WM_surface**(*gm_surf_fname*, *wm_surf_fname*, *mesh_folder*, *midlayer_surf_fname=None*, *delta=0.5*, *x_roi=None*, *y_roi=None*, *z_roi=None*, *layer=1*, *fn_mask=None*, *refine=False*)

Generating a surface between WM and GM in a distance of delta 0...1 for ROI, given by freesurfer mask or coordinates.

Parameters

- **gm_surf_fname** (*str or list of str*) – Filename(s) of GM surface generated by freesurfer (lh and/or rh) (e.g. in mri2msh: fs_ID/surf/lh.pial)
- **wm_surf_fname** (*str or list of str*) – Filename(s) of WM surface generated by freesurfer (lh and/or rh) (e.g. in mri2msh: fs_ID/surf/lh.white)
- **mesh_folder** (*str*) – Path of mesh (parent directory)
- **midlayer_surf_fname** (*str or list of str*) – filename(s) of midlayer surface generated by headreco (lh and/or rh) (e.g. in headreco: fs_ID/surf/lh.central) (after conversion)
- **m2m_mat_fname** (*[defunct]*) – Filename of mri2msh transformation matrix (e.g. in mri2msh: m2m_ProbandID/MNI2conform_6DOF.mat)
- **delta** (*float*) – Distance parameter where surface is generated 0...1 (default: 0.5)
  - 0 -> WM surface
  - 1 -> GM surface

- **x_roi** (*list of float or None*) – Region of interest [Xmin, Xmax], whole X range if empty [0,0] or None (left - right)
- **y_roi** (*list of float or None*) – Region of interest [Ymin, Ymax], whole Y range if empty [0,0] or None (anterior - posterior)
- **z_roi** (*list of float or None*) – Region of interest [Zmin, Zmax], whole Z range if empty [0,0] or None (inferior - superior)
- **layer** (*int*) – Define the number of layers:
  - 1: one layer
  - 3: additionally upper and lower layers are generated around the central midlayer
- **fn_mask** (*string or None*) – Filename for freesurfer mask. If given, this is used instead of *_ROIs
- **refine** (*bool, optional, default: False*) – Refine ROI by splitting elements

Returns

- *if layer == 3*
- **surface_points_upper** (*np.ndarray of float*) – (N_points, 3) Coordinates (x, y, z) of surface + epsilon (in GM surface direction)
- **surface_points_middle** (*np.ndarray of float*) – (N_points, 3) Coordinates (x, y, z) of surface
- **surface_points_lower** (*np.ndarray of float*) – (N_points, 3) Coordinates (x, y, z) of surface - epsilon (in WM surface direction)
- **connectivity** (*np.ndarray of int*) – (N_tri x 3) Connectivity of triangles (indexation starts at 0!)
- *else*
- **surface_points_middle** (*np.ndarray of float*) – (N_points, 3) Coordinates (x, y, z) of surface
- **connectivity** (*np.ndarray of int*) – (N_tri x 3) Connectivity of triangles (indexation starts at 0!)

Example

```
make_GM_WM_surface(self, gm_surf_fname, wm_surf_fname, delta, X_ROI, Y_ROI, Z_ROI)
make_GM_WM_surface(self, gm_surf_fname, wm_surf_fname, delta, mask_fn, layer=3)
```

pynibs.roi.**nii2msh**(*mesh*, *m2m_dir*, *nii*, *out_folder*, *hem*, *out_fsaverage=False*, *roi_name='ROI'*)

Transform a nifti ROI image to subject space .mgh file.

Parameters

- **mesh** (*simnibs.Mesh or str*) –
- **m2m_dir** (*str*) –
- **nii** (*nibabel.nifti1.Nifti1Image or str*) –
- **out_folder** (*str*) –
- **hem** (*str*) – 'lh' or 'rh'
- **out_fsaverage** (*bool*) –
- **roi_name** (*str*) – How to name the ROI

Returns

roi – f"{out_folder}/{hem}.mesh.central.{roi_name}"

Return type
file

1.1.10 pynibs.subject module

class pynibs.subject.**Subject**(*subject_id*, *subject_folder*)

Bases: object

Subject containing subject specific information, like mesh, roi, uncertainties, plot settings.

self.id

Subject id from MPI database

Type
str

Notes

Initialization

sub = pynibs.subject(subject_ID, mesh)

Parameters

id
[str] Subject id

fn_mesh
[str] .msh or .hdf5 file containing the mesh information

Subject.seg, segmentation information dictionary

fn_lh_wm
[str] Filename of left hemisphere white matter surface

fn_rh_wm
[str] Filename of right hemisphere white matter surface

fn_lh_gm
[str] Filename of left hemisphere grey matter surface

fn_rh_gm
[str] Filename of right hemisphere grey matter surface

fn_lh_curv
[str] Filename of left hemisphere curvature data on grey matter surface

fn_rh_curv
[str] Filename of right hemisphere curvature data on grey matter surface

Subject.mri, mri information dictionary

fn_mri_T1
[str] Filename of T1 image

fn_mri_T2
[str] Filename of T2 image

fn_mri_DTI
[str] Filename of DTI dataset

fn_mri_DTI_bvec
[str] Filename of DTI bvec file

fn_mri_DTI_bval
[str] Filename of DTI bval file

fn_mri_conform
[str] Filename of conform T1 image resulting from SimNIBS mri2mesh function

Subject.ps, plot settings dictionary

see plot functions in para.py for more details

Subject.exp, experiment dictionary

info
[str] General information about the experiment

date
[str] Date of experiment (e.g. 01/01/2018)

fn_tms_nav
[str] Path to TMS navigator folder

fn_data
[str] Path to data folder or files

fn_exp_csv
[str] Filename of experimental data .csv file containing the merged experimental data information

fn_coil
[str] Filename of .ccd or .nii file of coil used in the experiment (contains ID)

fn_mri_nii
[str] Filename of MRI .nii file used during the experiment

cond
[str or list of str] Conditions in the experiment in the recorded order (e.g. ['PA-45', 'PP-00'])

experimenter
[str] Name of experimenter who conducted the experiment

incidents
[str] Description of special events occured during the experiment

Subject.mesh, mesh dictionary

info
[str] Information about the mesh (e.g. dicretization etc)

fn_mesh_msh
[str] Filename of the .msh file containing the FEM mesh

fn_mesh_hdf5
[str] Filename of the .hdf5 file containing the FEM mesh

seg_idx
[int] Index indicating to which segmentation dictionary the mesh belongs

Subject.roi region of interest dictionary

type
[str] Specify type of ROI ('surface', 'volume')

info
[str] Info about the region of interest, e.g. "M1 midlayer from freesurfer mask xyz"

region
[list of str or float] Filename for freesurfer mask or [[X_min, X_max], [Y_min, Y_max], [Z_min, Z_max]]

delta
[float] Distance parameter between WM and GM (0 -> WM, 1 -> GM) (for surfaces only)

add_experiment_info(*exp_dict*)

Adding information about a particular experiment.

Parameters
exp_dict (*dict of dict or list of dict*) – Dictionary containing information about the experiment

Notes

Adds Attributes

exp
[list of dict] Dictionary containing information about the experiment

add_mesh_info(*mesh_dict*)

Adding filename information of the mesh to the subject object (multiple filenames possible).

Parameters
mesh_dict (*dictionary or list of dictionaries*) – Dictionary containing the mesh information

Notes

Adds Attributes

Subject.mesh
[list of dict] Dictionaries containing the mesh information

add_mri_info(*mri_dict*)

Adding MRI information to the subject object (multiple MRIs possible).

Parameters
mri_dict (*dictionary or list of dictionaries*) – Dictionary containing the MRI information of the subject

Notes

Adds Attributes

Subject.mri
[list of dict] Dictionary containing the MRI information of the subject

add_plotsettings(*ps_dict*)

Adding ROI information to the subject object (multiple ROIs possible).

Parameters
ps_dict (*dictionary or list of dictionaries*) – Dictionary containing plot settings of the subject

Notes

Adds Attributes

Subject.ps
[list of dict] Dictionary containing plot settings of the subject

add_roi_info(*roi_dict*)

Adding ROI (surface) information of the mesh with mesh_index to the subject object (multiple ROIs possible).

Parameters
roi_dict (*dict of dict or list of dictionaries*) – Dictionary containing the ROI information of the mesh with mesh_index [mesh_idx][roi_idx]

Notes

Adds Attributes

Subject.mesh[mesh_index].roi
[list of dict] Dictionaries containing ROI information

pynibs.subject.**check_file_and_format**(*fname*)

Checking existence of file and transforming to list if necessary.

Parameters
fname (*str or list of str*) – Filename(s) to check

Returns
fname – Checked filename(s) as list

Return type
list of str

pynibs.subject.**fill_from_dict**(*obj*, *d*)

Set all attributes from d in obj.

Parameters

- **obj** (*pynibs.Mesh or pynibs.ROI*) –
- **d** (*dict*) –

Returns
obj

Return type
pynibs.Mesh or pynibs.ROI

pynibs.subject.**load_subject**(*fname*, *filetype=None*)

Wrapper for pkl and hdf5 subject loader

Parameters

- **fname** (*str*) – endwith('.pkl') | endswith('.hdf5')
- **filetype** (*str*) – Explicitely set file version.

Returns
subject

Return type
pynibs.subject.Subject

pynibs.subject.**load_subject_hdf5**(*fname*)

Loading subject information from .hdf5 file and returning subject object.

Parameters
fname (*str*) – Filename with .hdf5 extension (incl. path)

Returns
subject – Loaded Subject object

Return type
pynibs.subject.Subject

pynibs.subject.**load_subject_pkl**(*fname*)

Loading subject object from .pkl file.

Parameters
fname (*str*) – Filename with .pkl extension

Returns
subject – Loaded Subject object

Return type
pynibs.subject.Subject

pynibs.subject.**save_subject**(*subject_id*, *subject_folder*, *fname*, *mri_dict=None*, *mesh_dict=None*, *roi_dict=None*, *exp_dict=None*, *ps_dict=None*, ***kwargs*)

Saves subject information in .pkl or .hdf5 format (preferred)

Parameters

- **subject_id** (*str*) – ID of subject
- **subject_folder** (*str*) – Subject folder
- **fname** (*str*) – Filename with .hdf5 or .pkl extension (incl. path)
- **mri_dict** (*list of dict, optional, default: None*) – MRI info
- **mesh_dict** (*list of dict, optional, default: None*) – Mesh info
- **roi_dict** (*list of list of dict, optional, default: None*) – Mesh info
- **exp_dict** (*list of dict, optional, default: None*) – Experiment info
- **ps_dict** (*list of dict, optional, default:None*) – Plot-settings info
- **kwargs** (*str or np.array*) – Additional information saved in the parent folder of the .hdf5 file

Returns
<File> – Subject information

Return type
.hdf5 file

pynibs.subject.**save_subject_hdf5**(*subject_id*, *subject_folder*, *fname*, *mri_dict=None*, *mesh_dict=None*, *roi_dict=None*, *exp_dict=None*, *ps_dict=None*, *overwrite=True*, *check_file_exist=False*, *verbose=False*, ***kwargs*)

Saving subject information in hdf5 file.

Parameters

- **subject_id** (*str*) – ID of subject
- **subject_folder** (*str*) – Subject folder
- **fname** (*str*) – Filename with .hdf5 extension (incl. path)
- **mri_dict** (*list of dict, optional, default: None*) – MRI info
- **mesh_dict** (*list of dict, optional, default: None*) – Mesh info
- **roi_dict** (*list of list of dict, optional, default: None*) – Mesh info
- **exp_dict** (*list of dict or dict of dict, optional, default: None*) – Experiment info
- **ps_dict** (*list of dict, optional, default:None*) – Plot-settings info
- **overwrite** (*bool*) – Overwrites existing .hdf5 file
- **check_file_exist** (*bool*) – Hide warnings.
- **verbose** (*bool*) – Print information about meshes and ROIs.

- **kwargs** (*str or np.ndarray*) – Additional information saved in the parent folder of the .hdf5 file

Returns
<File> – Subject information

Return type
.hdf5 file

pynibs.subject.**save_subject_pkl**(*sobj*, *fname*)

Saving subject object as pickle file.

Parameters

- **sobj** (*object*) – Subject object to save
- **fname** (*str*) – Filename with .pkl extension

Returns
<File> – Subject object instance

Return type
.pkl file

1.1.11 pynibs.tensor_scaling module

pynibs.tensor_scaling.**ellipse_eccentricity**(*a*, *b*)

Calculates the eccentricity of an 2D ellipse with the semi axis a and b. An eccentricity of 0 corresponds to a sphere and an eccentricity of 1 means complete eccentric (line) with full restriction to the other axis

Parameters

- **a** (*float*) – First semi axis parameter
- **b** (*float*) – Second semi axis parameter

Returns
e – Eccentricity (0...1)

Return type
float

pynibs.tensor_scaling.**rescale_lambda_centerized**(*D*, *s*, *tsc=False*)

Rescales the eigenvalues of the matrix D according to their eccentricity. The scale factor is between 0...1 a scale factor of 0.5 would not alter the eigenvalues of the matrix D. A scale factor of 0 would unify all eigenvalues to one value such that it corresponds to a isotropic sphere. A scale factor of 1 alters the eigenvalues in such a way that the resulting ellipsoid is fully eccentric and anisotropic.

Parameters

- **D** (*nparray of float [3 x 3]*) – Diffusion tensor
- **s** (*float*) – Scale parameter [0 (iso) ... 0.5 (unaltered)... 1 (aniso)]
- **tsc** (*boolean*) – Tensor singularity correction

Returns
Ds – Scaled diffusion tensor

Return type
nparray of float [3 x 3]

pynibs.tensor_scaling.**rescale_lambda_centerized_workhorse**(*D*, *s*, *tsc=False*)

Rescales the eigenvalues of the matrix D according to their eccentricity. The scale factor is between 0...1 a scale factor of 0.5 would not alter the eigenvalues of the matrix D. A scale factor of 0 would unify all eigenvalues to one value such that it corresponds to a isotropic sphere. A scale factor of 1 alters the eigenvalues in such a way that the resulting ellipsoid is fully eccentric and anisotropic

Parameters

- **D** (*ndarray of float [n x 9]*) – Diffusion tensor
- **s** (*float*) – Scale parameter [0 (iso) … 0.5 (unaltered)… 1 (aniso)]
- **tsc** (*boolean*) – Tensor singularity correction

Returns
Ds – Scaled diffusion tensor

Return type
list of nparray of float [3 x 3]

1.1.12 pynibs.tms_pulse module

pynibs.tms_pulse.**biphasic_pulse**(*t*, *R=0.0338*, *L=1.55e-05*, *C=0.0001936*, *alpha=1089.8*, *f=2900*)

Returns normalized single biphasic pulse waveform of electric field (first derivative of coil current)

Parameters

- **t** (*ndarray of float [n_t]*) – Time array in seconds
- **R** (*float, optional, default: 0.0338 Ohm*) – Resistance of coil in (Ohm)
- **L** (*float, optional, default: 15.5*1e-6 H*) – Inductance of coil in (H)
- **C** (*float, optional, default: 193.6*1e-6*) – Capacitance of coil in (F)
- **alpha** (*float, optional, default: 1089.8 1/s*) – Damping coefficient in (1/s)
- **f** (*float, optional, default: 2900 Hz*) – Frequency in (Hz)

Returns
e – Normalized electric field time course (can be scaled with electric field)

Return type
ndarray of float [n_t]

- genindex
- modindex
- search

CHAPTER

TWO

REFERENCES.

- Weise, K., Numssen, O., Thielscher, A., Hartwigsen, G., & Knösche, T. R. (2020). A novel approach to localize cortical TMS effects. Neuroimage, 209, 116486. doi: 10.1016/j.neuroimage.2019.116486)
- Numssen, O., Zier, A. L., Thielscher, A., Hartwigsen, G., Knösche, T. R., & Weise, K. (2021). Efficient high-resolution TMS mapping of the human motor cortex by nonlinear regression. NeuroImage, 245, 118654. doi:10.1016/j.neuroimage.2021.118654)
- Weise, K., Numssen, O., Kalloch, B., Zier, A. L., Thielscher, A., Hartwigsen, G., Knösche, T. R. (2022). Precise transcranial magnetic stimulation motor-mapping. Nature Protocols (in press). doi: 10.1038/s41596-022-00776-6

BIBLIOGRAPHY

[1] Weise, K., Numssen, O., Thielscher, A., Hartwigsen, G., & Knösche, T. R. (2020). A novel approach to localize cortical TMS effects. Neuroimage, 209, 116486.

[2] Numssen, O., Zier, A. L., Thielscher, A., Hartwigsen, G., Knösche, T. R., & Weise, K. (2021). Efficient high-resolution TMS mapping of the human motor cortex by nonlinear regression. NeuroImage, 245, 118654.

[1] Opitz, A., Legon, W., Rowlands, A., Bickel, W. K., Paulus, W., & Tyler, W. J. (2013). Physiological observations validate finite element models for estimating subject-specific electric field distributions induced by transcranial magnetic stimulation of the human motor cortex. Neuroimage, 81, 253-264.

[1] Aonuma, S., Gomez-Tames, J., Laakso, I., Hirata, A., Takakura, T., Tamura, M., & Muragaki, Y. (2018). A high-resolution computational localization method for transcranial magnetic stimulation mapping. NeuroImage, 172, 85-93.

[1] Bungert, A., Antunes, A., Espenhahn, S., & Thielscher, A. (2016). Where does TMS stimulate the motor cortex? Combining electrophysiological measurements and realistic field estimates to reveal the affected cortex position. Cerebral Cortex, 27(11), 5083-5094.

[1] Goetz, S. M., Luber, B., Lisanby, S. H., & Peterchev, A. V. (2014). A novel model incorporating two variability sources for describing motor evoked potentials. Brain stimulation, 7(4), 541-552.

[1] Goetz, S. M., Alavi, S. M., Deng, Z. D., & Peterchev, A. V. (2019). Statistical Model of Motor-Evoked Potentials. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 27(8), 1539-1545.

[1] Zienkiewicz, Olgierd Cecil, and Jian Zhong Zhu. "The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique." International Journal for Numerical Methods in Engineering 33.7 (1992): 1331-1364.

[1] Numssen, O., Zier, A. L., Thielscher, A., Hartwigsen, G., Knösche, T. R., & Weise, K. (2021). Efficient high-resolution TMS mapping of the human motor cortex by nonlinear regression. NeuroImage, 245, 118654.

[1] Dale, A.M., Fischl, B., Sereno, M.I., 1999. Cortical surface-based analysis. I. Segmentation and surface reconstruction. Neuroimage 9, 179-194.

[2] https://surfer.nmr.mgh.harvard.edu/fswiki/SurfaceRegAndTemplates

[1] Devillers, Olivier, Sylvain Pion, and Monique Teillaud. "Walking in a triangulation." International Journal of Foundations of Computer Science 13.02 (2002): 181-199.

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z