PyHRF Documentation

Release

Dec 16, 2017

Contents

1 Subpackages 3

2 Submodules 229

Python Module Index 261

PyHRF is a set of tools for within-subject fMRI data analysis, which focuses on the characterization of the hemodynamics.

Within the chain of fMRI data processing, these tools provide alternatives to the classical within-subject GLM estimation step. The inputs are preprocessed within-subject data and the outputs are statistical maps and/or fitted HRFs.

The package is mainly written in Python and provides the implementation of the two following methods:

- The joint-detection estimation (JDE) approach, which divides the brain into functionally homogeneous regions and provides one HRF estimate per region as well as response levels specific to each voxel and each experimental condition. This method embeds a temporal regularization on the estimated HRFs and an adaptive spatial regularization on the response levels.
- The Regularized Finite Impulse Response (RFIR) approach, which provides HRF estimates for each voxel and experimental conditions. This method embeds a temporal regularization on the HRF shapes, but proceeds independently across voxels (no spatial model).

Check the PyHRF website for details.

class pyhrf.**Verbose**(*verbosity=0*, *log=<open file '<stdout>', mode 'w'>*)

Bases: pyhrf._verbose.Verbose

This is a dummy class implementing the original Verbose class.

This is only to be able to raise a warning when one uses this old implementation.

old_to_new_log_dict = {0: 30, 1: 20, 2: 20, 3: 20, 4: 20, 5: 10, 6: 10}

CHAPTER 1

Subpackages

1.1 pyhrf.boldsynth package

Module designed to the BOLD signal synthesis according to the Linear and Time Invariant model described in:

- Makni, S., Ciuciu, P., Idier, J., & Poline, J.-B. (2005). Joint detection-estimation of brain activity in functional MRI: a Multichannel Deconvolution solution. IEEE Transactions on Signal Processing, 53(9), 3488–3502. http://doi.org/10.1109/TSP.2005.853303

It also provides a set of plotting functions (based on matplotlib) - see L{bolsynth.plot}

1.1.1 Subpackages

pyhrf.boldsynth.pottsfield package

Submodules

pyhrf.boldsynth.pottsfield.swendsenwang module

pyhrf.boldsynth.pottsfield.swendsenwang.**CptDefaultGraphLinks**(*RefGraph*)
computes a default list GraphLinks from RefGraph

input:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2

output:

- GraphLinks: Same shape as RefGraph. Each entry indicates whether the link of the corresponding edge in RefGraph is considered or not in Swensdsen-Wang Sampling (1 -> yes / 0 -> no).

pyhrf.boldsynth.pottsfield.swendsenwang.**CptDefaultGraphNodesLabels**(*RefGraph*)
computes a default list GraphNodesLabels from RefGraph

input:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2

output:

- GraphNodesLabels: List containing the nodes labels (consiedered for the computation of U). The sampler aims to modify its values in function of beta and NbLabels.

pyhrf.boldsynth.pottsfield.swendsenwang.**CptDefaultGraphWeight**(*RefGraph*)
computes a default list GraphWeight from RefGraph. Each edge weight is set to 1.0.

input:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2

output:

- GraphWeight: Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph

pyhrf.boldsynth.pottsfield.swendsenwang.**CptRefGrphNgbhPosi**(*RefGraph*)
computes the critical list CptRefGrphNgbhPosi from RefGraph

imput:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2

output:

- RefGrphNgbhPosi: Same shape as RefGraph. RefGrphNgbhPosi[i][j] indicates for which k is the link to i in RefGraph[RefGraph[i][j]][k]. It makes algorithms which run through the graph much faster since it avoids a critical loop.

pyhrf.boldsynth.pottsfield.swendsenwang.**Cpt_U_graph**(*RefGraph*, *GraphNodesLabels*, *GraphWeight=None*)
Computes an estimation of U(Graph)

inputs:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2
- GraphNodesLabels: list containing the nodes labels.
- GraphWeight: Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph. If not defined the weights are set to 1.0.

output:

- U value

pyhrf.boldsynth.pottsfield.swendsenwang.**Cpt_Vec_U_graph**(*RefGraph*, *beta*, *LabelsNb*, *SamplesNb*, *GraphWeight=None*, *GraphNodesLabels=None*, *GraphLinks=None*, *RefGrphNgbhPosi=None*)

Computes a given number of U for fields generated according to a given normalization constant Beta. Swendsen-Wang sampling is used to generate fields.

input:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2
- beta: normalization constant
- LabelsNb: Labels number
- SamplesNb: Samples number for the U estimations
- GraphWeight: Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph. If not defined the weights are set to 1.0.
- GraphNodesLabels: Optional list containing the nodes labels. The sampler aims to modify its values in function of beta and NbLabels. At this level this variable is seen as temporary and will be modified. Defining it slightly increases the calculation times.
- GraphLinks: Same shape as RefGraph. Each entry indicates if the link of the corresponding edge in RefGraph is considered (if yes ...=1 else ...=0). At this level this variable is seen as temporary and will be modified. Defining it slightly increases the calculation times.
- RefGrphNgbhPosi: Same shape as RefGraph. RefGrphNgbhPosi[i][j] indicates for which k is the link to i in RefGraph[RefGraph[i][j]][k]. This optional list is never modified.

output:

- VecU: Vector of size SamplesNb containing the U computations

pyhrf.boldsynth.pottsfield.swendsenwang.**GraphBetaMix**(*RefGraph*, *GraphNodesLabels*, *beta=0.5*, *NbLabels=2*, *NbIt=5*, *weights=None*)

Generate a partition in GraphNodesLabels with respect to beta.

input:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2
- GraphNodesLabels: list containing the nodes labels.
- beta: correlation factor for all conditions
- NbLabels: number of labels in all site conditions
- NbIt: number of sampling steps with SwendsenWangSampler_graph

output:

- GraphNodesLabels: sampled GraphNodesLabels (not returned but modified)

pyhrf.boldsynth.pottsfield.swendsenwang.**GraphToImage**(*GraphNodesCoord*, *GraphNodesLabels*, *NBZ*, *NBY*, *NBX*)

Computes a 3D image from a connectivity graph.

input:

- GraphNodesCoord: Coordinates of each node in Mask
- GraphNodesLabels: Nodes labels. For example, GraphNodesLabels[i] is the label of node i.
- NBZ: image size on Z axis
- NBY: image size on Y axis
- NBX: image size on X axis

output:

- Image

pyhrf.boldsynth.pottsfield.swendsenwang.**ImageToGraph**(*Image*, *Mask*, *LabelOI=1*, *ConnectivityType=6*)

Computes the connectivity graph of an image under a 3D mask voxels.

inputs:

- Image: the 3D image (label field).
- Mask: corresponding 3D mask.
- LabelOI: Voxels of Mask containing the label LabelOI are those considered.
- ConnectivityType: controles the connectivity considered in the graph. ConnectivityType=26 : 26-connectivity ConnectivityType=18 : 18-connectivity ConnectivityType=6 : 6-connectivity

outputs:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2
- GraphWeight: Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph
- GraphNodesCoord: Coordinates of each node in Mask
- GraphNodesLabels: list containing the nodes labels.

pyhrf.boldsynth.pottsfield.swendsenwang.**MaskToGraph**(*Mask*, *LabelOI=1*, *ConnectivityType=6*)

Computes the connectivity graph of in 3D mask voxels.

inputs:

- Mask: 3D mask.
- LabelOI: Voxels of Mask containing the label LabelOI are those considered.
- ConnectivityType: controles the connectivity considered in the graph. ConnectivityType=26 : 26-connectivity ConnectivityType=18 : 18-connectivity ConnectivityType=6 : 6-connectivity

outputs:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2
- GraphWeight: Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph
- GraphNodesCoord: Coordinates of each node in Mask

pyhrf.boldsynth.pottsfield.swendsenwang.**SwendsenWangSampler_graph**(*RefGraph*, *GraphNodesLabels*, *beta*, *NbLabels*, *GraphLinks=None*, *RefGrphNgbhPosi=None*, *method=1*, *weights=None*)

image sampling with Swendsen-Wang algorithm

input:

- RefGraph: List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. ex: RefGraph[2][3]=10 means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that RefGraph[10][i]=2
- GraphNodesLabels: list containing the nodes labels. The sampler aims to modify its values in function of beta and NbLabels.
- beta: normalization constant
- NbLabels: number of labels (connected voxel pairs are considered if their labels are equal)
- GraphLinks: Same shape as RefGraph. Each entry indicates if the link of the corresponding edge in RefGraph is considered (if yes ...=1 else ...=0). This optional list is used as a temporary variable and will be modified ! Defining it makes the algorithm faster (no memory allocation).
- RefGrphNgbhPosi: Same shape as RefGraph. RefGrphNgbhPosi[i][j] indicates for which k is the link to i in RefGraph[RefGraph[i][j]][k] This optional list is never modified.

output:

- GraphNodesLabels: resampled nodes labels. (not returned but modified)

pyhrf.boldsynth.pottsfield.swendsenwang.**linkNodes**(*RefGraph*, *beta*, *GraphNodesLabels*, *GraphLinks*, *RefGrphNgbhPosi*)

pyhrf.boldsynth.pottsfield.swendsenwang.**linkNodesSets**(*RefGraph*, *beta*, *GraphNodesLabels*, *links*, *weights=None*)

pyhrf.boldsynth.pottsfield.swendsenwang.**pickLabels**(*RefGraph*, *GraphLinks*, *GraphNodesLabels*, *NbLabels*, *TempVec*, *NextTempVec*)

pyhrf.boldsynth.pottsfield.swendsenwang.**set_cluster_labels**(*links*, *labels*, *nbClasses*)

pyhrf.boldsynth.pottsfield.swendsenwang.**walkCluster**(*i*, *links*, *labels*, *l*, *remainingIndexes*)

1.1.2 Submodules

pyhrf.boldsynth.field module

pyhrf.boldsynth.field.**count_homo_cliques**(*graph*, *labels*, *weights=None*)

pyhrf.boldsynth.field.**genPepperSaltField**(*size*, *nbLabels*, *initProps=None*)

pyhrf.boldsynth.field.**genPotts**(*graph*, *beta*, *nbLabels=2*, *labelsIni=None*, *method='SW'*, *weights=None*)

Simulate a realisation of a Potts Field with spatial correlation amount 'beta'. 'graph' is list of lists, ie a neighbors for each node index. 'nbLabels' is the number of labels 'method' can be either 'SW' (swensdsen-wang) or 'gibbs'

pyhrf.boldsynth.field.**genPottsMap**(*mask*, *beta*, *nbLabels*, *method='SW'*)

pyhrf.boldsynth.field.**potts_generator**(***args*)

class pyhrf.boldsynth.field.**random_field_generator**(*size*, *nbClasses*)

pyhrf.boldsynth.hrf module

pyhrf.boldsynth.hrf.**bezierCurve**(*p1*, *pc1*, *p2*, *pc2*, *xPrecision*)

pyhrf.boldsynth.hrf.**buildFiniteDiffMatrix**(*order*, *size*)

pyhrf.boldsynth.hrf.**genBezierHRF**(*timeAxis=array([0. , 0.6, 1.2, 1.8, 2.4, 3. , 3.6, 4.2, 4.8, 5.4, 6. , 6.6, 7.2, 7.8, 8.4, 9. , 9.6, 10.2, 10.8, 11.4, 12. , 12.6, 13.2, 13.8, 14.4, 15. , 15.6, 16.2, 16.8, 17.4, 18. , 18.6, 19.2, 19.8, 20.4, 21. , 21.6, 22.2, 22.8, 23.4, 24. , 24.6, 25.2]), pic=[6, 1], picw=2, ushoot=[15, -0.2], ushootw=3, normalize=False*)

pyhrf.boldsynth.hrf.**genCanoBezierHRF**(*duration=25.0*, *dt=0.6*, *normalize=False*)

pyhrf.boldsynth.hrf.**genExpHRF**(*timeAxis=array([0., 0.5, 1., 1.5, 2., 2.5, 3., 3.5, 4., 4.5, 5., 5.5, 6., 6.5, 7., 7.5, 8., 8.5, 9., 9.5, 10., 10.5, 11., 11.5, 12., 12.5, 13., 13.5, 14., 14.5, 15., 15.5, 16., 16.5, 17., 17.5, 18., 18.5, 19., 19.5, 20., 20.5, 21., 21.5, 22., 22.5, 23., 23.5, 24., 24.5])*, *ttp=6*, *pa=1*, *pw=0.2*, *ttu=11*, *ua=0.2*, *uw=0.01*)

pyhrf.boldsynth.hrf.**genGaussianSmoothHRF**(*zc*, *length*, *eventdt*, *rh*, *order=2*)

pyhrf.boldsynth.hrf.**genPriorCov**(*zc*, *pprcov*, *dt*)

pyhrf.boldsynth.hrf.**getCanoHRF**(*duration=25*, *dt=0.6*, *hrf_from_spm=True*, *delay_of_response=6.0*, *delay_of_undershoot=16.0*, *dispersion_of_response=1.0*, *dispersion_of_undershoot=1.0*, *ratio_resp_under=6.0*, *delay=0.0*)

Compute the canonical HRF.

Parameters

- **duration** (*int or float, optional*) – time lenght of the HRF in seconds
- **dt** (*float, optional*) – time resolution of the HRF in seconds
- **hrf_from_spm** (*bool, optional*) – if True, use the SPM formula to compute the HRF, if False, use the hard coded values and resample if necessary. It is strongly advised to use True.
- **delay_of_response** (*float, optional*) – delay of the first peak response in seconds
- **delay_of_undershoot** (*float, optional*) – delay of the second undershoot peak in seconds
- **dispersion_of_response** (*float, optional*) –
- **dispersion_of_undershoot** (*float, optional*) –

- **ratio_resp_under** (*float, optional*) – ratio between the response peak and the undershoot peak
- **delay** (*float, optional*) – delay of the HRF

Returns

- **time_axis** (*ndarray, shape (round(duration/dt)+1,)*) – time axis of the HRF in seconds
- **HRF** (*ndarray, shape (round(duration/dt)+1,)*)

pyhrf.boldsynth.hrf.**getCanoHRF_tderivative**(*duration=25.0*, *dt=0.5*)

pyhrf.boldsynth.scenarios module

pyhrf.boldsynth.scenarios.**build_ctrl_tag_matrix**(*asl_shape*)

pyhrf.boldsynth.scenarios.**calc_asl_shape**(*bold_stim_induced*, *dsf*)

pyhrf.boldsynth.scenarios.**createBiGaussCovarNRL**(*condition_defs*, *labels*, *covariance*)

pyhrf.boldsynth.scenarios.**create_3Dlabels_Potts**(*condition_defs*, *beta*, *dims*, *mask*)

pyhrf.boldsynth.scenarios.**create_AR_noise**(*bold_shape*, *v_noise*, *order=2*, *v_corr=0.1*)

pyhrf.boldsynth.scenarios.**create_Xh**(*nrls*, *rastered_paradigm*, *hrf*, *condition_defs*, *dt*, *hrf_territories=None*)

Retrieve the product X.h

pyhrf.boldsynth.scenarios.**create_alpha_for_hrfgroup**(*alpha_var*)

Create alpha from a normal distribution, for one subject

pyhrf.boldsynth.scenarios.**create_asl_from_stim_induced**(*bold_stim_induced*, *perf_stim_induced*, *ctrl_tag_mat*, *dsf*, *perf_baseline*, *noise*, *drift=None*, *outliers=None*)

Downsample stim_induced signal according to downsampling factor 'dsf' and add noise and drift (nuisance signals) which has to be at downsampled temporal resolution.

pyhrf.boldsynth.scenarios.**create_bigaussian_nrls**(*labels*, *mean_act*, *var_act*, *var_inact*)

Simulate bi-Gaussian NRLs (zero-centered inactive component)

pyhrf.boldsynth.scenarios.**create_bold**(*stim_induced_signal*, *dsf*, *noise*, *drift=None*, *outliers=None*)

a short-cut for function create_bold_from_stim_induced

pyhrf.boldsynth.scenarios.**create_bold_controlled_variance**(*stim_induced_signal*, *alpha*, *nb_voxels*, *dsf*, *nrls*, *Xh*, *drift=None*, *outliers=None*)

Create BOLD with controlled explained variance alpha: percentage of explained variance on total variance

pyhrf.boldsynth.scenarios.**create_bold_from_stim_induced**(*stim_induced_signal*, *dsf*, *noise*, *drift=None*, *outliers=None*)

Downsample stim_induced signal according to downsampling factor 'dsf' and add noise and drift (nuisance signals) which has to be at downsampled temporal resolution.

pyhrf.boldsynth.scenarios.**create_bold_from_stim_induced_RealNoise**(*stim_induced_signal*, *dsf*, *noise*, *drift*)

Downsample stim_induced signal according to downsampling factor 'dsf' and add noise and drift (nuisance signals) which has to be at downsampled temporal resolution.

pyhrf.boldsynth.scenarios.**create_bold_stim_induced_signal**(*brls*, *rastered_paradigm*, *brf*, *condition_defs*, *dt*, *hrf_territories=None*)

Create a stimulus induced signal for ASL from BOLD response levels, paradigm and BRF (sum_{m=1}^M a^m X^m h + sum_{m=1}^M c^m W X^m g) For each condition, compute the convolution of the paradigm binary sequence 'rastered_paradigm' with the given BRF and multiply by brls. Finally compute the sum over conditions.

Return a asl array of shape (nb scans, nb voxels)

pyhrf.boldsynth.scenarios.**create_canonical_hrf**(*hrf_duration=25.0*, *dt=0.5*)

pyhrf.boldsynth.scenarios.**create_connected_label_clusters**(*condition_defs*, *activ_label_graph*)

pyhrf.boldsynth.scenarios.**create_drift_coeffs**(*bold_shape*, *drift_order*, *drift_coeff_var*)

pyhrf.boldsynth.scenarios.**create_drift_coeffs_asl**(*asl_shape*, *drift_order*, *drift_var*)

pyhrf.boldsynth.scenarios.**create_gaussian_hrf_subject**(*hrf_group*, *var_subject_hrf*, *dt*, *alpha=0.0*)

Creation of hrf by subject. Use group level hrf and variance for each subject (var_subjects_hrfs must be a list) Simulated hrfs must be smooth enough: correlation between temporal coeffcients

pyhrf.boldsynth.scenarios.**create_gaussian_noise**(*bold_shape*, *v_noise*, *m_noise=0.0*)

pyhrf.boldsynth.scenarios.**create_gaussian_noise_asl**(*asl_shape*, *v_gnoise*, *m_noise=0.0*)

pyhrf.boldsynth.scenarios.**create_gaussian_nrls_sessions_and_mean**(*nrls*, *condition_defs*, *labels*, *var_sess*)

Creation of nrls by session (and by voxel and cond) - for one session n° sess The nrls by session vary around an nrl mean (nrls_bar) defined by voxel and cond (var_sess corresponds to the variation of session defined nrls around the nrl_bar) Here "nrls" is nrls_bar, mean over subjects!

pyhrf.boldsynth.scenarios.**create_gsmooth_hrf**(*hrf_duration=25.0*, *dt=0.5*, *order=2*, *hrf_var=1.0*, *zc=True*, *normalize_hrf=True*)

Create a smooth HRF according to the multivariate gaussian prior used in JDE *hrf_duration* and *dt* are the HRF duration and temporal resolution, respectively (in sec.). *order* is derivative order constraining the covariance matrix. *hrf_var* is the HRF variance. *zc* is a flag to impose zeros at the begining and the end of the HRF

return: a np array of HRF coefficients

pyhrf.boldsynth.scenarios.**create_hrf**(*picw*, *pic*, *under=2*, *hrf_duration=25.0*, *dt=0.5*)

pyhrf.boldsynth.scenarios.**create_hrf_from_territories**(*hrf_territories*, *primary_hrfs*)

pyhrf.boldsynth.scenarios.**create_labels_Potts**(*condition_defs*, *beta*, *nb_voxels*)

pyhrf.boldsynth.scenarios.**create_labels_vol**(*condition_defs*)

Create a seet labels from the field "label_map" in *condition_defs* Available choices for the field label_map: - 'random_small' : binary labels are randomly generated with shape (1,5,5) - a tag (str) : corresponds to a png file in pyhrf data files - a 3D np containing the labels

pyhrf.boldsynth.scenarios.**create_language_paradigm**(*condition_defs*)

pyhrf.boldsynth.scenarios.**create_localizer_paradigm**(*condition_defs*, *paradigm_label='av'*)

pyhrf.boldsynth.scenarios.**create_localizer_paradigm_a**(*condition_defs*)

pyhrf.boldsynth.scenarios.**create_localizer_paradigm_avd**(*condition_defs*)

pyhrf.boldsynth.scenarios.**create_multisess_stim_induced_signal**(*nrls_session*, *rastered_paradigm*, *hrf*, *condition_defs*, *dt*, *hrf_territories=None*)

Create a stimulus induced signal from neural response levels, paradigm and HRF (sum_{m=1}^M a^m X^m h) For each condition, compute the convolution of the paradigm binary sequence 'rastered_paradigm' with the given HRF and multiply by nrls. Finally compute the sum over conditions.

Return a bold array of shape (nb scans, nb voxels)

pyhrf.boldsynth.scenarios.**create_multisess_stim_induced_signal_asl**(*prls_session*, *rastered_paradigm*, *prf*, *condition_defs*, *dt*, *hrf_territories=None*)

Create a stimulus induced signal from neural response levels, paradigm and HRF (sum_{m=1}^M a^m X^m h) For each condition, compute the convolution of the paradigm binary sequence 'rastered_paradigm' with the given HRF and multiply by nrls. Finally compute the sum over conditions.

Return a bold array of shape (nb scans, nb voxels)

pyhrf.boldsynth.scenarios.**create_null_drift**(*bold_shape*)

pyhrf.boldsynth.scenarios.**create_outliers**(*bold_shape*, *stim_induced_signal*, *nb_outliers*, *outlier_scale=5.0*)

pyhrf.boldsynth.scenarios.**create_paradigm_un_evnt**(*condition_defs*)

pyhrf.boldsynth.scenarios.**create_perf_baseline**(*asl_shape*, *perf_baseline_var*, *perf_baseline_mean=0.0*)

pyhrf.boldsynth.scenarios.**create_perf_stim_induced_signal**(*prls*, *rastered_paradigm*, *prf*, *condition_defs*, *dt*, *hrf_territories=None*)

Create a stimulus induced signal for ASL from perfusion response levels, paradigm and PRF (sum_{m=1}^M c^m X^m g) For each condition, compute the convolution of the paradigm binary sequence 'rastered_paradigm' with the given PRF and multiply by prls. Finally compute the sum over conditions.

Return a asl array of shape (nb scans, nb voxels)

pyhrf.boldsynth.scenarios.**create_polynomial_drift**(*bold_shape*, *tr*, *drift_order*, *drift_var*)

pyhrf.boldsynth.scenarios.**create_polynomial_drift_from_coeffs**(*bold_shape*, *tr*, *drift_order*, *drift_coeffs*, *drift_mean=0.0*, *drift_amplitude=1.0*)

pyhrf.boldsynth.scenarios.**create_polynomial_drift_from_coeffs_asl**(*asl_shape*, *tr*, *drift_order*, *drift_coeffs*)

pyhrf.boldsynth.scenarios.**create_prf**(*prf_duration=25.0*, *dt=0.5*)

pyhrf.boldsynth.scenarios.**create_small_bold_simulation**(*snr='high'*, *output_dir=None*, *simu_items=None*)

pyhrf.boldsynth.scenarios.**create_stim_induced_signal**(*nrls*, *rastered_paradigm*, *hrf*, *dt*)

Create a stimulus induced signal from neural response levels, paradigm and HRF (sum_{m=1}^M a^m X^m h) For each condition, compute the convolution of the paradigm binary sequence 'rastered_paradigm' with the given HRF and multiply by nrls. Finally compute the sum over conditions.

Return a bold array of shape (nb scans, nb voxels)

pyhrf.boldsynth.scenarios.**create_stim_induced_signal_Parsi**(*nrls*, *rastered_paradigm*, *hrf*, *condition_defs*, *dt*, *w*)

Create a stimulus induced signal from neural response levels, paradigm and HRF (sum_{m=1}^M a^m w^m X^m h) For each condition, compute the convolution of the paradigm binary sequence 'rastered_paradigm' with the given HRF and multiply by nrls and W. Finally compute the sum over conditions.

Return a bold array of shape (nb scans, nb voxels)

pyhrf.boldsynth.scenarios.**create_time_invariant_gaussian_brls**(*condition_defs*, *labels*)

BOLD response levels for ASL

pyhrf.boldsynth.scenarios.**create_time_invariant_gaussian_nrls**(*condition_defs*, *labels*)

pyhrf.boldsynth.scenarios.**create_time_invariant_gaussian_prls**(*condition_defs*, *labels*)

Perfusion response levels for ASL

pyhrf.boldsynth.scenarios.**create_varying_hrf**(*hrf_duration=25.0*, *dt=0.5*)

pyhrf.boldsynth.scenarios.**duplicate_brf**(*nb_voxels*, *primary_brf*)
Duplicate brf over all voxels. Return an array of shape (nb_voxels, len(brf))

pyhrf.boldsynth.scenarios.**duplicate_hrf**(*nb_voxels*, *primary_hrf*)
Duplicate hrf over all voxels. Return an array of shape (nb_voxels, len(hrf))

pyhrf.boldsynth.scenarios.**duplicate_noise_var**(*nb_voxels*, *v_gnoise*)
Duplicate variance of noise over all voxels. Return an array of shape (nb_voxels, var noise)

pyhrf.boldsynth.scenarios.**duplicate_prf**(*nb_voxels*, *primary_prf*)
Duplicate prf over all voxels. Return an array of shape (nb_voxels, len(prf))

pyhrf.boldsynth.scenarios.**flatten_labels_vol**(*labels_vol*)

pyhrf.boldsynth.scenarios.**get_bold_shape**(*stim_induced_signal*, *dsf*)

pyhrf.boldsynth.scenarios.**load_drawn_labels**(*name*)

pyhrf.boldsynth.scenarios.**load_hrf_territories**(*nb_hrf_territories=0*, *hrf_territories_name=None*)

pyhrf.boldsynth.scenarios.**load_many_hrf_territories**(*nb_hrf_territories*)

pyhrf.boldsynth.scenarios.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.boldsynth.scenarios.**rasterize_paradigm**(*paradigm*, *dt*, *condition_defs*)

Return binary sequences of onsets approximated on temporal grid of temporal resolution dt, for all conditions. 'paradigm' is expected to be an instance of 'pyhrf.paradigm.mpar.Paradigm'

pyhrf.boldsynth.scenarios.**save_simulation**(*simulation*, *output_dir*)

short-hand for simulation_save_vol_outputs

pyhrf.boldsynth.scenarios.**simulation_save_vol_outputs**(*simulation*, *output_dir*, *bold_3D_vols_dir=None*, *simulation_graph_output=None*, *prefix=None*, *vol_meta=None*)

simulation_graph_output : None, 'simple', 'thumbnails' #TODO

pyhrf.boldsynth.spatialconfig module

class pyhrf.boldsynth.spatialconfig.**Mapper1D**(*mapping*, *expandedShape*)
Handles a mapping between a nD coordinate space (expanded) and a 1D coordinate space (flatten). Can be applied to numpy.ndarray objects

createExpandedArray(*flatShape*, *type*, *mappedAxis=0*, *fillValue=0*)

expandArray(*a*, *mappedAxis=0*, *dest=None*, *fillValue=0*)
Expand dimensions of 'a' following predefined mapping. 'mappedAxis' is the axis index to be expanded in 'a'. If dest is not None the map values from 'a' to dest. If dest is None then return a new array and fill positions not involved in mapping with 'fillValue'.

flattenArray(*array*, *firstMappedAxis=0*)
Reduce dimensions of 'array'. 'firstMappedAxis' is index of the axis to be reduced (other mapped axes are assumed to follow this one).

class pyhrf.boldsynth.spatialconfig.**NeighbourhoodSystem**(*neighboursSets*)

static fromLattice(*latticeIndexes*, *kerMask=None*, *depth=1*, *torusFlag=False*)
Creates a NeighbourhoodSystem instance from a n-dimensional lattice

static fromMesh(*polygonList*)

getMaxIndex()

getMaxNeighbours()

getNeighbours(*nodeId*)

getNeighboursArrays()

getNeighboursLists()

getNeighboursSets()

kerMask2D_4n = array([[-1, 0], [1, 0], [0, 1], [0, -1]])

kerMask3D_6n = array([[1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0], [0, -1, 0], [0

sub(*nodeIds*)

class pyhrf.boldsynth.spatialconfig.**PottsField**(*classNames*, *spConf*, *initProps=None*)
Bases: *pyhrf.boldsynth.spatialconfig.StateField*

class pyhrf.boldsynth.spatialconfig.**RegularLatticeMapping**(*shape=None*, *mapping=None*, *order=1*, *depth=1*)
Bases: *pyhrf.boldsynth.spatialconfig.SpatialMapping*

Define a SpatialMapping on a 3D regular lattice.

buildNeighboursCoordLists()

buildNeighboursIndexLists()

c = [1, 1, 1]

static createFromGUI(*GUIobject*)
Creates the actual object based on the parameters

getClosestNeighboursIndexes(*idVoxel*)

getCoord(*index*)

getIndex(*coord*)

getMapping()

getNbCliques()

getNbVoxels()

getNdArrayMask()

getNeighboursCoordLists()

getNeighboursCoords(*idvoxel*)

getNeighboursIndexLists()

getNeighboursIndexes(*idVoxel*)

getRoiMask()
Return a binary or n-ary 3D mask which has the shape of the target data

getTargetAxesNames()

mapVoxData(*data*, *fillValue=0*)

nbNeighboursOrder1 = 6

nbNeighboursOrder2 = 26

order1Mask = array([[1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0], [0, -1, 0], [0,

order2Mask = array([[0, 0, -1], [0, 0, 1], [0, -1, 0], [0, -1, -1], [0, -1, 1], [

class pyhrf.boldsynth.spatialconfig.**RegularLatticeMapping2**(*maskLattice*, *kerMask=None*, *nsDepth=1*, *parentMapping=None*, *torusFlag=False*)

Bases: *pyhrf.boldsynth.spatialconfig.SpatialMapping2*

flattenData(*data*, *firstMappedAxis=0*)

mapData(*data*, *mappedAxis=0*, *fillValue=0*)

class pyhrf.boldsynth.spatialconfig.**SpatialMapping**
Interface specification for the handling of a mapping between integer indexes and positions in a 3D space.

getCoord(*index*)
Return coord mapped with 'index'

getIndex(*coord*)
Return index mapped with 'coord'

getMapping()
Return a mapping object (list or dict) which maps an integer index to its 3D coordinates.

getNbVoxels()
Return the total number of mapped position

getNdArrayMask()
Return the set of mapped 3D coordinates in a tuple usable as a mask for numpy.ndarray

getNeighboursCoordLists()
Get lists of neighbours for all positions @param idVoxel: index of the voxel @return: a mapping object (list or dict) which maps each integer index to a list of 3D coordinates (the neighbours).

getNeighboursCoords(*idVoxel*)
@param idVoxel: index of the voxel @return: the list of 3D coordinates corresponding to the neighbours of the specified voxel.

getNeighboursIndexLists()
Get lists of neighbours for all positions @param idVoxel: index of the voxel @return: a mapping object (list or dict) which maps each integer index to a list of integer indexes (the neighbours).

getNeighboursIndexes(*idVoxel*)
@param idVoxel: index of the voxel @return: the list of integer indexes corresponding to the neighbours of the specified voxel.

getRoiMask()
Return a binary or n-ary mask which has the shape of the target data

class pyhrf.boldsynth.spatialconfig.**SpatialMapping2**(*positions*, *ns*, *parentMapping=None*, *parentIndex=None*)

static fromLattice(*lattice*, *kerMask=None*, *nsDepth=1*, *torusFlag=False*)

static fromMesh(*triangles*, *positions*)

getPositions()

sub(*nodeIds*)

class pyhrf.boldsynth.spatialconfig.**StateField**(*classNames*, *spConf*, *initProps=None*)
Class handling a field of states : a set of integers (ie labels) whose ranks can be spatially mapped to 3D coordinates. Each label refers to a class wich is identified by an ID and a name.

generate()
Generate values for every states. By default : if initProportions is set, generate values according to it. State values will be ordered by class ID

getClassId(*className*)
Return the class id corresponding to the string 'className'

getClassName(*classId*)
Return the class name corresponding to the integer 'classId'

getClassNames()
Return all the class names

getFieldValues()
Return all field values.

getMappedFieldValues()

getNbClasses()

getSize()
Return to size of the field.

randomize()
Randomize state values with a ramdom permutation.

setFieldValues(*values*, *mask=None*)
Copy the content of 'values' to state values masked by 'mask'.

setFieldValues0(*values*, *mask=None*)
Copy the content of 'values' to state values masked by 'mask'.

updateClassCounts()
Compute the size of every classes.

class pyhrf.boldsynth.spatialconfig.**UnboundSpatialMapping**(*nbVoxels=100*)
Bases: *pyhrf.boldsynth.spatialconfig.SpatialMapping*

Convinient class to provide an implementation of SpatialMapping when there is no mapping.

static createFromGUI(*GUIobject*)
Creates the actual object based on the parameters

getCoord(*index*)

getIndex(*coord*)

getMapping()

getNbVoxels()

getNdArrayMask()

getNeighboursCoordLists()

getNeighboursCoords(*idVoxel*)

getNeighboursIndexLists()

getNeighboursIndexes(*idVoxel*)

getRoiMask()
Return a binary or n-ary 3D mask which has the shape of the target data

pyhrf.boldsynth.spatialconfig.**flattenElements**(*l*)

pyhrf.boldsynth.spatialconfig.**getRotationMatrix**(*axis*, *angle*)
Compute the 3x3 matrix for the 3D rotation defined by 'angle' and the direction 'axis'.

pyhrf.boldsynth.spatialconfig.**hashMask**(*m*)

pyhrf.boldsynth.spatialconfig.**lattice_indexes**(*mask*)

pyhrf.boldsynth.spatialconfig.**maskToMapping**(*m*)

pyhrf.boldsynth.spatialconfig.**mask_to_coords**(*m*)

class pyhrf.boldsynth.spatialconfig.**xndarrayMapper1D**(*mapping*, *expandedShape*, *expandedAxesNames*, *flatAxisName*)
Bases: *pyhrf.boldsynth.spatialconfig.Mapper1D*

expandxndarray(*c*, *dest=None*, *fillValue=0*)

flattenxndarray(*c*)

isExpendable(*c*)

isFlattenable(*c*)

1.2 pyhrf.jde package

1.2.1 Subpackages

pyhrf.jde.nrl package

Submodules

pyhrf.jde.nrl.ar module

class pyhrf.jde.nrl.ar.**NRLARSampler**(*do_sampling=True*, *val_ini=None*, *contrasts={}*, *do_label_sampling=True*, *use_true_nrls=False*, *use_true_labels=False*, *labels_ini=None*, *ppm_proba_threshold=0.05*, *ppm_value_threshold=0*, *ppm_value_multi_threshold=array([0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2., 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3., 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.])*, *mean_activation_threshold=4*, *rescale_results=False*, *wip_variance_computation=False*)

Bases: *pyhrf.jde.nrl.bigaussian.NRLSampler*

Class handling the Gibbs sampling of Neural Response Levels according to:

> Makni, S., Ciuciu, P., Idier, J., & Poline, J. (2006). Joint Detection-Estimation of Brain Activity in fMRI using an Autoregressive Noise Model. In 3rd IEEE International Symposium on Biomedical Imaging: Macro to Nano, 2006. (pp. 1048–1051). IEEE. https://doi.org/10.1109/ISBI.2006.1625101

Inherits the abstract class C{ GibbsSamplerVariable}.

cleanMemory()

computeMeanVarClassApost(*j*, *variables*)

computeVarYTilde(*varXh*, *varMBYPl*)

linkToData(*dataInput*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)
#TODO : comment

pyhrf.jde.nrl.base module

pyhrf.jde.nrl.bigaussian module

class pyhrf.jde.nrl.bigaussian.**BiGaussMixtureParamsSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *hyper_prior_type='Jeffreys'*, *activ_thresh=4.0*, *var_ci_pr_alpha=2.04*, *var_ci_pr_beta=0.5*, *var_ca_pr_alpha=2.01*, *var_ca_pr_beta=0.5*, *mean_ca_pr_mean=5.0*, *mean_ca_pr_var=20.0*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

computeWithProperPriors(*j*, *cardCIj*, *cardCAj*)

finalizeSampling()

getCurrentMeans()

getCurrentVars()

getOutputs()

get_string_value(*v*)

linkToData(*dataInput*)

parametersComments = {'activ_thresh': 'Threshold for the max activ mean above which t

parametersToShow = []

sampleNextInternal(*variables*)

updateObsersables()

class pyhrf.jde.nrl.bigaussian.**BiGaussMixtureParamsSamplerWithRelVar**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *hyper_prior_type='Jeffreys'*, *activ_thresh=4.0*, *var_ci_pr_alpha=2.04*, *var_ci_pr_beta=0.5*, *var_ca_pr_alpha=2.01*, *var_ca_pr_beta=0.5*, *mean_ca_pr_mean=5.0*, *mean_ca_pr_var=20.0*)

Bases: *pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler*

computeWithProperPriorsWithRelVar(*nrlsj*, *j*, *cardCIj*, *cardCAj*, *wj*)

sampleNextInternal(*variables*)

class pyhrf.jde.nrl.bigaussian.**BiGaussMixtureParamsSamplerWithRelVar_OLD**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *hyper_prior_type='Jeffreys'*, *activ_thresh=4.0*, *var_ci_pr_alpha=2.04*, *var_ci_pr_beta=0.5*, *var_ca_pr_alpha=2.01*, *var_ca_pr_beta=0.5*, *mean_ca_pr_mean=5.0*, *mean_ca_pr_var=20.0*)

Bases: *pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler*

computeWithProperPriorsWithRelVar(*nrlsj*, *j*, *cardCIj*, *cardCAj*, *wj*)

sampleNextInternal(*variables*)

class pyhrf.jde.nrl.bigaussian.**MixtureWeightsSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

checkAndSetInitValue(*variables*)

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.nrl.bigaussian.**NRLSampler**(*do_sampling=True*, *val_ini=None*, *contrasts={}*, *do_label_sampling=True*, *use_true_nrls=False*, *use_true_labels=False*, *labels_ini=None*, *ppm_proba_threshold=0.05*, *ppm_value_threshold=0*, *ppm_value_multi_threshold=array([0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2., 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3., 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.])*, *mean_activation_threshold=4*, *rescale_results=False*, *wip_variance_computation=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Class handling the Gibbs sampling of Neural Response Levels with a prior bi-gaussian mixture model. It handles independent and spatial versions.

- Vincent, T., Risser, L., & Ciuciu, P. (2010). Spatially Adaptive Mixture Modeling for Analysis of fMRI Time Series. IEEE Transactions on Medical Imaging, 29(4), 1059–1074. https://doi.org/10.1109/TMI.2010.2042064
- Makni, S., Idier, J., Vincent, T., Thirion, B., Dehaene-Lambertz, G., & Ciuciu, P. (2008). A fully Bayesian approach to the parcel-based detection-estimation of brain activity in fMRI. NeuroImage, 41(3), 941–969. https://doi.org/10.1016/j.neuroimage.2008.02.017
- Sockel 2009 ICASSP (TODO: Complete reference)

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

FALSE_NEG = 3

FALSE_POS = 2

L_CA = 1

L_CI = 0

PPMcalculus(*threshold_value*, *apost_mean_activ*, *apost_var_activ*, *apost_mean_inactiv*, *apost_var_inactiv*, *labels_activ*, *labels_inactiv*)

Function to calculate the probability that the nrl in voxel j, condition m, is superior to a given hreshold_value

ThresholdPPM(*proba_voxel*, *threshold_pval*)

calcFracLambdaTilde(*cond*, *c1*, *c2*, *variables*)

checkAndSetInitLabels(*variables*)

checkAndSetInitNRL(*variables*)

checkAndSetInitValue(*variables*)

cleanMemory()

cleanObservables()

computeAA(*nrls*, *destaa*)

computeComponentsApost(*variables*, *j*, *gTQg*)

computeContrasts()

computeVarXhtQ(*h*, *varXQ*)

computeVarYTildeOpt(*varXh*)

compute_summary_stats()

countLabels(*labels*, *voxIdx*, *cardClass*)

finalizeSampling()

getClassifRate()

getFinalLabels(*thres=None*)

getOutputs()

getRocData(*dthres=0.005*)

get_final_summary()

initObservables()

init_contrasts()

linkToData(*dataInput*)

markWrongLabels(*labels*)

parametersComments = {'contrasts': 'Define contrasts as arithmetic expressions.\nCond

parametersToShow = ['contrasts']

printState(*_*)

reportDetection()

sampleLabels(*cond*, *variables*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

sampleNrlsParallel(*varXh*, *rb*, *h*, *varLambda*, *varCI*, *varCA*, *meanCA*, *gTQg*, *variables*)

sampleNrlsSerial(*rb*, *h*, *varCI*, *varCA*, *meanCA*, *gTQg*, *variables*)

samplingWarmUp(*variables*)
    #TODO : comment

saveCurrentValue(*it*)

saveObservables(*it*)

updateObsersables()

class pyhrf.jde.nrl.bigaussian.**NRLSamplerWithRelVar**(*do_sampling=True*, *val_ini=None*, *contrasts={}*, *do_label_sampling=True*, *use_true_nrls=False*, *use_true_labels=False*, *labels_ini=None*, *ppm_proba_threshold=0.05*, *ppm_value_threshold=0*, *ppm_value_multi_threshold=array([0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2., 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3., 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.])*, *mean_activation_threshold=4*, *rescale_results=False*, *wip_variance_computation=False*)

Bases: *pyhrf.jde.nrl.bigaussian.NRLSampler*

calcFracLambdaTildeWithIRRelCond(*cond*, *c1*, *c2*, *variables*, *nbVox*, *moyqvoxj*, *t1*, *t2*)

calcFracLambdaTildeWithRelCond(*l*, *nbVox*, *moyqvoxj*, *t1*, *t2*)

computeComponentsApostWithRelVar(*variables*, *j*, *gTQg*, *w*)

computeSumWAxh(*wa*, *varXh*)

computeVarYTildeOptWithRelVar(*varXh*, *w*)

computeWA(*a*, *w*, *wa*)

computemoyqvox(*cardClass*, *nbVox*)

Compute mean of labels in ROI (without the label of voxel i)

createWAxh(*aXh*, *w*)

deltaWCorr0(*nbVox*, *moyqvoxj*, *t1*, *t2*)

deltaWCorr1(*nbVox*, *moyqvoxj*, *t1*, *t2*)

sampleLabelsWithRelVar(*cond*, *variables*)

sampleNextInternal(*variables*)

sampleNrlsParallelWithRelVar(*varXh*, *rb*, *h*, *varLambda*, *varCI*, *varCA*, *meanCA*, *gTQg*, *variables*, *w*)

sampleNrlsSerialWithRelVar(*rb*, *h*, *gTQg*, *variables*, *w*, *t1*, *t2*)

samplingWarmUp(*variables*)

subtractYtildeWithRelVar()

class pyhrf.jde.nrl.bigaussian.**NRL_Multi_Sess_Sampler**(*parameters=None*, *xmlHandler=None*, *xmlLabel=None*, *xmlComment=None*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

P_OUTPUT_NRL = 'writeResponsesOutput'

P_SAMPLE_FLAG = 'sampleFlag'

P_TrueNrlFilename = 'TrueNrlFilename'

P_USE_TRUE_NRLS = 'useTrueNrls'

P_VAL_INI = 'initialValue'

checkAndSetInitValue(*variables*)

cleanMemory()

computeAA(*nrls*, *destaa*)

computeComponentsApost(*variables*, *m*, *varXh*, *s*)

computeVarYTildeSessionOpt(*varXh*, *s*)

defaultParameters = {'writeResponsesOutput': True, 'initialValue': None, 'TrueNrlFil

finalizeSampling()

getOutputs()

linkToData(*dataInput*)

parametersComments = {'TrueNrlFilename': 'Define the filename of simulated NRLs.\nIt

parametersToShow = ['writeResponsesOutput']

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)
#TODO : comment

saveCurrentValue(*it*)

class pyhrf.jde.nrl.bigaussian.**Variance_GaussianNRL_Multi_Sess**(*parameters=None*, *xmlHandler=None*, *xmlLabel=None*, *xmlComment=None*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

P_SAMPLE_FLAG = 'sampleFlag'

P_USE_TRUE_VALUE = 'useTrueValue'

P_VAL_INI = 'initialValue'

checkAndSetInitValue(*variables*)

defaultParameters = {'useTrueValue': False, 'initialValue': array([1.]), 'sampleFla

linkToData(*dataInput*)

parametersToShow = ['useTrueValue']

sampleNextInternal(*variables*)

pyhrf.jde.nrl.bigaussian_drift module

class pyhrf.jde.nrl.bigaussian_drift.**BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler**(*parameXmlHandler=NoXmlLabel=NoXmlComment=N*

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

P_ACTIV_THRESH = 'mean_activation_threshold'

P_HYPER_PRIOR = 'hyperPriorType'

P_MEAN_CA_PR_MEAN = 'meanCAPrMean'

P_MEAN_CA_PR_VAR = 'meanCAPrVar'

P_SAMPLE_FLAG = 'sampleFlag'

P_USE_TRUE_VALUE = 'useTrueValue'

P_VAL_INI = 'initialValue'

P_VAR_CA_PR_ALPHA = 'varCAPrAlpha'

P_VAR_CA_PR_BETA = 'varCAPrBeta'

P_VAR_CI_PR_ALPHA = 'varCIPrAlpha'

P_VAR_CI_PR_BETA = 'varCIPrBeta'

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

computeWithProperPriors(*j*, *cardCIj*, *cardCAj*)

defaultParameters = {'initialValue': None, 'mean_activation_threshold': 4.0, 'varCAP

finalizeSampling()

getCurrentMeans()

getCurrentVars()

getOutputs()

get_string_value(*v*)

linkToData(*dataInput*)

parametersComments = {'hyperPriorType': "Either 'proper' or 'Jeffrey'", 'mean_activat

parametersToShow = ['initialValue', 'sampleFlag', 'mean_activation_threshold', 'useTru

sampleNextInternal(*variables*)

updateObsersables()

class pyhrf.jde.nrl.bigaussian_drift.**NRL_Drift_Sampler**(*do_sampling=True*, *val_ini=None*, *contrasts={}*, *do_label_sampling=True*, *use_true_nrls=False*, *use_true_labels=False*, *labels_ini=None*, *ppm_proba_threshold=0.05*, *ppm_value_threshold=0*, *ppm_value_multi_threshold=array([0.*, *0.1*, *0.2*, *0.3*, *0.4*, *0.5*, *0.6*, *0.7*, *0.8*, *0.9*, *1.*, *1.1*, *1.2*, *1.3*, *1.4*, *1.5*, *1.6*, *1.7*, *1.8*, *1.9*, *2.*, *2.1*, *2.2*, *2.3*, *2.4*, *2.5*, *2.6*, *2.7*, *2.8*, *2.9*, *3.*, *3.1*, *3.2*, *3.3*, *3.4*, *3.5*, *3.6*, *3.7*, *3.8*, *3.9*, *4.])*, *mean_activation_threshold=4*, *rescale_results=False*, *wip_variance_computation=False*)

Bases: *pyhrf.jde.nrl.bigaussian.NRLSampler*

Class handling the Gibbs sampling of Neural Response Levels in the case of joint drift sampling.

computeVarYTildeOpt(*varXh*)

sampleNextInternal(*variables*)

sampleNrlsSerial(*rb*, *h*, *varCI*, *varCA*, *meanCA*, *gTg*, *variables*)

class pyhrf.jde.nrl.bigaussian_drift.**NRL_Drift_SamplerWithRelVar**(*do_sampling=True*, *val_ini=None*, *contrasts={}*, *do_label_sampling=True*, *use_true_nrls=False*, *use_true_labels=False*, *labels_ini=None*, *ppm_proba_threshold=0.05*, *ppm_value_threshold=0*, *ppm_value_multi_threshold=array([0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2., 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3., 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.])*, *mean_activation_threshold=4*, *rescale_results=False*, *wip_variance_computation=False*)

Bases: *pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar*

Class handling the Gibbs sampling of Neural Response Levels in the case of joint drift sampling and relevant variable.

computeVarYTildeOptWithRelVar(*varXh*, *w*)

sampleNextInternal(*variables*)

sampleNrlsSerialWithRelVar(*rb*, *h*, *gTg*, *variables*, *w*, *t1*, *t2*)

class pyhrf.jde.nrl.bigaussian_drift.**NRLsBar_Drift_Multi_Sess_Sampler**(*do_sampling=True*, *val_ini=None*, *contrasts={}*, *do_label_sampling=True*, *use_true_nrls=False*, *use_true_labels=False*, *labels_ini=None*, *ppm_proba_threshold=0.05*, *ppm_value_threshold=0*, *ppm_value_multi_threshold=array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.])*, *mean_activation_threshold=4*, *rescale_results=False*, *wip_variance_computation=F*

Bases: *pyhrf.jde.nrl.bigaussian.NRLSampler*

Class handling the Gibbs sampling of Neural Response Levels in the case of joint drift sampling.

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

sampleNrlsSerial(*varCI*, *varCA*, *meanCA*, *variables*)

samplingWarmUp(*variables*)
#TODO : comment

pyhrf.jde.nrl.bigaussian_drift.**permutation**(*x*)
Randomly permute a sequence, or return a permuted range.

If *x* is a multi-dimensional array, it is only shuffled along its first index.

Parameters **x** (*int or array_like*) – If *x* is an integer, randomly permute `np.arange(x)`. If *x* is an array, make a copy and shuffle the elements randomly.

Returns **out** – Permuted sequence or array range.

Return type *ndarray*

Examples

```
>>> np.random.permutation(10)
array([1, 7, 4, 3, 0, 9, 2, 5, 8, 6])
```

```
>>> np.random.permutation([1, 4, 9, 12, 15])
array([15,  1,  9,  4, 12])
```

```
>>> arr = np.arange(9).reshape((3, 3))
>>> np.random.permutation(arr)
array([[6, 7, 8],
       [0, 1, 2],
       [3, 4, 5]])
```

pyhrf.jde.nrl.bigaussian_drift.**rand**(*d0*, *d1*, ..., *dn*)
Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.

Returns **out** – Random values.

Return type ndarray, shape `(d0, d1, ..., dn)`

See also:

`random()`

Notes

This is a convenience function. If you want an interface that takes a shape-tuple as the first argument, refer to np.random.random_sample .

Examples

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

pyhrf.jde.nrl.bigaussian_drift.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

`sigma * np.random.randn(...) + mu`

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.jde.nrl.gammagaussian module

class pyhrf.jde.nrl.gammagaussian.**GamGaussMixtureParamsSampler**(*parameters=None*, *xmlHandler=None*, *xmlLabel=None*, *xmlComment=None*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

NB_PARAMS = 3

PARAMS_NAMES = ['Shape_Activ', 'Scale_Activ', 'Var_Inactiv']

P_SAMPLE_FLAG = 'sampleFlag'

P_SCALE_CA_PR_ALPHA = 'scaleCAPrAlpha'

P_SCALE_CA_PR_BETA = 'scaleCAPrBeta'

P_SHAPE_CA_PR_MEAN = 'shapeCAPrMean'

P_VAL_INI = 'initialValue'

P_VAR_CI_PR_ALPHA = 'varCIPrAlpha'

P_VAR_CI_PR_BETA = 'varCIPrBeta'

checkAndSetInitValue(*variables*)

defaultParameters = {'initialValue': None, 'varCIPrBeta': 0.5, 'sampleFlag': 1, 'sc

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.nrl.gammagaussian.**InhomogeneousNRLSampler**(*parameters=None*, *xmlHandler=None*, *xmlLabel=None*, *xmlComment=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Class handling the Gibbs sampling of Neural Response Levels according to:

- Makni, S., Ciuciu, P., Idier, J., & Poline, J.-B. (2005). Joint detection-estimation of brain activity in functional MRI: a Multichannel Deconvolution solution. IEEE Transactions on Signal Processing, 53(9), 3488–3502. https://doi.org/10.1109/TSP.2005.853303

Inherits the abstract class C{GibbsSamplerVariable}. #TODO : comment attributes

L_CA = 1

L_CI = 0

P_BETA = 'beta'

P_LABELS_COLORS = 'labelsColors'

P_LABELS_INI = 'labelsIni'

P_SAMPLE_FLAG = 'sampleFlag'

P_SAMPLE_LABELS = 'sampleLabels'

P_TRUE_LABELS = 'trueLabels'

P_VAL_INI = 'initialValue'

calcEnergy(*voxIdx*, *label*, *cond*)

checkAndSetInitValue(*variables*)

computeMean()

computeMeanClassApost(*j*, *nrls*, *varXhj*, *rb*)

computeVarYTilde(*varXh*)

computeVariablesApost(*varCI*, *shapeCA*, *scaleCA*, *rb*, *varXh*, *varLambda*)

countLabels()

defaultParameters = {'initialValue': None, 'sampleLabels': 1, 'labelsColors': array

finalizeSampling()

linkToData(*dataInput*)

sampleLabels(*cond*, *varCI*, *varCA*, *meanCA*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)
#TODO : comment

pyhrf.jde.nrl.habituation module

pyhrf.jde.nrl.habituation.**LaplacianPdf**(*beta*, *r0Hab*, *a*, *b*, *N=1*)

class pyhrf.jde.nrl.habituation.**NRLwithHabSampler**
Bases: *pyhrf.jde.nrl.bigaussian.NRLSampler*

Class handling the Gibbs sampling of Neural Response Levels in combination with habituation speed factor sampling. The underlying model is exponential decaying #TODO : comment attributes

P_HABITS_INI = 'habitIni'

P_HAB_ALGO_PARAM = 'paramLexp'

P_OUTPUT_RATIO = 'outputRatio'

P_SAMPLE_HABITS = 'sampleHabit'

P_TRUE_HABITS = 'trueHabits'

checkAndSetInitHabit(*variables*)

checkAndSetInitValue(*variables*)

cleanMemory()

cleanObservables()

computeComponentsApost(*variables*, *j*, *XhtQXh*)

computeVarXhtQ(*Q*)

computeVarYTildeHab(*varXh*)

computeVarYTildeHabOld(*varXh*)

finalizeSampling()

getOutputs()

habitCondSampler(*j*, *rb*, *varHRF*)

habitCondSamplerParallel(*rb*, *h*)

habitCondSamplerSerial(*rb*, *h*)

initObservables()

linkToData(*dataInput*)

parametersComments = {'contrasts': 'Define contrasts as arithmetic expressions.\nCond

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

sampleNrlsParallel(*rb*, *h*, *varLambda*, *varCI*, *varCA*, *meanCA*, *varXhtQXh*, *variables*)

sampleNrlsSerial(*varXh*, *rb*, *h*, *varCI*, *varCA*, *meanCA*, *variables*)

sampleNrlsSerial_bak(*rb*, *h*, *varLambda*, *varCI*, *varCA*, *meanCA*, *varXhtQXh*, *variables*)

samplingWarmUp(*variables*)

saveCurrentValue()

setupGamma()

setupTimeNrls()

spExtract(*spInd*, *mtrx*, *cond*)

updateGammaTimeNRLs(*nc*, *nv*)

updateObsersables()

updateXh(*varHRF*)

updateYtilde()

pyhrf.jde.nrl.habituation.**sparsedot**(*X*, *A*, *mask*, *taille*)

pyhrf.jde.nrl.habituation.**sparsedotdimun**(*X*, *A*, *mask*, *lenght*)

pyhrf.jde.nrl.habituation.**subcptGamma**(*nrl*, *habit*, *nbTrials*, *deltaOns*)

pyhrf.jde.nrl.trigaussian module

class pyhrf.jde.nrl.trigaussian.**GGGNRLSampler**(*do_sampling=True*, *val_ini=None*, *contrasts={}*, *do_label_sampling=True*, *use_true_nrls=False*, *use_true_labels=False*, *labels_ini=None*, *ppm_proba_threshold=0.05*, *ppm_value_threshold=0*, *ppm_value_multi_threshold=array([0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2., 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3., 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.])*, *mean_activation_threshold=4*, *rescale_results=False*, *wip_variance_computation=False*)

Bases: *pyhrf.jde.nrl.bigaussian.NRLSampler*

CLASSES = array([0, 1, 2])

CLASS_NAMES = ['inactiv', 'activ', 'deactiv']

FALSE_NEG = 4

FALSE_POS = 3

L_CA = 1

L_CD = 2

L_CI = 0

sampleLabels(*cond*, *variables*)

class pyhrf.jde.nrl.trigaussian.**TriGaussMixtureParamsSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *hyper_prior_type='Jeffreys'*, *activ_thresh=4.0*, *var_ci_pr_alpha=2.04*, *var_ci_pr_beta=0.5*, *var_ca_pr_alpha=2.01*, *var_ca_pr_beta=0.5*, *var_cd_pr_alpha=2.01*, *var_cd_pr_beta=0.5*, *mean_ca_pr_mean=5.0*, *mean_ca_pr_var=20.0*, *mean_cd_pr_mean=-20.0*, *mean_cd_pr_var=20.0*)

Bases: *pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler*

I_MEAN_CD = 3

I_VAR_CD = 4

L_CD = 2

NB_PARAMS = 5

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv', 'Mean_Deactiv', 'Var_Deactiv

P_MEAN_CD_PR_MEAN = 'meanCDPrMean'

P_MEAN_CD_PR_VAR = 'meanCDPrVar'

P_VAR_CD_PR_ALPHA = 'varCDPrAlpha'

P_VAR_CD_PR_BETA = 'varCDPrBeta'

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCDj*)

finalizeSampling()

getCurrentMeans()

getCurrentVars()

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

1.2.2 Submodules

pyhrf.jde.asl module

class pyhrf.jde.asl.**ASLSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *bold_response_levels=<pyhrf.jde.asl.BOLDResponseLevelSampler object>*, *perf_response_levels=<pyhrf.jde.asl.PerfResponseLevelSampler object>*, *labels=<pyhrf.jde.asl.LabelSampler object>*, *noise_var=<pyhrf.jde.asl.NoiseVarianceSampler object>*, *brf=<pyhrf.jde.asl.BOLDResponseSampler object>*, *brf_var=<pyhrf.jde.asl.BOLDResponseVarianceSampler object>*, *prf=<pyhrf.jde.asl.PerfResponseSampler object>*, *prf_var=<pyhrf.jde.asl.PerfResponseVarianceSampler object>*, *bold_mixt_params=<pyhrf.jde.asl.BOLDMixtureSampler object>*, *perf_mixt_params=<pyhrf.jde.asl.PerfMixtureSampler object>*, *drift=<pyhrf.jde.asl.DriftCoeffSampler object>*, *drift_var=<pyhrf.jde.asl.DriftVarianceSampler object>*, *perf_baseline=<pyhrf.jde.asl.PerfBaselineSampler object>*, *perf_baseline_var=<pyhrf.jde.asl.PerfBaselineVarianceSampler object>*, *check_final_value=None*, *output_fit=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

computeFit()

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

inputClass
alias of *WN_BiG_ASLSamplerInput*

parametersToShow = ['nb_its', 'bold_response_levels', 'brf', 'brf_var', 'prf', 'prf_va

class pyhrf.jde.asl.**BOLDMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl.**BOLDResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

computeVarYTildeOpt(*update_perf=False*)

if update_perf is True then also update sumcXg and prl.ytilde update_perf should only be used at init of variable values.

getOutputs()

samplingWarmUp(*v*)

class pyhrf.jde.asl.**BOLDResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()

y - sum cWXg - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

class pyhrf.jde.asl.**BOLDResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

class pyhrf.jde.asl.**DriftCoeffSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

get_final_value()

get_true_value()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

updateNorm()

class pyhrf.jde.asl.**DriftVarianceSampler**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

get_MAP_labels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.asl.**MixtureParamsSampler**(*name*, *response_level_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

get_current_means()

get_current_vars()

get_true_values_from_simulation_dict()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl.**NoiseVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl.**PerfBaselineSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_residuals()

compute_wa(*a=None*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl.**PerfBaselineVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl.**PerfMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl.**PerfResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

computeVarYTildeOpt()

class pyhrf.jde.asl.**PerfResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *diff_res=True*)

Bases: *pyhrf.jde.asl.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()

y - sum aXh - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

class pyhrf.jde.asl.**PerfResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

class pyhrf.jde.asl.**ResponseLevelSampler**(*name*, *response_name*, *mixture_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

computeRR()

computeVarYTildeOpt()

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

updateObsersables()

class pyhrf.jde.asl.**ResponseSampler**(*name*, *response_level_name*, *variance_name*, *smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Generic parent class to perfusion response & BOLD response samplers

calcXResp(*resp*, *stackX=None*)

checkAndSetInitValue(*variables*)

computeYTilde()

get_mat_X()

get_mat_XtX()

get_rlrl()

get_stackX()

get_ybar()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

setFinalValue()

updateNorm()

updateXResp()

class pyhrf.jde.asl.**ResponseVarianceSampler**(*name*, *response_name*, *val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl.**WN_BiG_ASLSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

pyhrf.jde.asl.**b**()

pyhrf.jde.asl.**compute_StS_StY**(*rls*, *v_b*, *mx*, *mxtx*, *ybar*, *rlrl*, *yaj*, *ajak_vb*)

yaj and ajak_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.jde.asl.**simulate_asl**(*output_dir=None*, *noise_scenario='high_snr'*, *spatial_size='tiny'*, *v_noise=None*, *dt=0.5*, *tr=2.5*)

pyhrf.jde.asl_2steps module

pyhrf.jde.asl_2steps.**dummy_jde**(*fmri_data*, *dt*)

pyhrf.jde.asl_2steps.**jde_analyse_2steps_v1**(*output_dir*, *fmri_data*, *dt*, *nb_iterations*, *brf_var=None*, *do_sampling_brf_var=False*, *prf_var=None*, *do_sampling_prf_var=False*)

#Return: # dict of outputs

pyhrf.jde.asl_2steps.**physio_build_jde_mcmc_sampler**(*nb_iterations*, *rf_prior*, *flag_zc=False*, *brf_var_ini=None*, *prf_var_ini=None*, *do_sampling_brf_var=False*, *do_sampling_prf_var=False*, *prf_ini=None*, *do_sampling_prf=True*, *prls_ini=None*, *do_sampling_prls=True*, *brf_ini=None*, *do_sampling_brf=True*, *brls_ini=None*, *do_sampling_brls=True*, *perf_bl_ini=None*, *do_sampling_perf_bl=True*, *do_sampling_perf_var=True*, *drift_ini=None*, *do_sampling_drift=True*, *drift_var_ini=None*, *do_sampling_drift_var=True*, *noise_var_ini=None*, *labels_ini=None*, *do_sampling_labels=True*)

pyhrf.jde.asl_physio module

class pyhrf.jde.asl_physio.**ASLPhysioSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *bold_response_levels=<pyhrf.jde.asl_physio.BOLDRespons object>*, *perf_response_levels=<pyhrf.jde.asl_physio.PerfResponseLe object>*, *labels=<pyhrf.jde.asl_physio.LabelSampler object>*, *noise_var=<pyhrf.jde.asl_physio.NoiseVarianceSampler object>*, *brf=<pyhrf.jde.asl_physio.PhysioBOLDResponseSampler object>*, *brf_var=<pyhrf.jde.asl_physio.PhysioBOLDResponseVarian object>*, *prf=<pyhrf.jde.asl_physio.PhysioPerfResponseSampler object>*, *prf_var=<pyhrf.jde.asl_physio.PhysioPerfResponseVariance object>*, *bold_mixt_params=<pyhrf.jde.asl_physio.BOLDMixtureSa object>*, *perf_mixt_params=<pyhrf.jde.asl_physio.PerfMixtureSampl object>*, *drift=<pyhrf.jde.asl_physio.DriftCoeffSampler object>*, *drift_var=<pyhrf.jde.asl_physio.DriftVarianceSampler object>*, *perf_baseline=<pyhrf.jde.asl_physio.PerfBaselineSampler object>*, *perf_baseline_var=<pyhrf.jde.asl_physio.PerfBaselineVaria object>*, *check_final_value=None*, *output_fit=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

computeFit()

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

inputClass
alias of *WN_BiG_ASLSamplerInput*

parametersToShow = ['nb_its', 'response_levels', 'hrf', 'hrf_var']

class pyhrf.jde.asl_physio.**BOLDMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.asl_physio.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio.**BOLDResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.asl_physio.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

computeVarYTildeOpt(*update_perf=False*)
if update_perf is True then also update sumcXg and prl.ytilde update_perf should only be used at init of variable values.

getOutputs()

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio.**DriftCoeffSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*v*)

updateNorm()

class pyhrf.jde.asl_physio.**DriftVarianceSampler**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio.**MixtureParamsSampler**(*name*, *response_level_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

get_current_means()

get_current_vars()

get_true_values_from_simulation_dict()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio.**NoiseVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio.**PerfBaselineSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_residuals()

compute_wa(*a=None*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio.**PerfBaselineVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio.**PerfMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio.**PerfResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

computeVarYTildeOpt()

class pyhrf.jde.asl_physio.**PhysioBOLDResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum cWXg - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample BRF

changes to mean: changes to var:

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio.**PhysioBOLDResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

class pyhrf.jde.asl_physio.**PhysioPerfResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *diff_res=True*, *prior_type='physio_stochastic_regularized'*)

Bases: *pyhrf.jde.asl_physio.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum aXh - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample PRF with physio prior

changes to mean: add a factor of Omega h Sigma_g^-1 v_g^-1

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio.**PhysioPerfResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

class pyhrf.jde.asl_physio.**ResponseLevelSampler**(*name*, *response_name*, *mixture_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

computeRR()

computeVarYTildeOpt()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

class pyhrf.jde.asl_physio.**ResponseSampler**(*name*, *response_level_name*, *variance_name*, *smooth_order=2*, *zero_constraint=False*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Generic parent class to perfusion response & BOLD response samplers

calcXResp(*resp*, *stackX=None*)

checkAndSetInitValue(*variables*)

computeYTilde()

getOutputs()

get_mat_X()

get_mat_XtX()

get_rlrl()

get_stackX()

get_ybar()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

setFinalValue()

updateNorm()

updateXResp()

class pyhrf.jde.asl_physio.**ResponseVarianceSampler**(*name*, *response_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

Sample variance of BRF or PRF

TODO: change code below –> no changes necessary so far

class pyhrf.jde.asl_physio.**WN_BiG_ASLSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

pyhrf.jde.asl_physio.**b**()

pyhrf.jde.asl_physio.**compute_StS_StY**(*rls*, *v_b*, *mx*, *mxtx*, *ybar*, *rlrl*, *yaj*, *ajak_vb*)

yaj and ajak_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio.**compute_StS_StY_deterministic**(*brls*, *prls*, *v_b*, *mx*, *mxtx*, *mwx*, *mxtwx*, *mwxtwx*, *ybar*, *rlrl_bold*, *rlrl_perf*, *brlprl*, *omega*, *yj*, *ajak_vb*)

yj, ajak_vb and cjck_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio.**compute_bRpR**(*brl*, *prl*, *nbConditions*, *nbVoxels*)

pyhrf.jde.asl_physio_1step module

class pyhrf.jde.asl_physio_1step.**ASLPhysioSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *bold_response_levels=<pyhrf.jde.asl_physio_1step.BOLDR object>*, *perf_response_levels=<pyhrf.jde.asl_physio_1step.PerfResp object>*, *labels=<pyhrf.jde.asl_physio_1step.LabelSampler object>*, *noise_var=<pyhrf.jde.asl_physio_1step.NoiseVarianceSamp object>*, *brf=<pyhrf.jde.asl_physio_1step.PhysioBOLDResponseSam object>*, *brf_var=<pyhrf.jde.asl_physio_1step.PhysioBOLDResponse object>*, *prf=<pyhrf.jde.asl_physio_1step.PhysioPerfResponseSampl object>*, *prf_var=<pyhrf.jde.asl_physio_1step.PhysioPerfResponseV object>*, *bold_mixt_params=<pyhrf.jde.asl_physio_1step.BOLDMixt object>*, *perf_mixt_params=<pyhrf.jde.asl_physio_1step.PerfMixture object>*, *drift=<pyhrf.jde.asl_physio_1step.DriftCoeffSampler object>*, *drift_var=<pyhrf.jde.asl_physio_1step.DriftVarianceSample object>*, *perf_baseline=<pyhrf.jde.asl_physio_1step.PerfBaselineSam object>*, *perf_baseline_var=<pyhrf.jde.asl_physio_1step.PerfBaselin object>*, *check_final_value=None*, *output_fit=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

computeFit()

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

inputClass
alias of *WN_BiG_ASLSamplerInput*

parametersToShow = ['nb_its', 'response_levels', 'hrf', 'hrf_var']

class pyhrf.jde.asl_physio_1step.**BOLDMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_1step.**BOLDResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

computeVarYTildeOpt(*update_perf=False*)

if update_perf is True then also update sumcXg and prl.ytilde update_perf should only be used at init of variable values.

getOutputs()

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_1step.**DriftCoeffSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*v*)

updateNorm()

class pyhrf.jde.asl_physio_1step.**DriftVarianceSampler**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_1step.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_1step.**MixtureParamsSampler**(*name*, *response_level_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

get_current_means()

get_current_vars()

get_true_values_from_simulation_dict()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_1step.**NoiseVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_1step.**PerfBaselineSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_residuals()

compute_wa(*a=None*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_1step.**PerfBaselineVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_1step.**PerfMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_1step.**PerfResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

computeVarYTildeOpt()

class pyhrf.jde.asl_physio_1step.**PhysioBOLDResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum cWXg - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample BRF

changes to mean: changes to var:

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_1step.**PhysioBOLDResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

class pyhrf.jde.asl_physio_1step.**PhysioPerfResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *diff_res=True*, *prior_type='physio_stochastic_regularized'*)

Bases: *pyhrf.jde.asl_physio_1step.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum aXh - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample PRF with physio prior

changes to mean: add a factor of Omega h Sigma_g^-1 v_g^-1

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_1step.**PhysioPerfResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.asl_physio_1step.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

class pyhrf.jde.asl_physio_1step.**ResponseLevelSampler**(*name*, *response_name*, *mixture_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

computeRR()

computeVarYTildeOpt()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

class pyhrf.jde.asl_physio_1step.**ResponseSampler**(*name*, *response_level_name*, *variance_name*, *smooth_order=2*, *zero_constraint=False*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Generic parent class to perfusion response & BOLD response samplers

calcXResp(*resp*, *stackX=None*)

checkAndSetInitValue(*variables*)

computeYTilde()

getOutputs()

get_mat_X()

get_mat_XtX()

get_rlrl()

get_stackX()

get_ybar()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

setFinalValue()

updateNorm()

updateXResp()

class pyhrf.jde.asl_physio_1step.**ResponseVarianceSampler**(*name*, *response_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

Sample variance of BRF or PRF

TODO: change code below –> no changes necessary so far

class pyhrf.jde.asl_physio_1step.**WN_BiG_ASLSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

pyhrf.jde.asl_physio_1step.**b**()

pyhrf.jde.asl_physio_1step.**compute_StS_StY**(*rls*, *v_b*, *mx*, *mxtx*, *ybar*, *rlrl*, *yaj*, *ajak_vb*)

yaj and ajak_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_1step.**compute_StS_StY_deterministic**(*brls*, *prls*, *v_b*, *mx*, *mxtx*, *mwx*, *mxtwx*, *mwxtwx*, *ybar*, *rlrl_bold*, *rlrl_perf*, *brlprl*, *omega*, *yj*, *ajak_vb*)

yj, ajak_vb and cjck_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_1step.**compute_bRpR**(*brl*, *prl*, *nbConditions*, *nbVoxels*)

pyhrf.jde.asl_physio_1step_params module

class pyhrf.jde.asl_physio_1step_params.**ASLPhysioSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallback object>*, *bold_response_levels=<pyhrf.jde.asl_physio_1ste object>*, *perf_response_levels=<pyhrf.jde.asl_physio_1ste object>*, *labels=<pyhrf.jde.asl_physio_1step_params.LabelS object>*, *noise_var=<pyhrf.jde.asl_physio_1step_params.N object>*, *brf=<pyhrf.jde.asl_physio_1step_params.PhysioE object>*, *brf_var=<pyhrf.jde.asl_physio_1step_params.Phy object>*, *prf=<pyhrf.jde.asl_physio_1step_params.PhysioE object>*, *prf_var=<pyhrf.jde.asl_physio_1step_params.Phy object>*, *bold_mixt_params=<pyhrf.jde.asl_physio_1step_ object>*, *perf_mixt_params=<pyhrf.jde.asl_physio_1step_p object>*, *drift=<pyhrf.jde.asl_physio_1step_params.DriftC object>*, *drift_var=<pyhrf.jde.asl_physio_1step_params.D object>*, *perf_baseline=<pyhrf.jde.asl_physio_1step_para object>*, *perf_baseline_var=<pyhrf.jde.asl_physio_1step_p object>*, *check_final_value=None*, *output_fit=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

computeFit()

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

inputClass
alias of *WN_BiG_ASLSamplerInput*

parametersToShow = ['nb_its', 'response_levels', 'hrf', 'hrf_var']

class pyhrf.jde.asl_physio_1step_params.**BOLDMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_1step_params.**BOLDResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

computeVarYTildeOpt(*update_perf=False*)

if update_perf is True then also update sumcXg and prl.ytilde update_perf should only be used at init of variable values.

getOutputs()

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_1step_params.**DriftCoeffSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*v*)

updateNorm()

class pyhrf.jde.asl_physio_1step_params.**DriftVarianceSampler**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_1step_params.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_1step_params.**MixtureParamsSampler**(*name*, *response_level_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

get_current_means()

get_current_vars()

get_true_values_from_simulation_dict()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_1step_params.**NoiseVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_1step_params.**PerfBaselineSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_residuals()

compute_wa(*a=None*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_1step_params.**PerfBaselineVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_1step_params.**PerfMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_1step_params.**PerfResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

computeVarYTildeOpt()

class pyhrf.jde.asl_physio_1step_params.**PhysioBOLDResponseSampler**(*phy_params={'E0': 0.8, 'tau_m': 1.0, 'r0': 100, 'linear': True, 'vt0': 80.6, 'tau_f': 2.5, 'eps': 0.5, 'tau_s': 1.25, 'V0': 0.02, 'alpha_w': 0.2, 'model': 'RBM', 'obata': False, 'buxton': False, 'e': 0.4, 'eps_max': 10.0, 'TE': 0.04, 'model_name': 'Friston00'}*, *smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step_params.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum cWXg - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample BRF

changes to mean: changes to var:

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_1step_params.**PhysioBOLDResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step_params.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

class pyhrf.jde.asl_physio_1step_params.**PhysioPerfResponseSampler**(*phy_params={'E0': 0.8, 'tau_m': 1.0, 'r0': 100, 'linear': True, 'vt0': 80.6, 'tau_f': 2.5, 'eps': 0.5, 'tau_s': 1.25, 'V0': 0.02, 'alpha_w': 0.2, 'model': 'RBM', 'obata': False, 'buxton': False, 'e': 0.4, 'eps_max': 10.0, 'TE': 0.04, 'model_name': 'Friston00'}*, *smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *diff_res=True*, *prior_type='physio_stochastic_regu*

Bases: *pyhrf.jde.asl_physio_1step_params.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum aXh - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample PRF with physio prior

changes to mean: add a factor of Omega h Sigma_g^-1 v_g^-1

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_1step_params.**PhysioPerfResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_1step_params.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

class pyhrf.jde.asl_physio_1step_params.**ResponseLevelSampler**(*name*, *response_name*, *mixture_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

computeRR()

computeVarYTildeOpt()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

class pyhrf.jde.asl_physio_1step_params.**ResponseSampler**(*name*, *response_level_name*, *variance_name*, *phy_params*, *smooth_order=2*, *zero_constraint=False*, *duration=25.0*, *normalise=0.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Generic parent class to perfusion response & BOLD response samplers

calcXResp(*resp*, *stackX=None*)

checkAndSetInitValue(*variables*)

computeYTilde()

getOutputs()

get_mat_X()

get_mat_XtX()

get_rlrl()

get_stackX()

get_ybar()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

setFinalValue()

updateNorm()

updateXResp()

class pyhrf.jde.asl_physio_1step_params.**ResponseVarianceSampler**(*name*, *response_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

Sample variance of BRF or PRF

TODO: change code below –> no changes necessary so far

class pyhrf.jde.asl_physio_1step_params.**WN_BiG_ASLSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

pyhrf.jde.asl_physio_1step_params.**b**()

pyhrf.jde.asl_physio_1step_params.**compute_StS_StY**(*rls*, *v_b*, *mx*, *mxtx*, *ybar*, *rlrl*, *yaj*, *ajak_vb*)

yaj and ajak_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_1step_params.**compute_StS_StY_deterministic**(*brls*, *prls*, *v_b*, *mx*, *mxtx*, *mwx*, *mxtwx*, *mwxtwx*, *ybar*, *rlrl_bold*, *rlrl_perf*, *brlprl*, *omega*, *yj*, *ajak_vb*)

yj, ajak_vb and cjck_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_1step_params.**compute_bRpR**(*brl*, *prl*, *nbConditions*, *nbVoxels*)

pyhrf.jde.asl_physio_det_fwdm module

Physio prior, deterministic version where fwd model is changed TODO: clean to remove stochastic parts

class pyhrf.jde.asl_physio_det_fwdm.**ASLPhysioSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHand object>*, *bold_response_levels=<pyhrf.jde.asl_physio_det_fwdm object>*, *perf_response_levels=<pyhrf.jde.asl_physio_det_fwdm. object>*, *labels=<pyhrf.jde.asl_physio_det_fwdm.LabelSampler object>*, *noise_var=<pyhrf.jde.asl_physio_det_fwdm.NoiseVaria object>*, *brf=<pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResp object>*, *brf_var=<pyhrf.jde.asl_physio_det_fwdm.PhysioBOLD object>*, *prf=<pyhrf.jde.asl_physio_det_fwdm.PhysioPerfRespon object>*, *prf_var=<pyhrf.jde.asl_physio_det_fwdm.PhysioPerfRe object>*, *bold_mixt_params=<pyhrf.jde.asl_physio_det_fwdm.B object>*, *perf_mixt_params=<pyhrf.jde.asl_physio_det_fwdm.Pe object>*, *drift=<pyhrf.jde.asl_physio_det_fwdm.DriftCoeffSampl object>*, *drift_var=<pyhrf.jde.asl_physio_det_fwdm.DriftVarian object>*, *perf_baseline=<pyhrf.jde.asl_physio_det_fwdm.PerfBa object>*, *perf_baseline_var=<pyhrf.jde.asl_physio_det_fwdm.Pe object>*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

computeFit()

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

inputClass
alias of *WN_BiG_ASLSamplerInput*

parametersToShow = ['nb_its', 'response_levels', 'hrf', 'hrf_var']

class pyhrf.jde.asl_physio_det_fwdm.**BOLDMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_det_fwdm.**BOLDResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_det_fwdm.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

computeVarYTildeOpt(*update_perf=False*)

if update_perf is True then also update sumcXg and prl.ytilde update_perf should only be used at init of variable values.

getOutputs()

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_det_fwdm.**DriftCoeffSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

updateNorm()

class pyhrf.jde.asl_physio_det_fwdm.**DriftVarianceSampler**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_det_fwdm.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_det_fwdm.**MixtureParamsSampler**(*name*, *response_level_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

get_current_means()

get_current_vars()

get_true_values_from_simulation_dict()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_det_fwdm.**NoiseVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_det_fwdm.**PerfBaselineSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_residuals()

compute_wa(*a=None*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_det_fwdm.**PerfBaselineVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_det_fwdm.**PerfMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_det_fwdm.**PerfResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_det_fwdm.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

computeVarYTildeOpt()

class pyhrf.jde.asl_physio_det_fwdm.**PhysioBOLDResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *use_omega=True*, *deterministic=False*)

Bases: *pyhrf.jde.asl_physio_det_fwdm.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()

y - sum cWXg - Pl - wa

get_mat_X()

get_mat_XtWX()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)

Sample BRF

changes to mean: changes to var:

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_det_fwdm.**PhysioBOLDResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_det_fwdm.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

sampleNextInternal(*v*)
Sample variance of BRF

TODO: change code below –> no changes necessary so far

class pyhrf.jde.asl_physio_det_fwdm.**PhysioPerfResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *diff_res=True*, *regularize=True*, *deterministic=False*)

Bases: *pyhrf.jde.asl_physio_det_fwdm.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum aXh - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample PRF with physio prior

changes to mean: add a factor of Omega h Sigma_g^-1 v_g^-1

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_det_fwdm.**PhysioPerfResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_det_fwdm.ResponseVarianceSampler*, *pyhrf.xmlio.Initable*

sampleNextInternal(*v*)
Sample variance of PRF

changes:

- mu_g = omega h
- new beta calculation, based on physio_inspired prior

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_det_fwdm.**ResponseLevelSampler**(*name*, *response_name*, *mixture_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

computeRR()

computeVarYTildeOpt()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_det_fwdm.**ResponseSampler**(*name*, *response_level_name*, *variance_name*, *smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *deterministic=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Generic parent class to perfusion response & BOLD response samplers

calcXResp(*resp*, *stackX=None*)

checkAndSetInitValue(*variables*)

computeYTilde()

get_mat_X()

get_mat_XtX()

get_rlrl()

get_stackX()

get_ybar()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

setFinalValue()

updateNorm()

updateXResp()

class pyhrf.jde.asl_physio_det_fwdm.**ResponseVarianceSampler**(*name*, *response_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_det_fwdm.**WN_BiG_ASLSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

pyhrf.jde.asl_physio_det_fwdm.**b**()

pyhrf.jde.asl_physio_det_fwdm.**compute_StS_StY**(*rls*, *v_b*, *mx*, *mxtx*, *ybar*, *rlrl*, *yaj*, *ajak_vb*)

yaj and ajak_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_det_fwdm.**compute_StS_StY_deterministic**(*brls*, *prls*, *v_b*, *mx*, *mxtx*, *mx_perf*, *mxtx_perf*, *mxtwx*, *ybar*, *rlrl_bold*, *rlrl_perf*, *brlprl*, *yj*, *ajak_vb*, *cjck_vb*, *omega*, *W*)

yaj and ajak_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_det_fwdm.**compute_bRpR**(*brl*, *prl*, *nbConditions*, *nbVoxels*)

pyhrf.jde.asl_physio_hierarchical module

class pyhrf.jde.asl_physio_hierarchical.**ASLPhysioSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallback object>*, *bold_response_levels=<pyhrf.jde.asl_physio_hier object>*, *perf_response_levels=<pyhrf.jde.asl_physio_hier object>*, *labels=<pyhrf.jde.asl_physio_hierarchical.LabelSa object>*, *noise_var=<pyhrf.jde.asl_physio_hierarchical.No object>*, *true_brf=<pyhrf.jde.asl_physio_hierarchical.PhysioTr object>*, *true_brf_var=<pyhrf.jde.asl_physio_hierarchical.Phys object>*, *brf=<pyhrf.jde.asl_physio_hierarchical.PhysioBO object>*, *brf_var=<pyhrf.jde.asl_physio_hierarchical.Phys object>*, *prf=<pyhrf.jde.asl_physio_hierarchical.PhysioPe object>*, *prf_var=<pyhrf.jde.asl_physio_hierarchical.Phys object>*, *bold_mixt_params=<pyhrf.jde.asl_physio_hierar object>*, *perf_mixt_params=<pyhrf.jde.asl_physio_hierarc object>*, *drift=<pyhrf.jde.asl_physio_hierarchical.DriftCo object>*, *drift_var=<pyhrf.jde.asl_physio_hierarchical.Dri object>*, *perf_baseline=<pyhrf.jde.asl_physio_hierarchica object>*, *perf_baseline_var=<pyhrf.jde.asl_physio_hierarc object>*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

computeFit()

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

inputClass
alias of *WN_BiG_ASLSamplerInput*

parametersToShow = ['nb_its', 'response_levels', 'hrf', 'hrf_var']

class pyhrf.jde.asl_physio_hierarchical.**BOLDMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_hierarchical.**BOLDResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_hierarchical.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

computeVarYTildeOpt(*update_perf=False*)

if update_perf is True then also update sumcXg and prl.ytilde update_perf should only be used at init of variable values.

getOutputs()

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_hierarchical.**DriftCoeffSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*v*)

updateNorm()

class pyhrf.jde.asl_physio_hierarchical.**DriftVarianceSampler**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_hierarchical.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_hierarchical.**MixtureParamsSampler**(*name*, *response_level_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

get_current_means()

get_current_vars()

get_true_values_from_simulation_dict()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_hierarchical.**NoiseVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_hierarchical.**PerfBaselineSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_residuals()

compute_wa(*a=None*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_hierarchical.**PerfBaselineVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_hierarchical.**PerfMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_hierarchical.**PerfResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_hierarchical.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

computeVarYTildeOpt()

class pyhrf.jde.asl_physio_hierarchical.**PhysioBOLDResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *prior_type='not_regularized'*)

Bases: *pyhrf.jde.asl_physio_hierarchical.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum cWXg - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample BRF

changes to mean: changes to var:

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_hierarchical.**PhysioBOLDResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)
Sample variance of BRF

class pyhrf.jde.asl_physio_hierarchical.**PhysioPerfResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *diff_res=True*, *prior_type='not_regularized'*)

Bases: *pyhrf.jde.asl_physio_hierarchical.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum aXh - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample PRF with physio prior

changes to mean: add a factor of Omega h Sigma_g^-1 v_g^-1

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_hierarchical.**PhysioPerfResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)
Sample variance of PRF

class pyhrf.jde.asl_physio_hierarchical.**PhysioTrueBOLDResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *prior_type='regularized'*)

Bases: *pyhrf.jde.asl_physio_hierarchical.ResponseSampler*, *pyhrf.xmlio.Initable*

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample TRUE BRF

class pyhrf.jde.asl_physio_hierarchical.**PhysioTrueBOLDResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=F*

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)
Sample variance of BRF

class pyhrf.jde.asl_physio_hierarchical.**ResponseLevelSampler**(*name*, *response_name*, *mixture_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

computeRR()

computeVarYTildeOpt()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_hierarchical.**ResponseSampler**(*name*, *response_level_name*, *variance_name*, *prior_type*, *smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Generic parent class to perfusion response & BOLD response samplers

calcXResp(*resp*, *stackX=None*)

checkAndSetInitValue(*variables*)

computeYTilde()

getOutputs()

get_mat_X()

get_mat_XtX()

get_rlrl()

get_stackX()

get_ybar()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

setFinalValue()

updateNorm()

updateXResp()

class pyhrf.jde.asl_physio_hierarchical.**WN_BiG_ASLSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

pyhrf.jde.asl_physio_hierarchical.**b**()

pyhrf.jde.asl_physio_hierarchical.**compute_StS_StY**(*rls*, *v_b*, *mx*, *mxtx*, *ybar*, *rlrl*, *yaj*, *ajak_vb*)

yaj and ajak_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_hierarchical.**compute_StS_StY_deterministic**(*brls*, *prls*, *v_b*, *mx*, *mxtx*, *mwx*, *mxtwx*, *mwxtwx*, *ybar*, *rlrl_bold*, *rlrl_perf*, *brl-prl*, *omega*, *yj*, *ajak_vb*)

yj, ajak_vb and cjck_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_hierarchical.**compute_bRpR**(*brl*, *prl*, *nbConditions*, *nbVoxels*)

pyhrf.jde.asl_physio_joint module

class pyhrf.jde.asl_physio_joint.**ASLPhysioSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *bold_response_levels=<pyhrf.jde.asl_physio_joint.BOLDRe object>*, *perf_response_levels=<pyhrf.jde.asl_physio_joint.PerfRespo object>*, *labels=<pyhrf.jde.asl_physio_joint.LabelSampler object>*, *noise_var=<pyhrf.jde.asl_physio_joint.NoiseVarianceSampl object>*, *brf=<pyhrf.jde.asl_physio_joint.PhysioBOLDResponseSamp object>*, *prf=<pyhrf.jde.asl_physio_joint.PhysioPerfResponseSample object>*, *prf-brf_var=<pyhrf.jde.asl_physio_joint.PhysioJointResponseV object>*, *bold_mixt_params=<pyhrf.jde.asl_physio_joint.BOLDMixtu object>*, *perf_mixt_params=<pyhrf.jde.asl_physio_joint.PerfMixture object>*, *drift=<pyhrf.jde.asl_physio_joint.DriftCoeffSampler object>*, *drift_var=<pyhrf.jde.asl_physio_joint.DriftVarianceSampler object>*, *perf_baseline=<pyhrf.jde.asl_physio_joint.PerfBaselineSam object>*, *perf_baseline_var=<pyhrf.jde.asl_physio_joint.PerfBaseline object>*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

computeFit()

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

inputClass
alias of *WN_BiG_ASLSamplerInput*

parametersToShow = ['nb_its', 'response_levels', 'hrf', 'hrf_var']

class pyhrf.jde.asl_physio_joint.**BOLDMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.asl_physio_joint.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_joint.**BOLDResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.asl_physio_joint.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

computeVarYTildeOpt(*update_perf=False*)
if update_perf is True then also update sumcXg and prl.ytilde update_perf should only be used at init of variable values.

getOutputs()

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_joint.**DriftCoeffSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*v*)

updateNorm()

class pyhrf.jde.asl_physio_joint.**DriftVarianceSampler**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_joint.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_joint.**MixtureParamsSampler**(*name*, *response_level_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

get_current_means()

get_current_vars()

get_true_values_from_simulation_dict()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_joint.**NoiseVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_y_tilde()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.asl_physio_joint.**PerfBaselineSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

compute_residuals()

compute_wa(*a=None*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_joint.**PerfBaselineVarianceSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*v*)

class pyhrf.jde.asl_physio_joint.**PerfMixtureSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_joint.MixtureParamsSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

get_true_values_from_simulation_cdefs(*cdefs*)

class pyhrf.jde.asl_physio_joint.**PerfResponseLevelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_joint.ResponseLevelSampler*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*variables*)

computeVarYTildeOpt()

class pyhrf.jde.asl_physio_joint.**PhysioBOLDResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.asl_physio_joint.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()

y - sum cWXg - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)

Sample BRF

changes to mean: changes to var:

samplingWarmUp(*v*)

class pyhrf.jde.asl_physio_joint.**PhysioJointResponseVarianceSampler**(*val_ini=array([0.001])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*, *pyhrf.xmlio.Initable*

checkAndSetInitValue(*v*)

linkToData(*dataInput*)

sampleNextInternal(*v*)
Sample joint variance of BRF and PRF

class pyhrf.jde.asl_physio_joint.**PhysioPerfResponseSampler**(*smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*, *diff_res=True*)
Bases: *pyhrf.jde.asl_physio_joint.ResponseSampler*, *pyhrf.xmlio.Initable*

computeYTilde()
y - sum aXh - Pl - wa

get_mat_X()

get_mat_XtX()

get_stackX()

sampleNextInternal(*variables*)
Sample PRF with physio prior

changes to mean: add a factor of Omega h Sigma_g^-1 v_g^-1

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_joint.**ResponseLevelSampler**(*name*, *response_name*, *mixture_name*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

computeRR()

computeVarYTildeOpt()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

class pyhrf.jde.asl_physio_joint.**ResponseSampler**(*name*, *response_level_name*, *variance_name*, *smooth_order=2*, *zero_constraint=True*, *duration=25.0*, *normalise=1.0*, *val_ini=None*, *do_sampling=True*, *use_true_value=False*)
Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Generic parent class to perfusion response & BOLD response samplers

calcXResp(*resp*, *stackX=None*)

checkAndSetInitValue(*variables*)

computeYTilde()

getOutputs()

get_mat_X()

get_mat_XtX()

get_rlrl()

get_stackX()

get_ybar()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

setFinalValue()

updateNorm()

updateXResp()

class pyhrf.jde.asl_physio_joint.**WN_BiG_ASLSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

pyhrf.jde.asl_physio_joint.**b**()

pyhrf.jde.asl_physio_joint.**compute_StS_StY**(*rls*, *v_b*, *mx*, *mxtx*, *ybar*, *rlrl*, *yaj*, *ajak_vb*)

yaj and ajak_vb are only used to store intermediate quantities, they're not inputs.

pyhrf.jde.asl_physio_joint.**compute_bRpR**(*brl*, *prl*, *nbConditions*, *nbVoxels*)

pyhrf.jde.beta module

class pyhrf.jde.beta.**BetaSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=array([0.7])*, *sigma=0.05*, *pr_beta_cut=1.2*, *pf_method='es'*, *pf=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

getOutputs()

get_string_value(*v*)

linkToData(*dataInput*)

loadBetaGrid()

parametersComments = {'pf_method': 'either "es" (extrapolation scheme) or "ps" (path

parametersToShow = ['do_sampling', 'val_ini']

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

saveCurrentValue(*it*)

pyhrf.jde.beta.**Cpt_AcceptNewBeta_Graph**(*RefGraph*, *GraphNodesLabels*, *VecEstim_lnZ*, *VecBetaVal*, *CurrentBeta*, *sigma*, *thresh=1.2*, *GraphWeight=None*)

Starting from a given Beta vector (1 value for each condition) *CurrentBeta*, computes new Beta values in *NewBeta* using a Metropolis-Hastings step.

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists *i* such that *RefGraph[10][i]=2*
- **GraphWeight** – Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph. If not defined the weights are set to 1.0.
- **GraphNodesLabels** – Nodes labels. *GraphNodesLabels[i]* is the node *i* label.
- **VecEstim_lnZ** – Vector containing the *ln(Z(beta,mask))* estimates (in accordance with the defined mask).
- **VecBetaVal** – Vector of the same size as *VecExpectZ* containing the corresponding beta value (in accordance with the defined mask).
- **CurrentBeta** – Beta at the current iteration
- **sigma** – such as *NewBeta = CurrentBeta + N(0,sigma)*
- **thresh** – the prior on beta is uniform between 0 and thresh and linearly decrease between thresh and *VecBetaVal[-1]*
- **GraphWeight** – Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph. If not defined the weights are set to 1.0.

Returns Contains the accepted beta value at the next iteration

Return type NewBeta

pyhrf.jde.beta.**Cpt_Distrib_P_beta_graph**(*RefGraph*, *GraphNodesLabels*, *VecEstim_lnZ*, *VecBetaVal*, *thresh=1.5*, *GraphWeight=None*)

Computes the distribution $P(beta|q)$

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists *i* such that *RefGraph[10][i]=2*
- **GraphNodesLabels** – Nodes labels. *GraphNodesLabels[i]* is the node *i* label.
- **VecEstim_lnZ** – Vector containing the $ln(Z(beta, mask))$ estimates (in accordance with the defined graph).
- **VecBetaVal** – Vector of the same size as *VecExpectZ* containing the corresponding beta value (in accordance with the defined graph).
- **thresh** – the prior on beta is uniform between 0 and thresh and linearly decrease between thresh and *VecBetaVal[-1]*
- **GraphWeight** – Same shape as *RefGraph*. Each entry is the weight of the corresponding edge in *RefGraph*. If not defined the weights are set to 1.0.

Returns contains the $P(beta|q)$ values (consistent with *VecBetaVal*).

Return type Vec_P_Beta

pyhrf.jde.beta.**Cpt_Exact_lnZ_graph**(*RefGraph*, *beta*, *LabelsNb*, *GraphWeight=None*)

Computes the logarithm of the exact partition function $Z(\beta)$.

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that *RefGraph[10][i]=2*
- **beta** – spatial regularization parameter
- **LabelsNb** – number of labels in each site (typically 2 or 3)
- **GraphWeight** – Same shape as *RefGraph*. Each entry is the weight of the corresponding edge in *RefGraph*. If not defined the weights are set to 1.0.

Returns exact value of ln(Z)

Return type exact_lnZ

pyhrf.jde.beta.**Cpt_Expected_U_graph**(*RefGraph*, *beta*, *LabelsNb*, *SamplesNb*, *GraphWeight=None*, *GraphNodesLabels=None*, *GraphLinks=None*, *RefGrphNgbhPosi=None*)

Useless now!

Estimates the expectation of U for a given normalization constant Beta and a given mask shape. Swendsen-Wang sampling is used to assess the expectation on significant images depending of beta.

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists *i* such that *RefGraph[10][i]=2*
- **beta** – normalization constant
- **LabelsNb** – Labels number
- **SamplesNb** – Samples number for the U expectation estimation
- **GraphWeight** – Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph. If not defined the weights are set to 1.0.
- **GraphNodesLabels** – Optional list containing the nodes labels. The sampler aims to modify its values in function of beta and NbLabels. At this level this variable is seen as temporary and will be modified. Defining it slightly increases the calculation times.
- **GraphLinks** – Same shape as RefGraph. Each entry indicates if the link of the corresponding edge in RefGraph is considered (if yes ...=1 else ...=0). At this level this variable is seen as temporary and will be modified. Defining it slightly increases the calculation times.
- **RefGrphNgbhPosi** – Same shape as *RefGraph*. *RefGrphNgbhPosi[i][j]* indicates for which k is the link to i in *RefGraph[RefGraph[i][j]][k]*. This optional list is never modified.

Returns U expectation

Return type ExpectU

pyhrf.jde.beta.**Cpt_Vec_Estim_lnZ_Graph**(*RefGraph*, *LabelsNb*, *SamplesNb=40*, *BetaMax=1.4*, *BetaStep=0.05*, *GraphWeight=None*)

Estimates ln(Z) for fields of a given size and Beta values between 0 and BetaMax. Estimates of ln(Z) are first computed on a coarse grid of Beta values. They are then computed and returned on a fine grid. No approximation using precomputed partition function is performed here.

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists *i* such that *RefGraph[10][i]=2*
- **LabelsNb** – number of labels
- **SamplesNb** – number of fields estimated for each beta
- **BetaMax** – Z(beta,mask) will be computed for beta between 0 and BetaMax
- **BetaStep** – gap between two considered values of beta (...in the fine grid. This gap in the coarse grid is automatically fixed and depends on the graph size.)
- **GraphWeight** – Same shape as RefGraph. Each entry is the weight of the corresponding edge in RefGraph. If not defined the weights are set to 1.0.

Returns

- *VecEstim_lnZ* – Vector containing the ln(Z(beta,mask)) estimates
- *VecBetaVal* – Vector of the same size as VecExpectZ containing the corresponding beta value

pyhrf.jde.beta.**Cpt_Vec_Estim_lnZ_Graph_fast**(*RefGraph*, *LabelsNb*, *MaxErrorAllowed=5*, *BetaMax=1.4*, *BetaStep=0.05*)

Estimate ln(Z(beta)) of Potts fields. The default Beta grid is between 0. and 1.4 with a step of 0.05. Extrapolation algorithm is used. Fast estimates are only performed for Ising fields (2 labels). Reference partition functions were pre-computed on Ising fields designed on regular and non-regular grids. They all respect a 6-connectivity system.

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that *RefGraph[10][i]=2*
- **LabelsNb** – possible number of labels in each site of the graph
- **MaxErrorAllowed** – maximum error allowed in the graph estimation (in percents).
- **BetaMax** – Z(beta,mask) will be computed for beta between 0 and *BetaMax*. Maximum considered value is 1.4
- **BetaStep** – gap between two considered values of beta. Actual gaps are not exactly those asked but very close.

Returns

- *Est_lnZ* – Vector containing the ln(Z(beta)) estimates
- *V_Beta* – Vector of the same size as VecExpectZ containing the corresponding beta value

pyhrf.jde.beta.**Cpt_Vec_Estim_lnZ_Graph_fast2**(*RefGraph*, *BetaMax=1.4*, *BetaStep=0.05*)

Estimate ln(Z(beta)) of Ising fields (2 labels). The default Beta grid is between 0. and 1.4 with a step of 0.05. Bilinar estimation with the number of sites and cliques is used. The bilinear functions were estimated using bilinear regression on reference partition functions on 240 non-regular grids and with respect to a 6-connectivity system. (Pfs are found in LoadBaseLogPartFctRef -> PFs 0:239)

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists *i* such that *RefGraph[10][i]=2*

- **BetaMax** – Z(beta,mask) will be computed for beta between 0 and *BetaMax*. Maximum considered value is 1.4
- **BetaStep** – gap between two considered values of beta. Actual gaps are not exactly those asked but very close.

Returns

- *Est_lnZ* – Vector containing the ln(Z(beta)) estimates
- *V_Beta* – Vector of the same size as VecExpectZ containing the corresponding beta value

pyhrf.jde.beta.**Cpt_Vec_Estim_lnZ_Graph_fast3**(*RefGraph*, *LabelsNb*, *MaxErrorAllowed=5*, *BetaMax=1.4*, *BetaStep=0.05*)

Estimate ln(Z(beta)) of Potts fields. The default Beta grid is between 0. and 1.4 with a step of 0.05. Extrapolation algorithm is used. Fast estimates are only performed for Ising fields (2 labels). Reference partition functions were pre-computed on Ising fields designed on regular and non-regular grids. They all respect a 6-connectivity system.

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists i such that *RefGraph[10][i]=2*
- **LabelsNb** – possible number of labels in each site of the graph
- **MaxErrorAllowed** – maximum error allowed in the graph estimation (in percents).
- **BetaMax** – Z(beta,mask) will be computed for beta between 0 and *BetaMax*. Maximum considered value is 1.4
- **BetaStep** – gap between two considered values of beta. Actual gaps are not exactly those asked but very close.

Returns

- *Est_lnZ* – Vector containing the ln(Z(beta)) estimates
- *V_Beta* – Vector of the same size as *VecExpectZ* containing the corresponding beta value

pyhrf.jde.beta.**Cpt_Vec_Estim_lnZ_OLD_Graph**(*RefGraph*, *LabelsNb*, *SamplesNb=50*, *BetaMax=1.0*, *BetaStep=0.01*, *GraphWeight=None*)

Useless now!

Estimates ln(Z) for fields of a given size and Beta values between 0 and *BetaMax*.

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists *i* such that *RefGraph[10][i]=2*
- **LabelsNb** – number of labels
- **BetaMax** – Z(beta,mask) will be computed for beta between 0 and BetaMax
- **BetaStep** – gap between two considered values of bseta
- **GraphWeight** – Same shape as *RefGraph*. Each entry is the weight of the corresponding edge in RefGraph. If not defined the weights are set to 1.0.

Returns

- *VecEstim_lnZ* – Vector containing the ln(Z(beta,mask)) estimates
- *VecBetaVal* – Vector of the same size as VecExpectZ containing the corresponding beta value

pyhrf.jde.beta.**Cpt_Vec_Estim_lnZ_Onsager**(*n*, *BetaMax=1.2*, *BetaStep=0.05*)

Estimate ln(Z(beta)) Onsager using Onsager technique (2D periodic fields - 2 labels - 4 connectivity)

Parameters

- **n** – number of sites
- **BetaMax** – Z(beta,mask) will be computed for beta between 0 and *BetaMax*. Maximum considered value is 1.2.
- **BetaStep** – gap between two considered values of beta. Actual gaps are not exactly those asked but very close.

Returns

- *Est_lnZ* – Vector containing the ln(Z(beta)) estimates
- *V_Beta* – Vector of the same size as *VecExpectZ* containing the corresponding beta value

pyhrf.jde.beta.**Estim_lnZ_Onsager**(*n*, *beta*)

Estimate ln(Z(beta)) using Onsager technique (2D periodic fields - 2 labels - 4 connectivity)

Parameters

- **n** – number of sites
- **beta** – beta

Returns ln(Z(beta)) estimate

Return type LogZ

pyhrf.jde.beta.**Estim_lnZ_ngbhd_graph**(*RefGraph*, *beta_Ngbhd*, *beta_Ref*, *lnZ_ref*, *VecU_ref*, *LabelsNb*)

Estimates ln(Z) for beta=betaNgbhd. *beta_Ngbhd* is supposed close to *beta_Ref* for which ln(Z) is known (lnZ_ref) and the energy U of fields generated according to it have already been computed (*VecU_ref*).

Parameters

- **RefGraph** – List which contains the connectivity graph. Each entry represents a node of the graph and contains the list of its neighbors entry location in the graph. Ex: *RefGraph[2][3]=10* means 3rd neighbour of the 2nd node is the 10th node. => There exists *i* such that *RefGraph[10][i]=2*
- **beta_Ngbhd** – normalization constant for which ln(Z) will be estimated
- **beta_Ref** – normalization constant close to beta_Ngbhd for which ln(Z) already known
- **lnZ_ref** – ln(Z) for beta=beta_Ref
- **VecU_ref** – energy U of fields generated according to beta_Ref
- **LabelsNb** – Labels number

Returns ln(Z) for beta=beta_Ngbhd

Return type lnZ_Ngbhd

pyhrf.jde.beta.**LoadBaseLogPartFctRef**()

output:

- BaseLogPartFctRef: dictionnary that contains the data base of log-PF (first value = nb labels / second value = nb. sites / third value = nb. cliques)

- V_Beta_Ref: Beta grid corresponding to the log-PF values in 'Est_lnZ_Ref'

pyhrf.jde.beta.**beta_estim_obs_field**(*graph*, *labels*, *gridLnz*, *method='MAP'*, *weights=None*)

Estimate the amount of spatial correlation of an Ising observed field. *graph* is the neighbours list defining the topology *labels* is the field realisation *gridLnz* is the log-partition function associated to the topology, ie a grid where *gridLnz[0]* stores values of lnz and *gridLnz[1]* stores corresponding values of beta.

Returns

- *estimated beta*
- *tabulated distribution p(beta|labels)*

pyhrf.jde.beta.**logpf_ising_onsager**(*size*, *beta*)

Calculate log partition function in terms of beta for an Ising field of size 'size'. 'beta' can be scalar or numpy.array. Assumptions: the field is 2D, squared, toroidal and has 4-connectivity

pyhrf.jde.drift module

class pyhrf.jde.drift.**DriftARSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Gibbs sampler of the parameters modelling the low frequency drift in the fMRI time course, in the case of AR noise

checkAndSetInitValue(*variables*)

computeVarYTilde(*varNrls*, *varXh*)

fillOutputs2(*outputs*, *iROI=-1*)

finalizeSampling()

initOutputs2(*outputs*, *nbROI=-1*)

linkToData(*dataInput*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)
#TODO : comment

updateNorm()

updateVarYmDrift()

class pyhrf.jde.drift.**DriftSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Gibbs sampler of the parameters modelling the low frequency drift in the fMRI time course, in the case of white noise.

checkAndSetInitValue(*variables*)

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

updateNorm()

class pyhrf.jde.drift.**DriftSamplerWithRelVar**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.jde.drift.DriftSampler*

Gibbs sampler of the parameters modelling the low frequency drift in the fMRI time course, in the case of white noise.

checkAndSetInitValue(*variables*)

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

updateNorm()

class pyhrf.jde.drift.**ETASampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=array([1.])*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Gibbs sampler of the variance of the Inverse Gamma prior used to regularise the estimation of the low frequency drift embedded in the fMRI time course

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.drift.**ETASampler_MultiSess**(*do_sampling=True*, *use_true_value=False*, *val_ini=array([1.])*)

Bases: *pyhrf.jde.drift.ETASampler*

linkToData(*dataInput*)

sampleNextInternal(*variables*)

pyhrf.jde.drift.**sampleDrift**(*varInvSigma_drift*, *ptLambdaY*, *dim*)

pyhrf.jde.hrf module

class pyhrf.jde.hrf.**HRFARSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *do_voxelwise_outputs=False*, *compute_ah_online=False*, *output_ah=False*)

Bases: *pyhrf.jde.hrf.HRFSampler*

#THis class implements the sampling of the HRF when modelling a serially AR(1) noise process in the data. The structure of this noise is spatially varying in the sense that there is one AR parameter in combination with one noise variance per voxel.

computeStDS_StDY(*reps*, *noiseInvCov*, *nrls*, *varMBYPl*)

finalizeSampling()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.hrf.**HRFSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *do_voxelwise_outputs=False*, *compute_ah_online=False*, *output_ah=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : HRF sampler for BiGaussian NLR mixture

calcXh(*hrf*)

checkAndSetInitValue(*variables*)

computeStDS_StDY(*rb*, *nrls*, *aa*)

detectSignError()

finalizeSampling()

getCurrentVar()

getFinalVar()

getOutputs()

getScaleFactor()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

get_final_value()
Used to compare with simulated value

initObservables()

linkToData(*dataInput*)

parametersComments = {'prior_type': 'Type of prior:\n - "singleHRF": one HRF modelle

parametersToShow = ['do_sampling', 'duration', 'zero_constraint']

reportCurrentVal()

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

updateNorm()

updateObsersables()

updateXh()

class pyhrf.jde.hrf.**HRFSamplerWithRelVar**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *do_voxelwise_outputs=False*, *compute_ah_online=False*, *output_ah=False*)

Bases: *pyhrf.jde.hrf.HRFSampler*

This class introduce a new variable w (Relevant Variable) that takes its value in {0, 1} with :

- w = 1 condition m is relevant in the studied parcel

- w = 1 otherwise

computeStDS_StDY_WithRelVar(*rb*, *nrls*, *aa*, *w*)

finalizeSampling()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.hrf.**HRF_Drift_Sampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *do_voxelwise_outputs=False*, *compute_ah_online=False*, *output_ah=False*)

Bases: *pyhrf.jde.hrf.HRFSampler*

Class handling the Gibbs sampling of Neural Response Levels in the case of joint drift sampling.

computeStDS_StDY(*rb*, *nrls*, *aa*)

class pyhrf.jde.hrf.**HRF_Drift_SamplerWithRelVar**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *do_voxelwise_outputs=False*, *compute_ah_online=False*, *output_ah=False*)

Bases: *pyhrf.jde.hrf.HRFSamplerWithRelVar*

Class handling the Gibbs sampling of Neural Response Levels in the case of joint drift sampling.

computeStDS_StDY_WithRelVar(*rb*, *nrls*, *aa*, *w*)

class pyhrf.jde.hrf.**HRF_two_parts_Sampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *do_voxelwise_outputs=False*, *compute_ah_online=False*, *output_ah=False*)

Bases: *pyhrf.jde.hrf.HRFSampler*

calcXh(*hrf*)

checkAndSetInitValue(*variables*)

computeStDS_StDY(*rb*, *nrls*, *aa*)

detectSignError()

finalizeSampling()

getCurrentVar()

getFinalVar()

getOutputs()

getScaleFactor()

initObservables()

linkToData(*dataInput*)

reportCurrentVal()

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

updateNorm()

updateObsersables()

updateXh()

class pyhrf.jde.hrf.**HRFwithHabSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *do_voxelwise_outputs=False*, *compute_ah_online=False*, *output_ah=False*)

Bases: *pyhrf.jde.hrf.HRFSampler*

computeStDS_StDY(*rb*, *sumaX*, *Q*)

finalizeSampling()

getScaleFactor()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

updateNorm()

class pyhrf.jde.hrf.**RHSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=array([0.1])*, *prior_mean=0.001*, *prior_var=10*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

checkAndSetInitValue(*variables*)

getOutputs()

get_final_value()

linkToData(*dataInput*)

parametersToShow = ['do_sampling', 'val_ini']

sampleNextInternal(*variables*)

class pyhrf.jde.hrf.**ScaleSampler**(*do_sampling=False*, *use_true_value=False*, *val_ini=array([1.])*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

pyhrf.jde.hrf.**buildDiagGaussianMat**(*size*, *width*)

pyhrf.jde.hrf.**msqrt**(*cov*)
sig = msqrt(cov)

Return a matrix square root of a covariance matrix. Tries Cholesky factorization first, and factorizes by diagonalization if that fails.

pyhrf.jde.hrf.**sampleHRF_single_hrf**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*)

pyhrf.jde.hrf.**sampleHRF_single_hrf_hack**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*)

pyhrf.jde.hrf.**sampleHRF_voxelwise_iid**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*)

pyhrf.jde.jde_multi_sess module

class pyhrf.jde.jde_multi_sess.**BOLDGibbs_Multi_SessSampler**(*nb_its=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallb object>*, *response_levels_sess=<pyhrf.jde.jde_multi_sess. object>*, *response_levels_mean=<pyhrf.jde.jde_multi_ses object>*, *beta=<pyhrf.jde.beta.BetaSampler object>*, *noise_var=<pyhrf.jde.jde_multi_sess.NoiseVar object>*, *hrf=<pyhrf.jde.jde_multi_sess.HRF_MultiSess object>*, *hrf_var=<pyhrf.jde.hrf.RHSampler object>*, *mixt_weights=<pyhrf.jde.nrl.bigaussian.Mixtu object>*, *mixt_params=<pyhrf.jde.nrl.bigaussian.BiGau object>*, *scale=<pyhrf.jde.hrf.ScaleSampler object>*, *drift=<pyhrf.jde.jde_multi_sess.Drift_MultiSes object>*, *drift_var=<pyhrf.jde.jde_multi_sess.ETASamp object>*, *stop_crit_threshold=-1*, *stop_crit_from_start=False*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

cleanObservables()

computeFit()

computePMStimInducedSignal()

compute_crit_diff(*old_vals*, *means=None*)

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

initGlobalObservables()

inputClass
alias of *BOLDSampler_Multi_SessInput*

parametersComments = {'obs_hist_pace': 'See comment for samplesHistoryPaceSave.', 'sm

parametersToShow = ['nb_its', 'response_levels_sess', 'response_levels_mean', 'hrf', '

saveGlobalObservables(*it*)

stop_criterion(*it*)

updateGlobalObservables()

class pyhrf.jde.jde_multi_sess.**BOLDSampler_Multi_SessInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Class holding data needed by the sampler : BOLD time courses for each voxel, onsets and voxel topology. It also perform some precalculation such as the convolution matrix based on the onsests (L{stackX}) —- Multi-sessions version

buildCosMat(*paramLFD*, *ny*)

buildOtherMatX()

buildParadigmConvolMatrix(*zc*, *estimDuration*, *availableDataIndex*, *parData*)

buildPolyMat(*paramLFD*, *n*)

calcDt(*dtMin*)

chewUpOnsets(*dt*, *hrfZc*, *hrfDuration*)

cleanMem()

cleanPrecalculations()

makePrecalculations()

setLFDMat(*paramLFD*, *typeLFD*)
Build the low frequency basis from polynomial basis functions.

class pyhrf.jde.jde_multi_sess.**BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler**(*do_sampling=Tr*
use_true_value=
val_ini=None,
hy-
per_prior_type=
ac-
tiv_thresh=4.0,
var_ci_pr_alpha
var_ci_pr_beta=
var_ca_pr_alpha
var_ca_pr_beta=
mean_ca_pr_me
mean_ca_pr_var

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

computeWithProperPriors(*j*, *cardCIj*, *cardCAj*)

finalizeSampling()

getCurrentMeans()

getCurrentVars()

getOutputs()

get_string_value(*v*)

linkToData(*dataInput*)

parametersComments = {'activ_thresh': 'Threshold for the max activ mean above which t

parametersToShow = []

sampleNextInternal(*variables*)

updateObsersables()

class pyhrf.jde.jde_multi_sess.**Drift_MultiSess_Sampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

getOutputs()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

get_final_value()

get_true_value()

linkToData(*dataInput*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

updateNorm()

class pyhrf.jde.jde_multi_sess.**ETASampler_MultiSess**(*do_sampling=True*, *use_true_value=False*, *val_ini=array([1.])*)

Bases: *pyhrf.jde.drift.ETASampler*

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sess.**HRF_MultiSess_Sampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *do_voxelwise_outputs=False*, *compute_ah_online=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

HRF sampler for multisession model

calcXh(*hrf*)

checkAndSetInitValue(*variables*)

computeStDS_StDY(*rb_allSess*, *nrls_allSess*, *aa_allSess*)

computeStDS_StDY_from_HRFSampler(*rb*, *nrls*, *aa*)
just for comparison purpose. Should be removed in the end.

computeStDS_StDY_one_session(*rb*, *nrls*, *aa*, *sess*)

finalizeSampling()

getCurrentVar()

getFinalVar()

getOutputs()

getScaleFactor()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

initObservables()

linkToData(*dataInput*)

parametersComments = {'prior_type': 'Type of prior:\n - "singleHRF": one HRF modelle

parametersToShow = ['do_sampling', 'duration', 'zero_constraint']

reportCurrentVal()

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

updateNorm()

updateObsersables()

updateXh()

class pyhrf.jde.jde_multi_sess.**NRL_Multi_Sess_Sampler**(*do_sampling=True*, *val_ini=None*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

cleanMemory()

computeAA(*nrls*, *destaa*)

computeComponentsApost(*s*, *m*, *varXh*)

computeVarYTildeSessionOpt(*varXh*, *s*)

finalizeSampling()

getOutputs()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

is_accurate()

linkToData(*dataInput*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)
#TODO : comment

saveCurrentValue(*it*)

class pyhrf.jde.jde_multi_sess.**NRLsBar_Drift_Multi_Sess_Sampler**(*do_sampling=True*, *val_ini=None*, *contrasts={}*, *do_label_sampling=True*, *use_true_nrls=False*, *use_true_labels=False*, *labels_ini=None*, *ppm_proba_threshold=0.05*, *ppm_value_threshold=0*, *ppm_value_multi_threshold=array([0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2., 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3., 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.])*, *mean_activation_threshold=4*, *rescale_results=False*, *wip_variance_computation=False*)

Bases: *pyhrf.jde.nrl.bigaussian.NRLSampler*

Class handling the Gibbs sampling of Neural Response Levels in the case of joint drift sampling.

checkAndSetInitValue(*variables*)

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

is_accurate()

linkToData(*dataInput*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

sampleNrlsSerial(*varCI*, *varCA*, *meanCA*, *variables*)

samplingWarmUp(*variables*)
#TODO : comment

setFinalValue()

class pyhrf.jde.jde_multi_sess.**NoiseVariance_Drift_Multi_Sess_Sampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sess.**Variance_GaussianNRL_Multi_Sess**(*do_sampling=True*, *use_true_value=False*, *val_ini=array([1.])*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

pyhrf.jde.jde_multi_sess.**b**()

pyhrf.jde.jde_multi_sess.**permutation**(*x*)

Randomly permute a sequence, or return a permuted range.

If *x* is a multi-dimensional array, it is only shuffled along its first index.

Parameters **x** (*int or array_like*) – If *x* is an integer, randomly permute `np.arange(x)`. If *x* is an array, make a copy and shuffle the elements randomly.

Returns **out** – Permuted sequence or array range.

Return type *ndarray*

Examples

```
>>> np.random.permutation(10)
array([1, 7, 4, 3, 0, 9, 2, 5, 8, 6])
```

```
>>> np.random.permutation([1, 4, 9, 12, 15])
array([15,  1,  9,  4, 12])
```

```
>>> arr = np.arange(9).reshape((3, 3))
>>> np.random.permutation(arr)
array([[6, 7, 8],
       [0, 1, 2],
       [3, 4, 5]])
```

pyhrf.jde.jde_multi_sess.**rand**(*d0*, *d1*, ..., *dn*)

Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.

Returns **out** – Random values.

Return type ndarray, shape `(d0, d1, ..., dn)`

See also:

`random()`

Notes

This is a convenience function. If you want an interface that takes a shape-tuple as the first argument, refer to np.random.random_sample .

Examples

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

pyhrf.jde.jde_multi_sess.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

`random.standard_normal()` Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.jde.jde_multi_sess.**sampleHRF_single_hrf**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*)

pyhrf.jde.jde_multi_sess.**sampleHRF_single_hrf_hack**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*)

pyhrf.jde.jde_multi_sess.**sampleHRF_voxelwise_iid**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*, *nbSess*)

pyhrf.jde.jde_multi_sess.**simulate_sessions**(*output_dir*, *snr_scenario='high_snr'*, *spatial_size='tiny'*)

pyhrf.jde.jde_multi_sess.**simulate_single_session**(*output_dir*, *var_sessions_nrls*, *cdefs*, *nrls_bar*, *labels*, *labels_vol*, *v_noise*, *drift_coeff_var*, *drift_amplitude*)

pyhrf.jde.jde_multi_sujets module

class pyhrf.jde.jde_multi_sujets.**BOLDGibbs_Multi_SubjSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCa object>*, *response_levels=<pyhrf.jde.jde_multi_sujets. object>*, *noise_var=<pyhrf.jde.jde_multi_sujets.Nois instance>*, *hrf_subj=<pyhrf.jde.jde_multi_sujets.HRF_ instance>*, *hrf_var_subj=<pyhrf.jde.jde_multi_sujets.H instance>*, *hrf_group=<pyhrf.jde.jde_multi_sujets.HRF instance>*, *hrf_var_group=<pyhrf.jde.jde_multi_sujets instance>*, *mixt_params=<pyhrf.jde.jde_multi_sujets.M instance>*, *labels=<pyhrf.jde.jde_multi_sujets.LabelSam instance>*, *drift=<pyhrf.jde.jde_multi_sujets.Drift_Mu instance>*, *drift_var=<pyhrf.jde.jde_multi_sujets.ETAS instance>*, *stop_crit_threshold=-1*, *stop_crit_from_start=False*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

cleanObservables()

computeFit()

computePMStimInducedSignal()

compute_crit_diff(*old_vals*, *means=None*)

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

initGlobalObservables()

inputClass
alias of *BOLDSampler_MultiSujInput*

parametersComments = {'obs_hist_pace': 'See comment for samplesHistoryPaceSave.', 'sm

parametersToShow = ['nb_iterations', 'response_levels', 'hrf_subj', 'hrf_var_subj', 'h

saveGlobalObservables(*it*)

stop_criterion(*it*)

updateGlobalObservables()

class pyhrf.jde.jde_multi_sujets.**BOLDSampler_MultiSujInput**(*GroupData*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Class holding data needed by the sampler : BOLD time courses for each voxel, onsets and voxel topology. It also perform some precalculation such as the convolution matrix based on the onsests (L{stackX}) —- Multi-subjects version (cf. merge_fmri_subjects in core.py)

buildCosMat(*paramLFD*, *ny*)

buildOtherMatX()

buildParadigmConvolMatrix(*zc*, *estimDuration*, *availableDataIndex*, *parData*)

buildPolyMat(*paramLFD*, *n*)

calcDt(*dtMin*)

chewUpOnsets(*dt*, *hrfZc*, *hrfDuration*)

cleanMem()

makePrecalculations()

setLFDMat(*paramLFD*, *typeLFD*)

Build the low frequency basis from polynomial basis functions.

class pyhrf.jde.jde_multi_sujets.**Drift_MultiSubj_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Gibbs sampler of the parameters modelling the low frequency drift in the fMRI time course, in the case of white noise.

checkAndSetInitValue(*variables*)

getOutputs()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

get_final_value()

get_true_value()

linkToData(*dataInput*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

updateNorm()

class pyhrf.jde.jde_multi_sujets.**ETASampler_MultiSubj**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Gibbs sampler of the variance of the Inverse Gamma prior used to regularise the estimation of the low frequency drift embedded in the fMRI time course

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets.**HRFVarianceSubjectSampler**(*val_ini=array([0.15])*, *do_sampling=True*, *use_true_value=False*, *prior_mean=0.001*, *prior_var=10.0*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

checkAndSetInitValue(*variables*)

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets.**HRF_Group_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*, *duration=25.0*, *zero_contraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *regularise=True*, *only_hrf_subj=False*, *compute_ah_online=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

HRF sampler for multisubjects model

P_COMPUTE_AH_ONLINE = 'compute_ah_online'

P_COVAR_HACK = 'hackCovarApost'

P_DERIV_ORDER = 'derivOrder'

P_DURATION = 'duration'

P_NORMALISE = 'normalise'

P_OUTPUT_PMHRF = 'writeHrfOutput'

P_PRIOR_TYPE = 'priorType'

P_REGULARIZE = 'regularize_hrf'

P_SAMPLE_FLAG = 'sampleFlag'

P_USE_TRUE_VALUE = 'useTrueValue'

P_VAL_INI = 'initialValue'

P_VOXELWISE_OUTPUTS = 'voxelwiseOutputs'

P_ZERO_CONSTR = 'zeroConstraint'

checkAndSetInitValue(*variables*)

defaultParameters = {'hackCovarApost': False, 'voxelwiseOutputs': False, 'initialVal

finalizeSampling()

getCurrentVar()

getFinalVar()

getOutputs()

getScaleFactor()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

get_true_value()

linkToData(*dataInput*)

parametersComments = {'hackCovarApost': 'Divide the term coming from the likelihood by

parametersToShow = ['duration', 'zeroConstraint', 'sampleFlag', 'writeHrfOutput']

reportCurrentVal()

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

updateNorm()

updateObsersables()

class pyhrf.jde.jde_multi_sujets.**HRF_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*, *duration=25.0*, *zero_contraint=True*, *normalise=1.0*, *deriv_order=2*, *covar_hack=False*, *prior_type='voxelwiseIID'*, *regularise=True*, *only_hrf_subj=False*, *compute_ah_online=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

HRF sampler for multi subject model

calcXh(*hrfs*)

checkAndSetInitValue(*variables*)

computeStDS_StDY(*rb_allSubj*, *nrls_allSubj*, *aa_allSubj*)

computeStDS_StDY_one_subject(*rb*, *nrls*, *aa*, *subj*)

finalizeSampling()

getCurrentVar()

getFinalVar()

getOutputs()

getScaleFactor()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

get_true_value()

initObservables()

linkToData(*dataInput*)

parametersComments = {'prior_type': 'Type of prior:\n - "singleHRF": one HRF modelle

reportCurrentVal()

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

updateNorm()

updateObsersables()

updateXh()

class pyhrf.jde.jde_multi_sujets.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

FALSE_NEG = 3

FALSE_POS = 2

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.jde_multi_sujets.**MixtureParamsSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *s*, *cardCIj*, *cardCAj*)

get_current_means()

return array of shape (class, subject, condition)

get_current_vars()
return array of shape (class, subject, condition)

get_true_values_from_simulation_cdefs(*cdefs*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets.**NRLs_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

computeAA()

computeVarYTildeOpt(*varXh*, *s*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

class pyhrf.jde.jde_multi_sujets.**NoiseVariance_Drift_MultiSubj_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets.**RHGroupSampler**(*val_ini=array([0.15])*, *do_sampling=True*, *use_true_value=False*, *prior_mean=0.001*, *prior_var=10.0*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

checkAndSetInitValue(*variables*)

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets.**Variance_GaussianNRL_Multi_Subj**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

pyhrf.jde.jde_multi_sujets.**b**()

pyhrf.jde.jde_multi_sujets.**create_gaussian_hrf_subject_and_group**(*hrf_group_base*, *hrf_group_var_base*, *hrf_subject_var_base*, *dt*, *alpha=0.0*)

pyhrf.jde.jde_multi_sujets.**create_unnormed_gaussian_hrf_subject**(*unnormed_hrf_group*, *unnormed_var_subject_hrf*, *dt*, *alpha=0.0*)

Creation of hrf by subject. Use group level hrf and variance for each subject (var_subjects_hrfs must be a list) Simulated hrfs must be smooth enough: correlation between temporal coeffcients

pyhrf.jde.jde_multi_sujets.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.jde.jde_multi_sujets.**rescale_hrf_group**(*unnormed_primary_hrf*, *unnormed_hrf_group*)

pyhrf.jde.jde_multi_sujets.**rescale_hrf_subj**(*unnormed_primary_hrf*)

pyhrf.jde.jde_multi_sujets.**rescale_hrf_subj_var**(*unnormed_primary_hrf*, *unnormed_var_subject_hrf*)

pyhrf.jde.jde_multi_sujets.**sampleHRF_single_hrf**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*, *hgroup*)

pyhrf.jde.jde_multi_sujets.**sampleHRF_single_hrf_hack**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*, *hgroup*)

pyhrf.jde.jde_multi_sujets.**sampleHRF_voxelwise_iid**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*, *hgroup*, *nbsubj*)

pyhrf.jde.jde_multi_sujets.**simulate_single_subject**(*output_dir*, *cdefs*, *var_subject_hrf*, *labels*, *labels_vol*, *v_noise*, *drift_coeff_var*, *drift_amplitude*, *hrf_group_level*, *var_hrf_group*, *dt=0.6*, *dsf=4*)

pyhrf.jde.jde_multi_sujets.**simulate_subjects**(*output_dir*, *snr_scenario='high_snr'*, *spatial_size='tiny'*, *hrf_group=None*, *nbSubj=10*)

Simulate daata for multiple subjects (5 subjects by default)

pyhrf.jde.jde_multi_sujets_alpha module

class pyhrf.jde.jde_multi_sujets_alpha.**AlphaVar_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Gibbs sampler of the variance of the Inverse Gamma prior used to regularise the estimation of the low frequency drift embedded in the fMRI time course

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets_alpha.**Alpha_hgroup_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

parametersToShow = []

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets_alpha.**BOLDGibbs_Multi_SubjSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSD object>*, *bold_response_levels_subj=<pyhrf object>*, *labels=<pyhrf.jde.jde_multi_sujets_ object>*, *noise_var=<pyhrf.jde.jde_multi_su object>*, *hrf_subj=<pyhrf.jde.jde_multi_suj object>*, *hrf_subj_var=<pyhrf.jde.jde_multi object>*, *hrf_group=<pyhrf.jde.jde_multi_su object>*, *hrf_group_var=<pyhrf.jde.jde_mul object>*, *mixt_params=<pyhrf.jde.jde_multi object>*, *drift=<pyhrf.jde.jde_multi_sujets_ object>*, *drift_var=<pyhrf.jde.jde_multi_suj object>*, *alpha=<pyhrf.jde.jde_multi_sujets_ object>*, *alpha_var=<pyhrf.jde.jde_multi_suj object>*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

cleanObservables()

computeFit()

computePMStimInducedSignal()

compute_crit_diff(*old_vals*, *means=None*)

default_nb_its = 3000

finalizeSampling()

getGlobalOutputs()

initGlobalObservables()

inputClass
alias of *BOLDSampler_MultiSujInput*

saveGlobalObservables(*it*)

stop_criterion(*it*)

updateGlobalObservables()

class pyhrf.jde.jde_multi_sujets_alpha.**BOLDSampler_MultiSujInput**(*GroupData*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Class holding data needed by the sampler : BOLD time courses for each voxel, onsets and voxel topology. It also perform some precalculation such as the convolution matrix based on the onsests (L{stackX}) —- Multi-subjects version (cf. merge_fmri_subjects in core.py)

buildCosMat(*paramLFD*, *ny*)

buildOtherMatX()

buildParadigmConvolMatrix(*zc*, *estimDuration*, *availableDataIndex*, *parData*)

buildPolyMat(*paramLFD*, *n*)

calcDt(*dtMin*)

chewUpOnsets(*dt*, *hrfZc*, *hrfDuration*)

cleanMem()

makePrecalculations()

setLFDMat(*paramLFD*, *typeLFD*)

Build the low frequency basis from polynomial basis functions.

class pyhrf.jde.jde_multi_sujets_alpha.**BiGaussMixtureParamsSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*, *prior_type='Jeffrey'*, *var_ci_pr_alpha=2.04*, *var_ci_pr_beta=2.08*, *var_ca_pr_alpha=2.01*, *var_ca_pr_beta=0.5*, *mean_ca_pr_mean=5.0*, *mean_ca_pr_var=20.0*, *mean_activation_threshold=4.0*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *cardCIj*, *cardCAj*)

computeWithProperPriors(*j*, *cardCIj*, *cardCAj*)

finalizeSampling()

getCurrentMeans()

getCurrentVars()

getOutputs()

get_string_value(*v*)

linkToData(*dataInput*)

parametersComments = {'prior_type': "Either 'proper' or 'Jeffrey'", 'mean_activation_

sampleNextInternal(*variables*)

updateObsersables()

class pyhrf.jde.jde_multi_sujets_alpha.**Drift_MultiSubj_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Gibbs sampler of the parameters modelling the low frequency drift in the fMRI time course, in the case of white noise.

P_SAMPLE_FLAG = 'sampleFlag'

P_USE_TRUE_VALUE = 'useTrueValue'

P_VAL_INI = 'initialValue'

checkAndSetInitValue(*variables*)

defaultParameters = {'useTrueValue': False, 'initialValue': None, 'sampleFlag': Tru

getOutputs()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

get_final_value()

get_true_value()

linkToData(*dataInput*)

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

updateNorm()

class pyhrf.jde.jde_multi_sujets_alpha.**ETASampler_MultiSubj**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

Gibbs sampler of the variance of the Inverse Gamma prior used to regularise the estimation of the low frequency drift embedded in the fMRI time course

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets_alpha.**HRFVarianceSubjectSampler**(*val_ini=array([0.05])*, *do_sampling=False*, *use_true_value=False*, *pr_mean=0.001*, *pr_var=10.0*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

PR_MEAN = 0.001

PR_VAR = 10.0

VAL_INI = 0.05

checkAndSetInitValue(*variables*)

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets_alpha.**HRF_Group_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *derivOrder=2*, *output_hrf_pm=True*, *hack_covar_apost=False*, *prior_type='voxelwiseIID'*, *compute_ah_online=False*, *regularize_hrf=True*, *model_subjects_only=False*, *voxelwise_outputs=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

HRF sampler for multisubjects model

checkAndSetInitValue(*variables*)

finalizeSampling()

getCurrentVar()

getFinalVar()

getOutputs()

getScaleFactor()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

linkToData(*dataInput*)

parametersComments = {'prior_type': 'Type of prior:\n - "singleHRF": one HRF modelle

parametersToShow = ['duration', 'zero_constraint', 'do_sampling', 'output_hrf_pm']

reportCurrentVal()

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

updateNorm()

updateObsersables()

class pyhrf.jde.jde_multi_sujets_alpha.**HRF_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*, *duration=25.0*, *zero_constraint=True*, *normalise=1.0*, *derivOrder=2*, *output_hrf_pm=True*, *hack_covar_apost=False*, *prior_type='voxelwiseIID'*, *compute_ah_online=False*, *regularize_hrf=True*, *model_subjects_only=False*, *voxelwise_outputs=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

HRF sampler for multisession model

calcXh(*hrfs*)

checkAndSetInitValue(*variables*)

computeStDS_StDY(*rb_allSubj*, *nrls_allSubj*, *aa_allSubj*)

computeStDS_StDY_one_subject(*rb*, *nrls*, *aa*, *subj*)

finalizeSampling()

getCurrentVar()

getFinalVar()

getOutputs()

getScaleFactor()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

initObservables()

linkToData(*dataInput*)

parametersComments = {'prior_type': 'Type of prior:\n - "singleHRF": one HRF modelle

parametersToShow = ['duration', 'zero_constraint', 'do_sampling', 'output_hrf_pm']

reportCurrentVal()

sampleNextAlt(*variables*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

setFinalValue()

updateNorm()

updateObsersables()

updateXh()

class pyhrf.jde.jde_multi_sujets_alpha.**LabelSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

compute_ext_field()

countLabels()

linkToData(*dataInput*)

sampleNextInternal(*v*)

samplingWarmUp(*v*)

class pyhrf.jde.jde_multi_sujets_alpha.**MixtureParamsSampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

I_MEAN_CA = 0

I_VAR_CA = 1

I_VAR_CI = 2

L_CA = 1

L_CI = 0

NB_PARAMS = 3

PARAMS_NAMES = ['Mean_Activ', 'Var_Activ', 'Var_Inactiv']

checkAndSetInitValue(*variables*)

computeWithJeffreyPriors(*j*, *s*, *cardCIj*, *cardCAj*)

get_current_means()
return array of shape (class, subject, condition)

get_current_vars()
return array of shape (class, subject, condition)

get_true_values_from_simulation_cdefs(*cdefs*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets_alpha.**NRLs_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

FALSE_NEG = 3

FALSE_POS = 2

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

computeAA()

computeVarYTildeOpt(*varXh*, *s*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

samplingWarmUp(*variables*)

class pyhrf.jde.jde_multi_sujets_alpha.**NoiseVariance_Drift_MultiSubj_Sampler**(*val_ini=None*, *do_sampling=True*, *use_true_value=Fal*

Bases: *pyhrf.jde.noise.NoiseVariance_Drift_Sampler*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

parametersToShow = []

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets_alpha.**RHGroupSampler**(*val_ini=array([0.05])*, *do_sampling=False*, *use_true_value=False*, *pr_mean=0.001*, *pr_var=10.0*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

PR_MEAN = 0.001

PR_VAR = 10.0

VAL_INI = 0.05

checkAndSetInitValue(*variables*)

getOutputs()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.jde_multi_sujets_alpha.**Variance_GaussianNRL_Multi_Subj**(*val_ini=array([1.])*, *do_sampling=True*, *use_true_value=False*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

pyhrf.jde.jde_multi_sujets_alpha.**b**()

pyhrf.jde.jde_multi_sujets_alpha.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.jde.jde_multi_sujets_alpha.**sampleHRF_single_hrf**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*, *hgroup*, *reg*)

pyhrf.jde.jde_multi_sujets_alpha.**sampleHRF_single_hrf_hack**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*, *hgroup*)

pyhrf.jde.jde_multi_sujets_alpha.**sampleHRF_voxelwise_iid**(*stLambdaS*, *stLambdaY*, *varR*, *rh*, *nbColX*, *nbVox*, *hgroup*, *only_hrf_subj*, *reg*, *nbsubj*)

pyhrf.jde.jde_multi_sujets_alpha.**simulate_single_subject**(*output_dir*, *cdefs*, *var_subject_hrf*, *labels*, *labels_vol*, *v_noise*, *drift_coeff_var*, *drift_amplitude*, *hrf_group_level*, *alpha_var*, *dt=0.6*, *dsf=4*)

pyhrf.jde.jde_multi_sujets_alpha.**simulate_subjects**(*output_dir*, *snr_scenario='high_snr'*, *spatial_size='tiny'*, *hrf_group=array([0. , 0.00078678, 0.01381744, 0.0575847 , 0.13317542, 0.22304737, 0.30459629, 0.36130416, 0.38656651, 0.38221983, 0.35502768, 0.3133342 , 0.26480303, 0.21531699, 0.16874149, 0.12718515, 0.09146061, 0.06155495, 0.03701372, 0.01720819, 0.00149434, -0.01071142, -0.01991078, -0.02653129, -0.03094522, -0.03348818, -0.03447231, -0.03419243, -0.0329265 , -0.03093224, -0.02844234, -0.02565985, -0.02275507, -0.01986438, -0.01709107, -0.0145079 , -0.01216086, -0.01007359, -0.00825211, -0.00668921, -0.00536856, -0.00426813, 0.])*, *nbSubj=10*, *vars_hrfs=[0.0006, 0.0004, 9e-05, 2e-05, 1.5e-05, 2e-05, 0.0001, 3e-05, 7.5e-05, 3.2e-05]*, *vars_noise=[0.2, 0.5, 0.4, 0.8, 0.6, 2.1, 2.5, 3.1, 2.75, 7.3]*, *alpha_var=0.6*)

Simulate data for multiple subjects (5 subjects by default)

pyhrf.jde.models module

class pyhrf.jde.models.**ARN_BiG_BOLDSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

class pyhrf.jde.models.**BOLDGibbsSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *response_levels=<pyhrf.jde.nrl.bigaussian.NRLSampler object>*, *beta=<pyhrf.jde.beta.BetaSampler object>*, *noise_var=<pyhrf.jde.noise.NoiseVarianceSampler object>*, *hrf=<pyhrf.jde.hrf.HRFSampler object>*, *hrf_var=<pyhrf.jde.hrf.RHSampler object>*, *mixt_weights=<pyhrf.jde.nrl.bigaussian.MixtureWeightsSampler object>*, *mixt_params=<pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsS object>*, *scale=<pyhrf.jde.hrf.ScaleSampler object>*, *stop_crit_threshold=-1*, *stop_crit_from_start=False*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

buildSharedDataTree()

cleanObservables()

computeFit()

computePMStimInducedSignal()

compute_crit_diff(*old_vals*, *means=None*)

default_nb_its = 3000

getGlobalOutputs()

initGlobalObservables()

inputClass
alias of *WN_BiG_BOLDSamplerInput*

parametersComments = {'obs_hist_pace': 'See comment for samplesHistoryPaceSave.', 'sm

parametersToShow = ['nb_iterations', 'response_levels', 'hrf', 'hrf_var']

saveGlobalObservables(*it*)

stop_criterion(*it*)

updateGlobalObservables()

class pyhrf.jde.models.**BOLDGibbsSampler_AR**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *response_levels=<pyhrf.jde.nrl.ar.NRLARSampler object>*, *beta=<pyhrf.jde.beta.BetaSampler object>*, *noise_var=<pyhrf.jde.noise.NoiseVarianceARSampler object>*, *noise_arp=<pyhrf.jde.noise.NoiseARParamsSampler object>*, *hrf=<pyhrf.jde.hrf.HRFARSampler object>*, *hrf_var=<pyhrf.jde.hrf.RHSampler object>*, *mixt_weights=<pyhrf.jde.nrl.bigaussian.MixtureWeightsSamp object>*, *mixt_params=<pyhrf.jde.nrl.bigaussian.BiGaussMixturePara object>*, *scale=<pyhrf.jde.hrf.ScaleSampler object>*, *drift=<pyhrf.jde.drift.DriftARSampler object>*, *drift_var=<pyhrf.jde.drift.ETASampler object>*, *stop_crit_threshold=-1*, *stop_crit_from_start=False*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

buildSharedDataTree()

cleanObservables()

computeFit()

computePMStimInducedSignal()

compute_crit_diff(*old_vals*, *means=None*)

default_nb_its = 3000

getGlobalOutputs()

initGlobalObservables()

inputClass
alias of *ARN_BiG_BOLDSamplerInput*

parametersComments = {'obs_hist_pace': 'See comment for samplesHistoryPaceSave.', 'sm

parametersToShow = ['nb_iterations', 'response_levels', 'hrf', 'hrf_var']

saveGlobalObservables(*it*)

stop_criterion(*it*)

updateGlobalObservables()

class pyhrf.jde.models.**BOLDSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Class holding data needed by the sampler : BOLD time courses for each voxel, onsets and voxel topology. It also perform some precalculation such as the convolution matrix based on the onsests (L{stackX})

buildCosMat(*paramLFD*, *ny*)

buildOtherMatX()

buildParadigmConvolMatrix(*zc*, *estimDuration*, *availableDataIndex*, *parData*)

buildParadigmSingleCondMatrix(*zc*, *estimDuration*, *availableDataIndex*, *parData*)

buildPolyMat(*paramLFD*, *n*)

calcDt(*dtMin*)

chewUpOnsets(*dt*, *hrfZc*, *hrfDuration*)

cleanMem()

cleanPrecalculations()

makePrecalculations()

setLFDMat(*paramLFD*, *typeLFD*)
Build the low frequency basis from polynomial basis functions.

class pyhrf.jde.models.**BOLDSampler_Multi_SessInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)
Class holding data needed by the sampler : BOLD time courses for each voxel, onsets and voxel topology. It also perform some precalculation such as the convolution matrix based on the onsests (L{stackX}) —- Multi-sessions version

buildCosMat(*paramLFD*, *ny*)

buildOtherMatX()

buildParadigmConvolMatrix(*zc*, *estimDuration*, *availableDataIndex*, *parData*)

buildPolyMat(*paramLFD*, *n*)

calcDt(*dtMin*)

chewUpOnsets(*dt*, *hrfZc*, *hrfDuration*)

cleanMem()

cleanPrecalculations()

makePrecalculations()

setLFDMat(*paramLFD*, *typeLFD*)
Build the low frequency basis from polynomial basis functions.

class pyhrf.jde.models.**CallbackCritDiff**
Bases: *pyhrf.jde.samplerbase.GSDefaultCallbackHandler*

callback(*it*, *variables*, *samplerEngine*)

class pyhrf.jde.models.**Drift_BOLDGibbsSampler**(*nb_iterations=3000*, *obs_hist_pace=-1*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *response_levels=<pyhrf.jde.nrl.bigaussian_drift.NRL_Drift_Sample object>*, *beta=<pyhrf.jde.beta.BetaSampler object>*, *noise_var=<pyhrf.jde.noise.NoiseVariance_Drift_Sample object>*, *hrf=<pyhrf.jde.hrf.HRF_Drift_Sampler object>*, *hrf_var=<pyhrf.jde.hrf.RHSampler object>*, *mixt_weights=<pyhrf.jde.nrl.bigaussian.MixtureWeightsS object>*, *mixt_params=<pyhrf.jde.nrl.bigaussian.BiGaussMixture object>*, *scale=<pyhrf.jde.hrf.ScaleSampler object>*, *drift=<pyhrf.jde.drift.DriftSampler object>*, *drift_var=<pyhrf.jde.drift.ETASampler object>*, *stop_crit_threshold=-1*, *stop_crit_from_start=False*, *check_final_value=None*)
Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

computeFit()

default_nb_its = 3000

inputClass
alias of *WN_BiG_Drift_BOLDSamplerInput*

parametersToShow = ['nb_iterations', 'response_levels', 'hrf', 'hrf_var']

class pyhrf.jde.models.**Hab_WN_BiG_BOLDSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.WN_BiG_BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

class pyhrf.jde.models.**WN_BiG_BOLDSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

class pyhrf.jde.models.**WN_BiG_Drift_BOLDSamplerInput**(*data*, *dt*, *typeLFD*, *paramLFD*, *hrfZc*, *hrfDuration*)

Bases: *pyhrf.jde.models.BOLDSamplerInput*

cleanPrecalculations()

makePrecalculations()

class pyhrf.jde.models.**W_BOLDGibbsSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *response_levels=<pyhrf.jde.nrl.bigaussian.NRLSamplerWithRe object>*, *beta=<pyhrf.jde.beta.BetaSampler object>*, *noise_var=<pyhrf.jde.noise.NoiseVarianceSampler object>*, *hrf=<pyhrf.jde.hrf.HRFSamplerWithRelVar object>*, *hrf_var=<pyhrf.jde.hrf.RHSampler object>*, *mixt_weights=<pyhrf.jde.nrl.bigaussian.MixtureWeightsSample object>*, *mixt_params=<pyhrf.jde.nrl.bigaussian.BiGaussMixtureParam object>*, *scale=<pyhrf.jde.hrf.ScaleSampler object>*, *relevantVariable=<pyhrf.jde.wsampler.WSampler object>*, *stop_crit_threshold=-1*, *stop_crit_from_start=False*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

default_nb_its = 3000

inputClass
alias of *WN_BiG_BOLDSamplerInput*

parametersToShow = ['nb_iterations', 'response_levels', 'hrf', 'hrf_var']

class pyhrf.jde.models.**W_Drift_BOLDGibbsSampler**(*nb_iterations=3000*, *obs_hist_pace=-1.0*, *glob_obs_hist_pace=-1*, *smpl_hist_pace=-1.0*, *burnin=0.3*, *callback=<pyhrf.jde.samplerbase.GSDefaultCallbackHandler object>*, *response_levels=<pyhrf.jde.nrl.bigaussian_drift.NRL_Drift_Sam object>*, *beta=<pyhrf.jde.beta.BetaSampler object>*, *noise_var=<pyhrf.jde.noise.NoiseVariance_Drift_Sampler object>*, *hrf=<pyhrf.jde.hrf.HRF_Drift_SamplerWithRelVar object>*, *hrf_var=<pyhrf.jde.hrf.RHSampler object>*, *mixt_weights=<pyhrf.jde.nrl.bigaussian.MixtureWeightsSample object>*, *mixt_params=<pyhrf.jde.nrl.bigaussian.BiGaussMixtureParam object>*, *scale=<pyhrf.jde.hrf.ScaleSampler object>*, *condion_relevance=<pyhrf.jde.wsampler.W_Drift_Sampler object>*, *drift=<pyhrf.jde.drift.DriftSamplerWithRelVar object>*, *drift_var=<pyhrf.jde.drift.ETASampler object>*, *stop_crit_threshold=-1*, *stop_crit_from_start=False*, *check_final_value=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSampler*

default_nb_its = 3000

inputClass

alias of *WN_BiG_Drift_BOLDSamplerInput*

parametersToShow = ['nb_iterations', 'response_levels', 'hrf', 'hrf_var']

pyhrf.jde.models.**computePl**(*drift*, *varP*, *dest=None*)

pyhrf.jde.models.**computeSumjaXh**(*nrl*, *matXh*, *dest=None*)

pyhrf.jde.models.**computeXh**(*hrf*, *varX*, *dest=None*)

pyhrf.jde.models.**computeYBar**(*varMBY*, *varPl*, *dest=None*)

pyhrf.jde.models.**computeYTilde**(*sumj_aXh*, *varMBY*, *dest=None*)

pyhrf.jde.models.**computeYTilde_Pl**(*sumj_aXh*, *yBar*, *dest=None*)

pyhrf.jde.models.**computehXQXh**(*hrf*, *matXQX*, *dest=None*)

pyhrf.jde.models.**permutation**(*x*)

Randomly permute a sequence, or return a permuted range.

If *x* is a multi-dimensional array, it is only shuffled along its first index.

Parameters **x** (*int or array_like*) – If *x* is an integer, randomly permute `np.arange(x)`. If *x* is an array, make a copy and shuffle the elements randomly.

Returns **out** – Permuted sequence or array range.

Return type *ndarray*

Examples

```
>>> np.random.permutation(10)
array([1, 7, 4, 3, 0, 9, 2, 5, 8, 6])
```

```
>>> np.random.permutation([1, 4, 9, 12, 15])
array([15,  1,  9,  4, 12])
```

```
>>> arr = np.arange(9).reshape((3, 3))
>>> np.random.permutation(arr)
array([[6, 7, 8],
       [0, 1, 2],
       [3, 4, 5]])
```

pyhrf.jde.models.**rand**(*d0*, *d1*, ..., *dn*)

Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.

Returns **out** – Random values.

Return type ndarray, shape `(d0, d1, ..., dn)`

See also:

`random()`

Notes

This is a convenience function. If you want an interface that takes a shape-tuple as the first argument, refer to np.random.random_sample .

Examples

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

pyhrf.jde.models.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A (d0, d1, ..., dn)-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.jde.models.**simulate_bold**(*output_dir=None*, *noise_scenario='high_snr'*, *spatial_size='tiny'*, *normalize_hrf=True*)

pyhrf.jde.noise module

class pyhrf.jde.noise.**NoiseARParamsSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

MH_ARsampling_gauss_proposal(*sig2*, *M*)

MH_ARsampling_optim(*A*, *reps*, *M*)

P_SAMPLE_FLAG = 'sampleFlag'

P_USE_TRUE_VALUE = 'useTrueValue'

P_VAL_INI = 'initialValue'

checkAndSetInitValue(*variables*)

computeInvAutoCorrNoise(*ARp*)

defaultParameters = {'useTrueValue': False, 'initialValue': None, 'sampleFlag': Tru

finalizeSampling()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.noise.**NoiseVarianceARSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.jde.noise.NoiseVarianceSampler*

checkAndSetInitValue(*variables*)

computeVarYTilde(*varNrls*, *varXh*, *varMBYPl*)

finalizeSampling()

sampleNextInternal(*variables*)

class pyhrf.jde.noise.**NoiseVarianceSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

#TODO : comment

checkAndSetInitValue(*variables*)

computeMXhQXh(*h*, *varXQX*)

compute_aaXhQXhi(*aa*, *i*)

finalizeSampling()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

sampleNextInternal_bak(*variables*)

class pyhrf.jde.noise.**NoiseVarianceSamplerWithRelVar**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.jde.noise.NoiseVarianceSampler*

computeWW(*w*, *destww*)

compute_aawwXhQXhi(*ww*, *aa*, *i*)

finalizeSampling()

sampleNextInternal(*variables*)

class pyhrf.jde.noise.**NoiseVariance_Drift_Sampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

checkAndSetInitValue(*variables*)

linkToData(*dataInput*)

sampleNextInternal(*variables*)

class pyhrf.jde.noise.**NoiseVariancewithHabSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*)

Bases: *pyhrf.jde.noise.NoiseVarianceSampler*

#TODO : Sampling procedure for noise variance parameters (white noise) #in case of habituation modeling wrt magnitude

finalizeSampling()

sampleNextInternal(*variables*)

pyhrf.jde.samplerbase module

exception pyhrf.jde.samplerbase.**DuplicateVariableException**(*vName*, *v*)
Bases: exceptions.Exception

class pyhrf.jde.samplerbase.**GSDefaultCallbackHandler**
Bases: *pyhrf.xmlio.Initable*

Class handling default action after Gibbs Sampling step (nothing). Should be inherited to define more specialized actions (such as plotting and reporting).

callback(*it*, *vars*, *samplerEngine*)
Execute action to be made after each Gibbs Sampling step (here : nothing). Should be overriden to define more specialized actions. @param it: the number of iterations elapsed in the current sampling process. @param samplerEngine: the parent gibbs sampler object @param vars: variables envolved in the sampling process (list of C{GibbsSamplerVariable} whose index is defined in L{samplerEngine})

class pyhrf.jde.samplerbase.**GSPrintCallbackHandler**(*pace*)
Bases: *pyhrf.jde.samplerbase.GSDefaultCallbackHandler*

Class defining behaviour after each Gibbs Sampling step : printing reports to stdout.

callback(*it*, *variables*, *samplerEngine*)

class pyhrf.jde.samplerbase.**GibbsSampler**(*variables*, *nbIt*, *smplHistoryPace=-1*, *obsHistoryPace=-1*, *nbSweeps=None*, *callbackObj=None*, *randomSeed=None*, *globalObsHistoryPace=-1*, *check_ftval=None*, *output_fit=False*)
Generic class of a Gibbs sampler with gathers common operations for any gibbs sampling: variable initialisation, observables updates (posterior mean), outputs ...

computeFit()

finalizeSampling()

getFitAxes()

getGlobalOutputs()

getOutputs()

getShortProfile()

getTinyProfile()

get_variable(*label*)

initGlobalObservables()

iterate_sampling()

linkToData(*dataInput*)

regVarsInPipeline()

runSampling(*atomData=None*)
Launch a complete sampling process by calling the function L{GibbsSamplerVariable.sampleNext()} of each variable. Call the callback function after each iteration. Measure time elapsed and store it in L{tSamplinOnly} and L{analysis_duration}

saveGlobalObservables(*it*)

set_nb_iterations(*n*)

stop_criterion(*it*)

updateGlobalObservables()

class pyhrf.jde.samplerbase.**GibbsSamplerVariable**(*name*, *valIni=None*, *trueVal=None*, *sampleFlag=1*, *useTrueValue=False*, *axes_names=None*, *axes_domains=None*, *value_label='value'*)

checkAndSetInitValue(*variables*)

check_final_value()

chooseSampleNext(*flag*)

cleanObservables()

finalizeSampling()

getMean()
Wip ... Compute mean over MCMC iterations within the window defined by itStart, itEnd and pace. By default itStart is set to 'nbSweeps' and itEnd to the last iteration.

getMeanHistory()

getOutputs()

get_accuracy(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)
Return the accuray of the estimate *fv*, compared to the true value *tv*

Output: axes_names (list of str), accuracy (numpy array of booleans)

get_final_summary()

get_final_value()

get_string_value(*v*)

get_summary()

get_true_value()

get_variable(*label*)
Return a sibling GibbsSamplerVariable

initObservables()

linkToData()

manageMapping(*cuboid*)

manageMappingInit(*shape*, *axes_names*)

record_trajectories(*it*)

registerNbIterations(*nbIt*)

roiMapped()

sampleNextAlt(*variables*)
Define the behaviour of the variable at each sampling step when its sampling is not activated.

sampleNextInternal(*variables*)
Define the behaviour of the variable at each sampling step when its sampling is not activated. Must be overriden in child classes.

samplingWarmUp(*variables*)
Called before the launch of the main sampling loop by the sampler engine. Should be overriden and perform precalculations.

saveCurrentValue(*it*)

saveObservables(*it*)

setFinalValue()

setSamplerEngine(*sampler*)

track_obs_quantity(*q*, *name*, *axes_names=None*, *axes_domains=None*, *history_pace=None*)

track_sampled_quantity(*q*, *name*, *axes_names=None*, *axes_domains=None*, *history_pace=None*)

updateObsersables()

class pyhrf.jde.samplerbase.**Trajectory**(*variable*, *axes_names*, *axes_domains*, *history_pace*, *history_start*, *max_iterations*, *first_saved_iteration=-1*)
Keep track of a numpy array that is modified _inplace_ iteratively

get_last()
Return the last saved element

record(*iteration*)
Increment the history saving.

to_cuboid()
Pack the current trajectory in a xndarray

exception pyhrf.jde.samplerbase.**VariableTypeException**(*vClass*, *vName*, *v*)
Bases: exceptions.Exception

pyhrf.jde.wsampler module

class pyhrf.jde.wsampler.**WSampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *pr_sigmoid_slope=1.0*, *pr_sigmoid_thresh=0.0*)
Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

computeProbW1(*Qgj*, *gTQgj*, *rb*, *moyqj*, *t1*, *t2*, *mCAj*, *vCIj*, *vCAj*, *j*, *cardClassCAj*)
ProbW1 is the probability that condition is relevant It is a vecteur on length nbcond

computeVarXhtQ(*h*, *matXQ*)

computemoyq(*cardClassCA*, *nbVoxels*)
Compute mean of labels in ROI

finalizeSampling()

getOutputs()

initObservables()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

saveCurrentValue(*it*)

saveObservables(*it*)

threshold_W(*meanW*, *thresh*)

updateObsersables()

class pyhrf.jde.wsampler.**W_Drift_Sampler**(*do_sampling=True*, *use_true_value=False*, *val_ini=None*, *pr_sigmoid_slope=1.0*, *pr_sigmoid_thresh=0.0*)

Bases: *pyhrf.xmlio.Initable*, *pyhrf.jde.samplerbase.GibbsSamplerVariable*

CLASSES = array([0, 1])

CLASS_NAMES = ['inactiv', 'activ']

L_CA = 1

L_CI = 0

checkAndSetInitValue(*variables*)

computeProbW1(*gj*, *gTgj*, *rb*, *t1*, *t2*, *mCAj*, *vCIj*, *vCAj*, *j*, *cardClassCAj*)
ProbW1 is the probability that condition is relevant It is a vecteur on length nbcond

computemoyq(*cardClassCA*, *nbVoxels*)
Compute mean of labels in ROI

finalizeSampling()

getOutputs()

initObservables()

linkToData(*dataInput*)

sampleNextInternal(*variables*)

saveCurrentValue(*it*)

saveObservables(*it*)

threshold_W(*meanW*, *thresh*)

updateObsersables()

1.3 pyhrf.sandbox package

1.3.1 Submodules

pyhrf.sandbox.data_parser module

class pyhrf.sandbox.data_parser.**StructuredDataParser**(*directory_labels*, *allowed_subdirectories*, *directory_forgers=None*, *file_forgers=None*, *root_path=None*)

get_file_blocs(*file_defs*, ***subdirs*)

get_files(*file_def*, ***subdirs*)

set_root(*root*)

pyhrf.sandbox.data_parser.**apply_to_dict**(*d*, *f*)

pyhrf.sandbox.data_parser.**check_subdirs**(*path*, *labels*, *tree*, *not_found=None*)

pyhrf.sandbox.data_parser.**forge_nrl_files**(*conditions*)
construct list of nrl files from list of conditions

pyhrf.sandbox.data_parser.**safe_init**(*x*, *default*)

pyhrf.sandbox.data_parser.**safe_list**(*l*)

pyhrf.sandbox.data_parser.**same**(*x*)
identity function

pyhrf.sandbox.data_parser.**unformat_nrl_file**(*file_nrl*)

pyhrf.sandbox.func_BMA_consensus_clustering module

pyhrf.sandbox.func_BMA_consensus_clustering.**BMA_consensus_cluster_parallel**(*cfg*, *remote_path*, *remote_BOLD_fn*, *remote_mask_fn*, *Y*, *nifti_masker*, *num_vox*, *K_clus*, *K_clusters*, *parc*, *alpha*, *prop*, *nbItRFIR*, *onsets*, *durations*, *output_sub_parc*, *rescale=True*, *averg_bold=False*)
Performs all steps for one clustering case (Kclus given, number l of the parcellation given) remote_path: path on the cluster, where results will be stored

pyhrf.sandbox.func_BMA_consensus_clustering.**compute_consensus_clusters_parallel**(*K_clus*, *consensus_matrices*, *clust-count_matrices*, *total-count_matrices*, *num_voxels*, *remote_mask_fn*, *clusters_consensi*)

pyhrf.sandbox.make_parcellation module

Performs parcellation to a list of subjects

Directory structure:

- subject:
  - preprocessed_data –> GM+WM mask, functional data, normalised tissue masks
  - t_maps –> T-maps previously computed with GLM (nipy, SALMA)
  - parcellation –> Output

pyhrf.sandbox.make_parcellation.**make_mask**(*mask*, *volume*, *mask_file*)

pyhrf.sandbox.make_parcellation.**make_parcellation**(*subject*, *dest_dir='parcellation'*, *roi_mask_file=None*)

Perform a functional parcellation from input fmri data

Return: parcellation file name (str)

pyhrf.sandbox.parcellation module

Hierarchical Agglomerative Clustering

These routines perform some hierarchical agglomerative clustering of some input data. Currently, only Ward's algorithm is implemented.

Authors : Vincent Michel, Bertrand Thirion, Alexandre Gramfort, Gael Varoquaux Modified: Aina Frau License: BSD 3 clause

class pyhrf.sandbox.parcellation.**AgglomerationTransform**
Bases: object

class pyhrf.sandbox.parcellation.**BaseEstimator**
Bases: object

class pyhrf.sandbox.parcellation.**ClusterMixin**
Bases: object

pyhrf.sandbox.parcellation.**FWHM**(*Y*)

pyhrf.sandbox.parcellation.**GLM_method**(*name*, *data0*, *ncond*, *dt=0.5*, *time_length=25.0*, *ndelays=0*)

pyhrf.sandbox.parcellation.**Memory**(**args*, ***kwargs*)

class pyhrf.sandbox.parcellation.**Ward**(*n_clusters=2*, *memory=None*, *connectivity=None*, *copy=True*, *n_components=None*, *compute_full_tree='auto'*, *dist_type='uward'*, *cov_type='spherical'*, *save_history=False*)

Bases: *pyhrf.sandbox.parcellation.BaseEstimator*, *pyhrf.sandbox.parcellation.ClusterMixin*

Ward hierarchical clustering: constructs a tree and cuts it.

Parameters

- **n_clusters** (*int or ndarray*) – The number of clusters to find.
- **connectivity** (*sparse matrix.*) – Connectivity matrix. Defines for each sample the neighboring samples following a given structure of the data. Default is None, i.e, the hierarchical clustering algorithm is unstructured.
- **memory** (*Instance of joblib.Memory or str*) – Used to cache the output of the computation of the tree. By default, no caching is done. If a string is given, it is the path to the caching directory.
- **copy** (*bool*) – Copy the connectivity matrix or work inplace.
- **n_components** (*int (optional)*) – The number of connected components in the graph defined by the connectivity matrix. If not set, it is estimated.
- **compute_full_tree** (bool or *auto* (optional)) – Stop early the construction of the tree at n_clusters. This is useful to decrease computation time if the number of clusters is not small compared to the number of samples. This option is useful only when specifying a connectivity matrix. Note also that when varying the number of cluster and using caching, it may be advantageous to compute the full tree.

children_
array-like, shape = [n_nodes, 2] – List of the children of each nodes. Leaves of the tree do not appear.

labels_
array [n_samples] – cluster labels for each point

n_leaves_
int – Number of leaves in the hierarchical tree.

n_components_
sparse matrix. – The estimated number of connected components in the graph.

fit(*X*, *var=None*, *act=None*, *var_ini=None*, *act_ini=None*)
Fit the hierarchical clustering on the data

Parameters **X** (*array-like, shape = [n_samples, n_features]*) – The samples a.k.a. observations.

Returns

Return type self

class pyhrf.sandbox.parcellation.**WardAgglomeration**(*n_clusters=2*, *memory=None*, *connectivity=None*, *copy=True*, *n_components=None*, *compute_full_tree='auto'*, *dist_type='uward'*, *cov_type='spherical'*, *save_history=False*)

Bases: *pyhrf.sandbox.parcellation.AgglomerationTransform*, *pyhrf.sandbox.parcellation.Ward*

Feature agglomeration based on Ward hierarchical clustering

Parameters

- **n_clusters** (*int or ndarray*) – The number of clusters.
- **connectivity** (*sparse matrix*) – connectivity matrix. Defines for each feature the neighboring features following a given structure of the data. Default is None, i.e, the hierarchical agglomeration algorithm is unstructured.
- **memory** (*Instance of joblib.Memory or str*) – Used to cache the output of the computation of the tree. By default, no caching is done. If a string is given, it is the path to the caching directory.
- **copy** (*bool*) – Copy the connectivity matrix or work inplace.
- **n_components** (*int (optional)*) – The number of connected components in the graph defined by the connectivity matrix. If not set, it is estimated.
- **compute_full_tree** (bool or *auto* (optional)) – Stop early the construction of the tree at n_clusters. This is useful to decrease computation time if the number of clusters is not small compared to the number of samples. This option is useful only when specifying a connectivity matrix. Note also that when varying the number of cluster and using caching, it may be advantageous to compute the full tree.
- **variance** (*array with variances of all samples (default: None)*) – Injected in the calculation of the inertia.
- **activation** (*level of activation detected (default: None)*) – Used to weight voxels depending on level of activation in inertia computation. A non-active voxel will not estimate the HRF correctly, so features will not be correct either.

children_
array-like, shape = [n_nodes, 2] – List of the children of each nodes. Leaves of the tree do not appear.

labels_
array [n_samples] – cluster labels for each point

n_leaves_
int – Number of leaves in the hierarchical tree.

fit(*X*, *y=None*, ***params*)
Fit the hierarchical clustering on the data

Parameters **X** (*array-like, shape = [n_samples, n_features]*) – The data

Returns

Return type self

pyhrf.sandbox.parcellation.**align_parcellation**(*p1*, *p2*, *mask=None*)
Align two parcellation p1 and p2 as the minimum number of positions to remove in order to obtain equal partitions. :returns: (p2 aligned to p1)

pyhrf.sandbox.parcellation.**assert_parcellation_equal**(*p1*, *p2*, *mask=None*, *tol=0*, *tol_pos=None*)

pyhrf.sandbox.parcellation.**calculate_uncertainty**(*dm*, *g*)

pyhrf.sandbox.parcellation.**compute_fwhm**(*F*, *dt*, *a=0*)

pyhrf.sandbox.parcellation.**compute_hrf**(*method*, *my_glm*, *can*, *ndelays*, *i*)

pyhrf.sandbox.parcellation.**compute_mixt_dist**(*features*, *alphas*, *coord_row*, *coord_col*, *cluster_masks*, *moments*, *cov_type*, *res*)

Within one given territory: bi-Gaussian mixture model with known posterior weights:

$$phi \sum_i \lambda_i N(mu_i, v_i) p(q_j = i | phi_j) \text{is an input (alphas)}$$

Estimation: $\lambda_1 = 1 - \lambda_0$ is the mean of posterior weights. mu_i is estimated by weighted sample mean and v_i is estimated by weighted sample variance.

Parameters

- **features** (*np.array((nsamples, nfeatures), float)*) – the feature to parcellate
- **alphas** (*np.array(nsamples, float)*) – confidence levels on the features -> identified to posterior weights of class activating in the GMM fit
- **coord_row** (*list of int*) – row candidates for merging
- **coord_col** (*list of int*) – col candidates for merging
- **cluster_masks** –
- **moments** –
- **res** –

pyhrf.sandbox.parcellation.**compute_mixt_dist_skgmm**(*features*, *alphas*, *coord_row*, *coord_col*, *cluster_masks*, *moments*, *cov_type*, *res*)

pyhrf.sandbox.parcellation.**compute_uward_dist**(*m_1*, *m_2*, *coord_row*, *coord_col*, *variance*, *actlev*, *res*)

Function computing Ward distance: inertia = !!!!0

Parameters

- **m_1,m_2,coord_row,coord_col** (*clusters' parameters*) –
- **variance** (*uncertainty*) –
- **actlev** (*activation level*) –

Returns

- **res** (*Ward distance*)
- **Modified** (*Aina Frau*)

pyhrf.sandbox.parcellation.**compute_uward_dist2**(*m_1*, *features*, *alphas*, *coord_row*, *coord_col*, *cluster_masks*, *res*)

Function computing Ward distance: In this case we are using the model-based definition to compute the inertia

Parameters

- **m_1,m_2,coord_row,coord_col** (*clusters' parameters*) –
- **variance** (*uncertainty*) –
- **actlev** (*activation level*) –

Returns

- **res** (*Ward distance*)
- **Modified** (*Aina Frau*)

pyhrf.sandbox.parcellation.**feature_extraction**(*fmri_data*, *method*, *dt=0.5*, *time_length=25.0*, *ncond=1*)

fmri_data (pyhrf.core.FmriData): single ROI fMRI data

pyhrf.sandbox.parcellation.**generate_features**(*parcellation*, *act_labels*, *feat_levels*, *noise_var=0.0*)

Generate noisy features with different levels across positions depending on parcellation and activation clusters.

Parameters

- **parcellation** (*np.ndarray of integers in [1, nb_parcels]*) – the input parcellation
- **act_labels** (*binary np.ndarray*) – define which positions are active (1) and non-active (0)
- **feat_levels** (*dict of (int : (array((n_features,), float), array(n_features), float)) -> (dict of (parcel_idx : (feat_lvl_inact, feat_lvl_act))*) – map a parcel labels to feature levels in non-active and active-pos. Eg: *{1: ([1., .5], [10., 15])}* indicates that features in parcel 1 have values [1., .5] in non-active positions (2 features per position) and value 10. in active-positions
- **noise_var** (*float>0*) – variance of additive Gaussian noise

Returns The simulated the features.

Return type np.array((n_positions, n_features), float)

pyhrf.sandbox.parcellation.**hc_get_heads**(*parents*, *copy=True*)

Return the heads of the forest, as defined by parents :param parents: :type parents: array of integers :param The parent structure defining the forest (ensemble of trees): :param copy: :type copy: boolean :param If copy is False, the input 'parents' array is modified inplace:

Returns

- **heads** (*array of integers of same shape as parents*)
- *The indices in the 'parents' of the tree heads*

pyhrf.sandbox.parcellation.**hrf_canonical_derivatives**(*tr*, *oversampling=2.0*, *time_length=25.0*)

pyhrf.sandbox.parcellation.**informedGMM**(*features*, *alphas*)

Given a set of features, parameters (mu, v, lambda), and alphas: updates the parameters WARNING: only works for nb features = 1

pyhrf.sandbox.parcellation.**informedGMM_MV**(*fm*, *am*, *cov_type='spherical'*)

Given a set of multivariate features, parameters (mu, v, lambda), and alphas: fit a GMM where posterior weights are known (alphas)

pyhrf.sandbox.parcellation.**loglikelihood_computation**(*fm*, *mu0*, *v0*, *mu1*, *v1*, *a*)

pyhrf.sandbox.parcellation.**mixtp_to_str**(*mp*)

pyhrf.sandbox.parcellation.**norm2_bc**(*a*, *b*)

broadcast the computation of ||a-b||^2 where size(a) = (m,n), size(b) = n

pyhrf.sandbox.parcellation.**parcellation_hemodynamics**(*fmri_data*, *feature_extraction_method*, *parcellation_method*, *nb_clusters*)

Perform a hemodynamic-driven parcellation on masked fMRI data

Parameters

- **fmri_data** (–) – input fMRI data
- **feature_extraction_method** (–) – one of 'glm_hderiv', 'glm_hdisp' ...
- **parcellation_method** (–) – one of 'spatial_ward', 'spatial_ward_uncertainty', ...

Returns parcellation array (numpy array of integers) with flatten spatial axes

Examples #TODO

pyhrf.sandbox.parcellation.**render_ward_tree**(*tree*, *fig_fn*, *leave_colors=None*)

pyhrf.sandbox.parcellation.**represent_features**(*features*, *labels*, *ampl*, *territories*, *t*, *fn*)

Generate chart with features represented.

Parameters

- **features** (–) – features to be represented
- **labels** (–) – territories
- **ampl** (–) – amplitude of the positions

Returns

the size of the spots depends on ampl, and the color on labels

Return type features represented in 2D

pyhrf.sandbox.parcellation.**spatial_ward**(*features*, *graph*, *nb_clusters=0*)

pyhrf.sandbox.parcellation.**spatial_ward_sk**(*features*, *graph*, *nb_clusters=0*)

pyhrf.sandbox.parcellation.**spatial_ward_with_uncertainty**(*features*, *graph*, *variance*, *activation*, *var_ini=None*, *act_ini=None*, *nb_clusters=0*, *dist_type='uward'*, *cov_type='spherical'*, *save_history=False*)

Parcellation the given features with the spatial Ward algorithm, taking into account uncertainty on features (variance) and activation level:

- the greater the variance of a given sample, the lower its importance in the distance.
- the lower the activation level of a given sample, the lower its distance to any other sample.

Parameters

- **feature** (*np.ndarray*) – observations to parcellate - size: (nsamples, nfeatures)
- **graph** (*list of (list of PositionIndex)*) – spatial dependency between positions
- **variance** (*np.ndarray*) – variance of features - size: (nsamples, nfeatures)
- **activation** (*np.ndarray*) – activation level associated with observation.
- **size** (*int*) – n samples
- **var_ini** –
- **act_ini** –
- **nb_clusters** (*int*) – number of clusters
- **dist_type** (*str*) – *ward* | *mixt*

pyhrf.sandbox.parcellation.**squared_error**(*n*, *m*)

pyhrf.sandbox.parcellation.**ward_tree**(*X*, *connectivity=None*, *n_components=None*, *copy=True*, *n_clusters=None*, *var=None*, *act=None*, *var_ini=None*, *act_ini=None*, *dist_type='uward'*, *cov_type='spherical'*, *save_history=False*)

Ward clustering based on a Feature matrix.

The inertia matrix uses a Heapq-based representation.

This is the structured version, that takes into account a some topological structure between samples.

Parameters

- **X** (*array of shape (n_samples, n_features)*) – feature matrix representing n_samples samples to be clustered
- **connectivity** (*sparse matrix.*) – connectivity matrix. Defines for each sample the neigbhoring samples following a given structure of the data. The matrix is assumed to be symmetric and only the upper triangular half is used. Default is None, i.e, the Ward algorithm is unstructured.
- **n_components** (*int (optional)*) – Number of connected components. If None the number of connected components is estimated from the connectivity matrix.
- **copy** (*bool (optional)*) – Make a copy of connectivity or work inplace. If connectivity is not of LIL type there will be a copy in any case.
- **n_clusters** (*int (optional)*) – Stop early the construction of the tree at n_clusters. This is useful to decrease computation time if the number of clusters is not small compared to the number of samples. In this case, the complete tree is not computed, thus the 'children' output is of limited use, and the 'parents' output should rather be used. This option is valid only when specifying a connectivity matrix.
- **dist_type** (*str -> uward | mixt*) –

Returns

- **children** (*2D array, shape (n_nodes, 2)*) – list of the children of each nodes. Leaves of the tree have empty list of children.
- **n_components** (*sparse matrix.*) – The number of connected components in the graph.
- **n_leaves** (*int*) – The number of leaves in the tree
- **parents** (*1D array, shape (n_nodes,) or None*) – The parent of each node. Only returned when a connectivity matrix is specified, elsewhere 'None' is returned.
- **Modified** (*Aina Frau*)

pyhrf.sandbox.parcellation.**ward_tree_save**(*tree*, *output_dir*, *mask*)

pyhrf.sandbox.physio module

pyhrf.sandbox.physio.**buildOrder1FiniteDiffMatrix_central**(*size*, *dt*)

returns a toeplitz matrix for central differences to correct for errors on the first and last points (due to the fact that there is no rf[-1] or rf[size] to average with):

- uses the last point to calcuate the first and vis-versa
- this is acceptable bc the rf is assumed to begin & end at steady state (thus the first and last points should both be zero)

pyhrf.sandbox.physio.**calc_linear_rfs**(*simu_brf*, *simu_prf*, *phy_params*, *dt*, *normalized_rfs=True*)

Calculate 'prf given brf' and 'brf given prf' based on the a linearization around steady state of the physiological model as described in Friston 2000.

Input:

- **simu_brf, simu_prf: brf and prf from the physiological simulation** from which you wish to calculate the respective prf and brf. Assumed to be of size (1,hrf.size)
- phy_params
- normalized_rfs: set to True if simu_hrfs are normalized

Output:

- calc_brf, calc_prf: np.arrays of shape (hrf.size, 1)
- q_linear, v_linear: q and v calculated according to the linearized model

Note: These calculations do not account for any rescaling between brf and prf. This means the input simu_brf, simu_prf should NOT be rescaled.

** Warning**:**

- this function assumes prf.size == brf.size and uses this to build D, I
- if making modifications: calc_brf, calc_prf have a truncation error (due to the finite difference matrix used) on the order of O(dt)^2. If for any reason a hack is later implemented to set the y-intecepts of brf_calc, prf_calc to zero by setting the first row of X4, X3 = 0, this will raise a singular matrix error in the calculation of calc_prf (due to X.I command), so this error is helpful in this case

pyhrf.sandbox.physio.**create_asl_from_stim_induced**(*bold_stim_induced_rescaled*, *perf_stim_induced*, *ctrl_tag_mat*, *dsf*, *perf_baseline*, *noise*, *drift=None*, *outliers=None*)

Downsample stim_induced signal according to downsampling factor 'dsf' and add noise and drift (nuisance signals) which has to be at downsampled temporal resolution.

pyhrf.sandbox.physio.**create_bold_from_hbr_and_cbv**(*physiological_params*, *hbr*, *cbv*)

Compute BOLD signal from HbR and blood volume variations obtained by a physiological model

pyhrf.sandbox.physio.**create_evoked_physio_signals**(*physiological_params*, *paradigm*, *neural_efficacies*, *dt*, *integration_step=0.05*)

Generate evoked hemodynamics signals by integrating a physiological model.

Parameters

- **physiological_params** (*dict (<pname (str)> : <pvalue (float)>)*) – parameters of the physiological model. In jde.sandbox.physio see PHY_PARAMS_FRISTON00, PHY_PARAMS_FMRII ...
- **paradigm** (pyhrf.paradigm.Paradigm) – the experimental paradigm
- **neural_efficacies** (*np.ndarray (nb_conditions, nb_voxels, float)*) – neural efficacies involved in flow inducing signal.
- **dt** (*float*) – temporal resolution of the output signals, in second
- **integration_step** (*float*) – time step used for integration, in second

Returns

All generated signals, indexes of the first axis correspond to:

- 0: flow inducing
- 1: inflow
- 2: blood volume
- 3: [HbR]

Return type np.array((nb_signals, nb_scans, nb_voxels), float)

pyhrf.sandbox.physio.**create_omega_prf**(*primary_brf*, *dt*, *physiological_params*)

pyhrf.sandbox.physio.**create_physio_brf**(*physiological_params*, *response_dt=0.5*, *response_duration=25.0*, *return_brf_q_v=False*)

Generate a BOLD response function by integrating a physiological model and setting its driving input signal to a single impulse.

Parameters

- **physiological_params** (–) – <pvalue (float)>)): parameters of the physiological model. In jde.sandbox.physio see PHY_PARAMS_FRISTON00, PHY_PARAMS_FMRII ...
- **response_dt** (–) – temporal resolution of the response, in second
- **response_duration** (–) – duration of the response, in second

Returns

- np.array(nb_time_coeffs, float) -> the BRF (normalized)
- also return brf_not_normalized, q, v when return_prf_q_v=True (for error checking of v and q generation in calc_hrfs)

pyhrf.sandbox.physio.**create_physio_prf**(*physiological_params*, *response_dt=0.5*, *response_duration=25.0*, *return_prf_q_v=False*)

Generate a perfusion response function by setting the input driving signal of the given physiological model with a single impulse.

Parameters

- **physiological_params** (–) – <pvalue (float)>)): parameters of the physiological model. In jde.sandbox.physio see PHY_PARAMS_FRISTON00, PHY_PARAMS_FMRII ...
- **response_dt** (–) – temporal resolution of the response, in second
- **response_duration** (–) – duration of the response, in second

Returns

- np.array(nb_time_coeffs, float) -> the PRF
- also return brf_not_normalized, q, v when return_prf_q_v=True (for error checking of v and q generation in calc_hrfs)

pyhrf.sandbox.physio.**create_tbg_neural_efficacies**(*physiological_params*, *condition_defs*, *labels*)

Create neural efficacies from a truncated bi-Gaussian mixture.

TODO: settle how to relate brls and prls to neural efficacies

Parameters

- **physiological_params** (*dict (<param_name> : <param_value>)*) – parameters of the physiological model
- **condition_defs** (*list of pyhrf.Condition*) – list of condition definitions. Each item should have the following fields (moments of the mixture):
  - m_act (0<=float<eff_max): mean of activating component
  - v_act (0<float): variance of activating component
  - v_inact (0<float): variance of non-activating component
- **labels** (*np.array((nb_cond, nb_vox), int)*) – binary activation states

Returns the generated neural efficacies

Return type np.array(np.array((nb_cond, nb_vox), float))

pyhrf.sandbox.physio.**linear_rf_operator**(*rf_size*, *phy_params*, *dt*, *calculating_brf=False*)

Calculates the linear operator A needed to convert brf to prf & vis-versa prf = (A^{-1})brf brf = (A)prf

Inputs:

- size of the prf and/or brf (assumed to be same)
- physiological parameters
- time resolution of data:
- if you wish to calculate brf (return A), or prf (return inverse of A)

Outputs:

- np.array of size (hrf_size,1) linear operator to convert hrfs

pyhrf.sandbox.physio.**phy_integrate_euler**(*phy_params*, *tstep*, *stim*, *epsilon*, *Y0=None*)

Integrate the ODFs of the physiological model with the Euler method.

TODO: should the output signals be rescaled wrt their value at rest?

Parameters

- **phy_params** (*dict (<param_name> : <param_value>)*) – parameters of the physiological model
- **tstep** (*float*) – time step of the integration, in seconds.
- **stim** (*np.array(nb_steps, float)*) – stimulation sequence with a temporal resolution equal to the time step of the integration
- **epsilon** (*float*) – neural efficacy
- **Y0** (*np.array(4, float) | None*) – initial values for the physiologica signals. If None: [0, 1, 1, 1.] s f_in q v

Returns

the integrated physiological signals, where indexes of the first axis correspond to:

- 0 : flow inducing
- 1 : inflow
- 2 : HbR
- 3 : blood volume

Return type np.array((4, nb_steps), float)

pyhrf.sandbox.physio.**plot_calc_hrf**(*hrf1_simu*, *hrf1_simu_name*, *hrf1_calc*, *hrf1_calc_name*, *hrf2_simu*, *hrf2_simu_name*, *dt*)

pyhrf.sandbox.physio.**rescale_bold_over_perf**(*bold_stim_induced*, *perf_stim_induced*, *bold_perf_ratio=5.0*)

pyhrf.sandbox.physio.**run_calc_linear_rfs**()

Choose physio parameters. Choose to generate *simu_rfs* from multiple or single stimulus.

TODO:

- figure out why there is an issue that *perf_stim_induced* is much greater than *bold_stim_induced*
- figure out why when *simu_brf*='*bold_stim_induced_rescaled*, *calc_brf* is so small it appears to be 0

pyhrf.sandbox.physio.**simulate_asl_full_physio**(*output_dir=None*, *noise_scenario='high_snr'*, *spatial_size='tiny'*)

Generate ASL data by integrating a physiological dynamical system.

Ags:

- **output_dir (str|None): path where to save outputs as nifti files.** If None: no output files
- noise_scenario ("high_snr"|"low_snr"): scenario defining the SNR
- **spatial_size ("tiny"|"normal")** [scenario for the size of the map]
  - "tiny" produces 2x2 maps
  - "normal" produces 20x20 maps

Result: dict (<item_label (str)> : <simulated_item (np.ndarray)>) -> a dictionary mapping names of simulated items to their values

WARNING: in this dict the 'bold' item is in fact the ASL signal. This name was used to be compatible with JDE which assumes that the functional time series is named "bold". TODO: rather use the more generic label 'fmri_signal'.

TODO: use magnetization model to properly simulate final ASL signal

pyhrf.sandbox.physio.**simulate_asl_phylin_prf**(*output_dir=None*, *noise_scenario='high_snr'*, *spatial_size='tiny'*)

Generate ASL data according to a LTI system, with canonical BRF and PRF = Omega.BRF.

Parameters

- **output_dir** (–) – path where to save outputs as nifti files. If None: no output files
- **noise_scenario** (–) – scenario defining the SNR
- **spatial_size** (–) – scenario for the size of the map - "tiny" produces 2x2 maps - "normal" produces 20x20 maps

Result: dict (<item_label (str)> : <simulated_item (np.ndarray)>) -> a dictionary mapping names of simulated items to their values

WARNING: in this dict the 'bold' item is in fact the ASL signal. This name was used to be compatible with JDE which assumes that the functional time series is named "bold". TODO: rather use the more generic label 'fmri_signal'.

pyhrf.sandbox.physio.**simulate_asl_physio_rfs**(*output_dir=None*, *noise_scenario='high_snr'*, *spatial_size='tiny'*, *v_noise=None*)

Generate ASL data according to a LTI system, with PRF and BRF generated from a physiological model.

Parameters

- **output_dir** (–) – path where to save outputs as nifti files. If None: no output files
- **noise_scenario** (–) – scenario defining the SNR
- **spatial_size** (–) – scenario for the size of the map - "tiny" produces 2x2 maps - "normal" produces 20x20 maps

Result: dict (<item_label (str)> : <simulated_item (np.ndarray)>) -> a dictionary mapping names of simulated items to their values

WARNING: in this dict the 'bold' item is in fact the ASL signal. This name was used to be compatible with JDE which assumes that the functional time series is named "bold". TODO: rather use the more generic label 'fmri_signal'.

pyhrf.sandbox.physio_params module

pyhrf.sandbox.physio_params.**buildOrder1FiniteDiffMatrix_central**(*size*, *dt*)

Returns a toeplitz matrix for central differences to correct for errors on the first and last points (due to the fact that there is no rf[-1] or rf[size] to average with):

- uses the last point to calculate the first and vise-versa
- this is acceptable bc the rf is assumed to begin & end at steady state (thus the first and last points should both be zero)

pyhrf.sandbox.physio_params.**calc_linear_rfs**(*simu_brf*, *simu_prf*, *phy_params*, *dt*, *normalized_rfs=True*)

Calculate 'prf given brf' and 'brf given prf' based on the a linearization around steady state of the physiological model as described in Friston 2000

Input:

- **simu_brf, simu_prf: brf and prf from the physiological simulation** from which you wish to calculate the respective prf and brf. Assumed to be of size (1,hrf.size)
- phy_params
- normalized_rfs: set to True if simu_hrfs are normalized

Output:

- calc_brf, calc_prf: np.arrays of shape (hrf.size, 1)
- q_linear, v_linear: q and v calculated according to the linearized model

Note: These calculations do not account for any rescaling between brf and prf. This means the input simu_brf, simu_prf should NOT be rescaled.

**** Warning****:**

- this function assumes prf.size == brf.size and uses this to build D, I
- if making modifications: calc_brf, calc_prf have a truncation error (due to the finite difference matrix used) on the order of O(dt)^2. If for any reason a hack is later implemented to set the y-intecepts of brf_calc, prf_calc to zero by setting the first row of X4, X3 = 0, this will raise a singular matrix error in the calculation of calc_prf (due to X.I command), so this error is helpful in this case

pyhrf.sandbox.physio_params.**create_bold_from_hbr_and_cbv**(*physiological_params*, *hbr*, *cbv*)

Compute BOLD signal from HbR and blood volume variations obtained by a physiological model

pyhrf.sandbox.physio_params.**create_evoked_physio_signals**(*physiological_params*, *paradigm*, *neural_efficacies*, *dt*, *integration_step=0.05*)

Generate evoked hemodynamics signals by integrating a physiological model.

Parameters

- **physiological_params** (*dict (<pname (str)> : <pvalue (float)>)*) – parameters of the physiological model. In jde.sandbox.physio see PHY_PARAMS_FRISTON00, PHY_PARAMS_FMRII
- **paradigm** (pyhrf.paradigm.Paradigm) – the experimental paradigm
- **neural_efficacies** (*np.ndarray (nb_conditions, nb_voxels, float)*) – neural efficacies involved in flow inducing signal.
- **dt** (*float*) – temporal resolution of the output signals, in second
- **integration_step** (*float*) – time step used for integration, in second

Returns

All generated signals, indexes of the first axis correspond to:

- 0: flow inducing
- 1: inflow
- 2: blood volume
- 3: [HbR]

Return type np.array((nb_signals, nb_scans, nb_voxels), float)

pyhrf.sandbox.physio_params.**create_k_parameters**(*physiological_params*)

Create field strength dependent parameters k1, k2, k3

pyhrf.sandbox.physio_params.**create_omega_prf**(*primary_brf*, *dt*, *phy_params*)

create prf from omega and brf

pyhrf.sandbox.physio_params.**create_physio_brf**(*physiological_params*, *response_dt=0.5*, *response_duration=25.0*, *return_brf_q_v=False*)

Generate a BOLD response function by integrating a physiological model and setting its driving input signal to a single impulse.

Parameters

- **physiological_params** (–) – <pvalue (float)>)): parameters of the physiological model. In jde.sandbox.physio see PHY_PARAMS_FRISTON00, PHY_PARAMS_FMRII...
- **response_dt** (–) – temporal resolution of the response, in second
- **response_duration** (–) – duration of the response, in second

Returns

- np.array(nb_time_coeffs, float) -> the BRF (normalized)

- also return brf_not_normalized, q, v when return_prf_q_v=True (for error checking of v and q generation in calc_hrfs)

pyhrf.sandbox.physio_params.**create_physio_prf**(*physiological_params*, *response_dt=0.5*, *response_duration=25.0*, *return_prf_q_v=False*)

Generate a perfusion response function by setting the input driving signal of the given physiological model with a single impulse.

Parameters

- **physiological_params** (–) – <pvalue (float)>)): parameters of the physiological model. In jde.sandbox.physio see PHY_PARAMS_FRISTON00, PHY_PARAMS_FMRII...
- **response_dt** (–) – temporal resolution of the response, in second
- **response_duration** (–) – duration of the response, in second

Returns

- np.array(nb_time_coeffs, float) -> the PRF
- also return brf_not_normalized, q, v when return_prf_q_v=True (for error checking of v and q generation in calc_hrfs)

pyhrf.sandbox.physio_params.**create_tbg_neural_efficacies**(*physiological_params*, *condition_defs*, *labels*)

Create neural efficacy from a truncated bi-Gaussian mixture.

Parameters

- **physiological_params** (*dict (<param_name> : <param_value>)*) – parameters of the physiological model
- **condition_defs** (*list of pyhrf.Condition*) – list of condition definitions. Each item should have the following fields (moments of the mixture):
  - m_act (0<=float<eff_max): mean of activating component
  - v_act (0<float): variance of activating component
  - v_inact (0<float): variance of non-activating component
- **labels** (*np.array((nb_cond, nb_vox), int)*) – binary activation states

Returns

- *np.array(np.array((nb_cond, nb_vox), float))* – the generated neural efficacies
- **TODO** (*settle how to relate brls and prls to neural efficacy*)

pyhrf.sandbox.physio_params.**linear_rf_operator**(*rf_size*, *phy_params*, *dt*, *calculating_brf=False*)

Calculates the linear operator A needed to convert brf to prf & vis-versa prf = (A^{-1})brf brf = (A)prf

Inputs:

- size of the prf and/or brf (assumed to be same)
- physiological parameters
- time resolution of data:
- if you wish to calculate brf (return A), or prf (return inverse of A)

Outputs:

- np.array of size (hrf_size,1) linear operator to convert hrfs

pyhrf.sandbox.physio_params.**phy_integrate_euler**(*phy_params*, *tstep*, *stim*, *epsilon*, *Y0=None*)

Integrate the ODFs of the physiological model with the Euler method.

Parameters

- **phy_params** (*dict (<param_name> : <param_value>)*) – parameters of the physiological model
- **tstep** (*float*) – time step of the integration, in seconds.
- **stim** (*np.array (nb_steps, float)*) – stimulation sequence with temporal resolution equal to the time step of the integration
- **epsilon** (*float*) – neural efficacy
- **Y0** (*np.array (4, float) | None*) – initial values for the physiological signals. If None: [0, 1, 1, 1.] s f_in q v

Returns

- *np.array((4, nb_steps), float)* – the integrated physiological signals, where indexes of the first axis correspond to:
  - 0 : flow inducing
  - 1 : inflow
  - 2 : HbR
  - 3 : blood volume
- **TODO** (*should the output signals be rescaled wrt their value at rest?*)

pyhrf.sandbox.stats module

class pyhrf.sandbox.stats.**GSVariable**(*name*, *initialization*, *do_sampling=True*, *axes_names=None*, *axes_domains=None*)

check_against_truth(*atol*, *rtol*, *inaccuracy_handling='print'*)

check_initialization_arg(*ia*)

enable_sampling(*flag=True*)

get_accuracy_against_truth(*abs_error*, *rel_error*, *fv*, *tv*, *atol*, *rtol*)

Return the accuray of the estimate *fv*, compared to the true value *tv*

get_custom_init()

Must return a numpy.ndarray. Consider initializing with a good guess so that sampling converges more quickly.

get_estim_value_for_check()

get_random_init()

Must return a random numpy.ndarray that will then be used as init value for sampling. For example, it can be a sample from the prior distribution. This function will also be used to test for the sensitivity to initialization.

get_true_value_for_check()

get_variable(*vname*)

get_variable_value(*vname*)
Short-hand to get variable among all those defined in the parent sampler

init_observables()

init_sampling()

reset()

sample()
Draw a sample conditionally to the current Gibbs Sampler state. Must return a numpy.ndarray.

Variables which have been registered in the parent GibbsSampler object can be retrieved via methods self.get_variable(var_name) and self.get_variable_value(var_name)

set_init_value()
Set the initial value of self.current_value, depending on the *initialization* scenario (random, custom, truth).

set_initialization(*init*)

set_outputs(*outputs*, *output_type='ndarray'*)

Parameters

- **outputs** (–) – dictionary to be updated with custom outputs.
- **output_type** (–) – 'ndarray' or 'cuboid'

Return: None

set_true_value(*true_value*)

track_obs_quantity(*name*, *quantity*, *history_pace=None*, *axes_names=None*, *axes_domains=None*)

track_sampled_quantity(*name*, *quantity*, *history_pace=None*, *axes_names=None*, *axes_domains=None*)

update_observables()
Update quantities after the burnin period

class pyhrf.sandbox.stats.**GibbsSampler**(*sampled_variables*, *nb_its_max*, *obs_pace=1*, *burnin=0.3*, *sample_hist_pace=-1*, *obs_hist_pace=-1*)

check_against_truth(*default_atol=0.1*, *default_rtol=0.1*, *var_specific_atol=None*, *var_specific_rtol=None*, *inaccuracy_handling='print'*)

get_outputs(*output_type='ndarray'*)
output_type : 'ndarray' or 'cuboid'

get_variable(*vname*)

get_variable_value(*vname*)

iterate_sampling()

reset()

Reset the Gibbs Sampler:

- remove all previous history of quantities (trajectories)
- call reset method of all variables

run()

set_initialization(*vname*, *init*)

set_true_value(*vname*, *true_value*)

set_true_values(*true_values*)

set_variable(*name*, *var*)

set_variables(*var_dict*)

stop_criterion(*iteration*)

track_obs_quantity(*name*, *q*, *history_pace=None*, *axes_names=None*, *axes_domains=None*)

track_sampled_quantity(*name*, *q*, *history_pace=None*, *axes_names=None*, *axes_domains=None*)

class pyhrf.sandbox.stats.**Trajectory**(*variable*, *history_pace*, *history_start*, *max_iterations*, *init_iteration=None*, *axes_names=None*, *axes_domains=None*)

Keep track of a numpy array that is modified _inplace_ iteratively TODO: when mature, should be moved to pyhrf.ndarray should replace pyhrf.jde.samplerbase.Trajectory

get_last()
Return the last saved element

to_cuboid()
Pack the current trajectory in a xndarray

update(*iteration*)
Record the current variable value

1.4 pyhrf.stats package

1.4.1 Submodules

pyhrf.stats.misc module

pyhrf.stats.misc.**acorr**(*x*, *maxlags=10*, *scale='var'*)

pyhrf.stats.misc.**compute_T_Pvalue**(*betas*, *stds_beta*, *mask_file*, *null_hyp=True*)
Compute Tvalues statistic and Pvalue based upon estimates and their standard deviation beta and std_beta for all voxels beta: shape (nb_vox, 1) std: shape (1) Assume null hypothesis if null_hyp is True

pyhrf.stats.misc.**compute_roc_labels**(*mlabels*, *true_labels*, *dthres=0.005*, *lab_ca=1*, *lab_ci=0*, *false_pos=2*, *false_neg=3*)

pyhrf.stats.misc.**compute_roc_labels_scikit**(*e_labels*, *true_labels*)

pyhrf.stats.misc.**cpt_ppm_a_apost**(*means*, *variances*, *props*, *alpha=0.05*)

pyhrf.stats.misc.**cpt_ppm_a_mcmc**(*samples*, *alpha=0.05*)
Compute a Posterior Probability Map (fixed alpha) from NRL MCMC samples. Expected shape of 'samples': (sample, voxel)

pyhrf.stats.misc.**cpt_ppm_a_norm**(*mean*, *variance*, *alpha=0.0*)
Compute a Posterior Probability Map (fixed alpha) by assuming a Gaussian distribution.

Parameters

- **mean** (*array_like*) – mean value(s) of the Gaussian distribution(s)
- **variance** (*array_like*) – variance(s) of the Gaussian distribution(s)

- **alpha** (*array_like, optional*) – quantile value(s) (default=0)

Returns **ppm** – Posterior Probability Map evaluated at alpha

Return type array_like

pyhrf.stats.misc.**cpt_ppm_g_apost**(*means*, *variances*, *props*, *gamma=0.0*)
Compute a Posterior Probability Map (fixed gamma) from posterior gaussian mixture components estimates. Expected shape of 'means', 'variances' and 'probs': (nb_classes, voxel)

pyhrf.stats.misc.**cpt_ppm_g_mcmc**(*samples*, *gamma=0.0*)
Compute a Posterior Probability Map (fixed gamma) from NRL MCMC samples. Expected shape of 'samples': (sample, voxel)

pyhrf.stats.misc.**cpt_ppm_g_norm**(*mean*, *variance*, *gamma=0.95*)
Compute a Posterior Probability Map (fixed gamma) by assuming a Gaussian distribution.

Parameters

- **mean** (*array_like*) – mean value(s) of the Gaussian distribution(s)
- **variance** (*array_like*) – variance(s) of the Gaussian distribution(s)
- **gamma** (*array_like, optional*) – upper tail probability (default=0.95)

Returns **ppm** – Posterior Probability Map corresponding to the upper tail probability gamma

Return type *ndarray* or scalar

pyhrf.stats.misc.**cumFreq**(*data*, *thres=None*)

pyhrf.stats.misc.**gm_cdf**(*x*, *means*, *variances*, *props*)
Compute the cumulative density function of gaussian mixture, ie: p(x<a) = sum_i Nc(mean_i, variance_i)

pyhrf.stats.misc.**gm_mean**(*means*, *variances*, *props*)

pyhrf.stats.misc.**gm_var**(*means*, *variances*, *props*)

pyhrf.stats.misc.**mark_wrong_labels**(*labels*, *true_labels*, *lab_ca=1*, *lab_ci=0*, *false_pos=2*, *false_neg=3*)

pyhrf.stats.misc.**threshold_labels**(*labels*, *thresh=None*, *act_class=1*)
Threshold input labels which are assumed being of shape (nb classes, nb conds, nb vox). If thresh is None then take the argmax over classes. Else use it on labels for activating class (act_class), suitable for the 2class case only.

pyhrf.stats.random module

class pyhrf.stats.random.**BetaGenerator**(*mean=0.5*, *var=0.1*)
Bases: *pyhrf.stats.random.RandomGenerator*

Class encapsulating the beta random generator of numpy

generate(*size*)

class pyhrf.stats.random.**GammaGenerator**(*mean=1.0*, *var=1.0*)
Bases: *pyhrf.stats.random.RandomGenerator*

Class encapsulating the gamma random generator of numpy

generate(*size*)

class pyhrf.stats.random.**GaussianGenerator**(*mean=0.0*, *var=1.0*)
Bases: *pyhrf.stats.random.RandomGenerator*

Class encapsulating the gaussian random generator of numpy

generate(*size*)

class pyhrf.stats.random.**IndependentMixtureLaw**(*states*, *generators*)
Class handling the generation of values following an indenpendent mixture law. Requires the prior generator of label values.

generate()
Generate realisations of the mixture law.

class pyhrf.stats.random.**LogNormalGenerator**(*meanLogN=1.0*, *varLogN=1.0*)
Bases: *pyhrf.stats.random.RandomGenerator*

Class encapsulating the log normal generator of numpy

generate(*size*)

class pyhrf.stats.random.**RandomGenerator**
B Abstract class to ensure the definition of the function generate.

generate(*size*)

class pyhrf.stats.random.**UniformGenerator**(*minV=0.0*, *maxV=1.0*)
Bases: *pyhrf.stats.random.RandomGenerator*

Class encapsulating the random generator

generate(*size*)

class pyhrf.stats.random.**ZeroGenerator**
Bases: *pyhrf.stats.random.RandomGenerator*

Class encapsulating the null distribution !!!!!!!!!

generate(*size*)

pyhrf.stats.random.**gm_sample**(*means*, *variances*, *props*, *n=1*)

pyhrf.stats.random.**rand**(*d0*, *d1*, ..., *dn*)
Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.

Returns **out** – Random values.

Return type ndarray, shape `(d0, d1, ..., dn)`

See also:

`random()`

Notes

This is a convenience function. If you want an interface that takes a shape-tuple as the first argument, refer to np.random.random_sample .

Examples

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

pyhrf.stats.random.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

`sigma * np.random.randn(...) + mu`

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.stats.random.**rpnorm**(*n*, *m*, *s*)

Random numbers from the positive normal distribution. rpnorm(n,m,s) is a vector of length n with random entries, generated from a positive normal distribution with mean m and standard deviation s.

Original matlab code from: (c) Vincent Mazet, 06/2005 Centre de Recherche en Automatique de Nancy, France vincent.mazet@cran.uhp-nancy.fr

Reference: V. Mazet, D. Brie, J. Idier, 'Simulation of Positive Normal Variables using several Proposal Distributions', IEEE Workshop Statistical Signal Processing 2005, july 17-20 2005, Bordeaux, France.

Adapted by Thomas VINCENT: thomas.vincent@cea.fr

pyhrf.stats.random.**truncRandn**(*size*, *mu=0.0*, *sigma=1.0*, *a=0.0*, *b=inf*)

1.5 pyhrf.test package

pyhrf.test.**rand**(*d0*, *d1*, ..., *dn*)

Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.

Returns **out** – Random values.

Return type ndarray, shape `(d0, d1, ..., dn)`

See also:

`random()`

Notes

This is a convenience function. If you want an interface that takes a shape-tuple as the first argument, refer to np.random.random_sample .

Examples

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

pyhrf.test.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

`random.standard_normal()` Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

`sigma * np.random.randn(...) + mu`

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

1.5.1 Submodules

pyhrf.test.analysertest module

class pyhrf.test.analysertest.**BetaEstimESTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_obs_2Dfield_MAP()
Test estimation of beta with an observed 2D field. Partition function estimation method : extrapolation scheme. Use the MAP on p(beta|label).

test_obs_3Dfield_MAP()
Test estimation of beta with an observed field: a small 3D case. Partition function estimation method : extrapolation scheme. Use the MAP on p(beta|label).

test_obs_field_ML()
Test estimation of beta with an observed field: a small 2D case. Partition function estimation method : extrapolation scheme. Use the ML on p(label|beta). PF estimation: Onsager

pyhrf.test.boldsynthTest module

class pyhrf.test.boldsynthTest.**FieldFuncsTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_count_homo_cliques()

test_count_homo_cliques1()

test_count_homo_cliques2()

test_potts_gibbs()

test_swendsenwang()

class pyhrf.test.boldsynthTest.**Mapper1DTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test3D()

testIncompleteMapping()

testIrregularMapping()

pyhrf.test.boldsynthTest.**rand**(*d0*, *d1*, ..., *dn*)

Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.

Returns **out** – Random values.

Return type ndarray, shape `(d0, d1, ..., dn)`

See also:

`random()`

Notes

This is a convenience function. If you want an interface that takes a shape-tuple as the first argument, refer to np.random.random_sample .

Examples

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

pyhrf.test.boldsynthTest.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.test.commandTest module

class pyhrf.test.commandTest.**MiscCommandTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_gls_default()

test_gls_recursive()

test_gls_recursive_group()
test pyhrf_gls command in recursive mode with file groups specified by a regular expression

class pyhrf.test.commandTest.**TreatmentCommandTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

makeQuietOutputs(*xmlFile*)

setDummyInputData(*xmlFile*)

setSimulationData(*xmlFile*, *simu_file*)

setUp()

tearDown()

testDetectEstimDefault()

testHrfEstim()

test_WNSGGMS()

test_WNSGGMS_surf_cmd()

test_buildcfg_contrasts()

test_buildcfg_jde_loc_vol_default()

test_buildcfg_jde_locav_surf_default()

test_buildcfg_jde_locav_vol_default()

pyhrf.test.core_test module

class pyhrf.test.core_test.**FMRIDataTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_from_vol_ui_default()

test_multisession_simu()

pyhrf.test.graphtest module

class pyhrf.test.graphtest.**GraphTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_bfs()

test_from_lattice1()
Test default behaviour of graph_from_lattice, in 2D

test_from_lattice2()
Test graph_from_lattice in 3D with another kernel mask

test_from_lattice_toro()
Test graph_from_lattice, 2D toroidal case

test_from_lattice_toro_huge()
Test graph_from_lattice, 2D toroidal case

test_from_mesh()

test_graph_is_sane()

test_parcels_to_graphs()

test_pyhrf_extract_cc_vol()

test_split_vol_cc_2D()

test_split_vol_cc_3D()

test_sub_graph()

pyhrf.test.iotest module

class pyhrf.test.iotest.**DataLoadTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_frmi_vol()
Test volumic data loading

test_paradigm_csv()

test_paradigm_csv2()

test_paradigm_csv3()

test_paradigm_csv4()

class pyhrf.test.iotest.**FileHandlingTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_split4DVol()

test_split_ext()

class pyhrf.test.iotest.**GiftiTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_load_fmri_surf_data()
Test surfacic data loading

test_read_default_real_data_tiny()

test_read_tex_gii_label()

test_write_tex_gii_2D_float()

test_write_tex_gii_float()

test_write_tex_gii_labels()

test_write_tex_gii_time_series()

class pyhrf.test.iotest.**NiftiTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_process_history_extension()

class pyhrf.test.iotest.**RxCopyTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

assert_file_exists(*fn*, *test_exists=True*)

setUp()

tearDown()

test_advanced()

test_basic()

test_callback()

test_dry()

test_duplicates_targets()

test_missing_tags_dest_basename()

test_missing_tags_dest_folder()

test_replacement()

test_with_subfolders()

class pyhrf.test.iotest.**SPMIOTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_load_regnames_SPM12()

test_load_regnames_SPM5()

test_load_regnames_SPM8()

class pyhrf.test.iotest.**xndarrayIOTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_save_nii_3D()

test_save_nii_4D()

test_save_nii_multi()

pyhrf.test.jdetest module

class pyhrf.test.jdetest.**ASLPhysioTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_default_jde_small_simulation()
Test ASL Physio sampler on small simulation with small nb of iterations. Estimation accuracy is not tested.

class pyhrf.test.jdetest.**ASLTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_default_jde_small_simulation()
Test ASL sampler on small simulation with small nb of iterations. Estimation accuracy is not tested.

test_simulation()

class pyhrf.test.jdetest.**JDETest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

testDefaultWithOutputs()

test_parcellation()

test_surface_treatment()

class pyhrf.test.jdetest.**MultiSessTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_default_jde_small_simulation()
Test JDE multi-sessions sampler on small simulation with small nb of iterations. Estimation accuracy is not tested.

class pyhrf.test.jdetest.**PartitionFunctionTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

testExtrapolation2C()

pyhrf.test.jdetest.**test_suite**()

pyhrf.test.rfir_test module

pyhrf.test.seppotest module

pyhrf.test.seppotest.**foo**(*o*)

pyhrf.test.statsTest module

class pyhrf.test.statsTest.**PPMTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_gm_cdf()

test_gm_sample_active()

test_gm_sample_half()

test_gm_sample_inactive()

test_ppm_a_mcmc()

test_ppm_a_norm()

test_ppm_g_apost()

test_ppm_g_mcmc()

test_ppm_g_norm()

class pyhrf.test.statsTest.**RPNormTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testSimple()

pyhrf.test.test module

pyhrf.test.test.**rand**(*d0*, *d1*, ..., *dn*)
Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.

Returns **out** – Random values.

Return type ndarray, shape `(d0, d1, ..., dn)`

See also:

`random()`

Notes

This is a convenience function. If you want an interface that takes a shape-tuple as the first argument, refer to np.random.random_sample .

Examples

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

pyhrf.test.test.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

`sigma * np.random.randn(...) + mu`

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.test.test_glm module

class pyhrf.test.test_glm.**NipyGLMTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

makeQuietOutputs(*xmlFile*)

setUp()

tearDown()

test_command_line()

test_fir_glm()

test_glm_contrasts()

test_glm_default_real_data()

test_glm_with_files()

pyhrf.test.test_glm.**test_suite**()

pyhrf.test.test_jde_multi_subj module

class pyhrf.test.test_jde_multi_subj.**MultiSubjTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_quick()
Test running of JDE multi subject (do not test result accuracy)

pyhrf.test.test_jde_multi_subj.**simulate_subjects**(*output_dir*, *snr_scenario='high_snr'*, *spatial_size='tiny'*, *hrf_group=None*, *nb_subjects=15*, *vhrf=0.1*, *vhrf_group=0.1*)
Simulate daata for multiple subjects (5 subjects by default)

pyhrf.test.test_jde_vem_asl module

class pyhrf.test.test_jde_vem_asl.**VEMASLTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_jdevemanalyser()
Test BOLD VEM sampler on small simulation with small nb of iterations. Estimation accuracy is not tested.

pyhrf.test.test_jde_vem_bold module

class pyhrf.test.test_jde_vem_bold.**VEMBOLDTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_jdevemanalyser()
Test BOLD VEM sampler on small simulation with small nb of iterations. Estimation accuracy is not tested.

test_vem_bold_constrained(**args*, ***kwargs*)
Test BOLD VEM constraint function. Estimation accuracy is not tested.

test_vem_bold_constrained_python(**args*, ***kwargs*)
Test BOLD VEM constraint function. Estimation accuracy is not tested.

pyhrf.test.test_jde_vem_tools module

class pyhrf.test.test_jde_vem_tools.**VEMToolsTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_PolyMat()

test_buildFiniteDiffMatrix()

test_computeFit()

test_compute_mat_X2()

test_create_conditions()

test_create_neighbours()

test_entropyA()

test_entropyH()

test_entropyZ()

test_expectA()

test_expectH()

test_expectZ()

test_free_energy()
Test of vem tool to compute free energy

test_gradient()

test_matrix()

test_max_L()

test_max_beta()

test_max_mu_sigma()

test_max_sigmaH()

test_max_sigmaH_prior()

test_max_sigma_noise()

test_maximum()

test_normpdf()

test_polyFit()

pyhrf.test.test_jde_vem_tools_UtilsC module

class pyhrf.test.test_jde_vem_tools_UtilsC.**VEMToolsTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_expectA()

test_expectH()

test_expectZ()

test_max_L()

test_max_sigma_noise()

pyhrf.test.test_jde_vem_tools_asl module

class pyhrf.test.test_jde_vem_tools_asl.**VEMToolsTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_PolyMat()

test_buildFiniteDiffMatrix()

test_computeFit()

test_compute_mat_X2()

test_entropyA()

test_entropyH()

test_entropyZ()

test_matrix()

test_max_sigmaH()

test_max_sigmaH_prior()

test_maximum()

test_normpdf()

test_polyFit()

pyhrf.test.test_ndarray module

class pyhrf.test.test_ndarray.**TestHtml**(*methodName='runTest'*)
Bases: unittest.case.TestCase

assert_html_equal(*html*, *expected*)

setUp()

tearDown()

test_plot()

test_table_header()

test_txt_1d_col_axes_only()

test_txt_1d_row_axes_only()

test_txt_tooltip()

class pyhrf.test.test_ndarray.**xndarrayTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_cartesian_eval()
Test the multiple evaluations of a function that returns a xndarray, over the cartesian products of given arguments.

test_combine_domains()

test_equality()

test_expansion()

test_explode()

test_fill()
TODO

test_flatten_and_expand()

test_init()

test_merge()
TODO !!!

test_operations()

test_save_as_gii()

test_save_as_nii()

test_set_orientation()

test_split()

test_squeeze()

test_stack()

test_sub_cuboid()

test_sub_cuboid_with_float_domain()

test_to_latex_1d()

test_to_latex_3d()

test_to_latex_3d_col_align()

test_to_latex_3d_hide_name_style()

test_to_latex_3d_inner_axes()

test_to_latex_3d_join_style()

test_tree_to_xndarray()

test_unstack_2D()

test_unstack_empty_inner_axes()

test_xmapping()

test_xmapping_inconsistent_domain()

test_xmapping_inconsistent_mapping_value()

pyhrf.test.test_paradigm module

class pyhrf.test.test_paradigm.**ParadigmTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_merge_onsets()

test_onsets_loc_av()

test_to_nipy_Block()
Test event-related paradigm

test_to_nipy_Block_2sess()
Test event-related paradigm

test_to_nipy_ER()
Test event-related paradigm

test_to_nipy_ER_2sess()
Test event-related paradigm

test_to_spm_mat_1st_level()

pyhrf.test.test_parallel module

class pyhrf.test.test_parallel.**ParallelTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_remote_map_local()

test_remote_map_local_cartesian_args()

test_remote_map_serial()

pyhrf.test.test_parallel.**foo**(*a*, *b*)

pyhrf.test.test_parallel.**foo_raise**(*a*, *b*)

pyhrf.test.test_parcellation module

class pyhrf.test.test_parcellation.**CmdParcellationTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_voronoi_with_seeds()

test_ward_spatial_cmd(**args*, ***kwargs*)

test_ward_spatial_real_data(**args*, ***kwargs*)

class pyhrf.test.test_parcellation.**MeasureTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_intersection_matrix()

test_parcellation_distance(**args*, ***kwargs*)

class pyhrf.test.test_parcellation.**ParcellationMethodTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_ward_spatial_scikit(**args*, ***kwargs*)

test_ward_spatial_scikit_with_mask(**args*, ***kwargs*)

class pyhrf.test.test_parcellation.**SpatialTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_balanced_parcellation()
Test if balanced partitioning returns parcels with almost equal sizes (tolerance=1) on a 3D rectangular mask

test_split_parcel()

test_voronoi_parcellation()

pyhrf.test.test_plot module

class pyhrf.test.test_plot.**PlotCommandTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_plot_func_slice_func_only()

test_plot_func_slice_func_only_multiple_slices()

test_plot_func_slice_func_roi()

test_plot_func_slice_func_roi_anat()

test_plot_func_slice_func_roi_anat_multiple_slices()

class pyhrf.test.test_plot.**PlotFunctionsTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_plot_cuboid1d_as_curve()

test_plot_cuboid1d_as_image()

test_plot_cuboid2d_as_image()

test_plot_cuboid_as_curve()

pyhrf.test.test_rfir module

class pyhrf.test.test_rfir.**RFIRTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test the Regularized FIR (RFIR)-based methods implemented in pyhrf.rfir

setUp()

tearDown()

test_rfir_on_small_simulation()
Check if pyhrf.rfir runs properly and that returned outputs contains the expected items

pyhrf.test.test_sampler module

class pyhrf.test.test_sampler.**GibbsTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_var_tracking()

class pyhrf.test.test_sampler.**TrajectoryTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_basic()

test_burnin()

pyhrf.test.test_sandbox_physio module

class pyhrf.test.test_sandbox_physio.**SimulationTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_create_evoked_physio_signal()

test_create_physio_brf()

test_create_physio_prf()

test_create_tbg_neural_efficacies()
Test the generation of neural efficacies from a truncated bi-Gaussian mixture

test_phy_integrate_euler()

test_simulate_asl_full_physio()

test_simulate_asl_full_physio_outputs()

test_simulate_asl_physio_rfs()

pyhrf.test.test_treatment module

class pyhrf.test.test_treatment.**CmdInputTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test extraction of information from the command line to create an FmriTreatment

setUp()

tearDown()

test_spm12_option_parse()
Test parsing of option "-s SPM.mat" (SPM12)

test_spm5_option_parse()
Test parsing of option "-s SPM.mat" (SPM5)

test_spm8_option_parse()
Test parsing of option "-s SPM.mat" (SPM8)

class pyhrf.test.test_treatment.**TreatmentTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_default_jde_cmd_parallel_local()

test_default_treatment()

test_default_treatment_parallel_LAN()

test_default_treatment_parallel_cluster()

test_default_treatment_parallel_local()

test_jde_estim_from_treatment_pck()

test_parallel_local()

test_pickle_treatment()

test_remote_dir_writable()

test_sub_treatment()

pyhrf.test.test_xml module

class pyhrf.test.test_xml.**A**(*p=1*, *c='a'*)
Bases: *pyhrf.xmlio.Initable*

class pyhrf.test.test_xml.**B**(*obj_t=array([5])*)
Bases: *pyhrf.xmlio.Initable*

classmethod from_stuff(*a=2*, *b=5*)

class pyhrf.test.test_xml.**BaseTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testNumpy()

test_basic_types()

test_bool()

test_list_of_int()

test_list_of_misc()

test_list_of_str()

test_ordered_dict()

test_tuple_of_misc()

class pyhrf.test.test_xml.**C**
Bases: *pyhrf.xmlio.Initable*

class pyhrf.test.test_xml.**ChildClass**(*p_child=2*)
Bases: *pyhrf.xmlio.Initable*

class pyhrf.test.test_xml.**D**(*p=2*)
Bases: *pyhrf.xmlio.Initable*

class pyhrf.test.test_xml.**InitableTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_JDEMCMCAnalyzerXML()

test_JDEMCMCAnalyzer_Uinode_bijection()

test_TreatmentXML()

test_bijection_from_classmethod_init()

test_bijection_from_init()

test_bijection_from_init_no_arg()

test_classmethod_init()

test_init()

test_pickle_classmethod()

test_xml_from_classmethod_init()

test_xml_from_init()

class pyhrf.test.test_xml.**T**(*param_a=1*)
Bases: *pyhrf.xmlio.Initable*

classmethod from_param_c(*param_c=array([56])*)

class pyhrf.test.test_xml.**TestXML**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_simple_bijection()

class pyhrf.test.test_xml.**TopClass**(*p_top='1'*)
Bases: *pyhrf.xmlio.Initable*

class pyhrf.test.test_xml.**XMLableTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testDynamicParamsHierachic()

testDynamicParamsSingleClass()

test_set_init_param()

pyhrf.test.test_xml.**create_t**()

pyhrf.test.toolsTest module

class pyhrf.test.toolsTest.**CachedEvalTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_code_digest()

test_simple()

test_simple_args()

test_slow_func()

class pyhrf.test.toolsTest.**CartesianTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testCartesianBasic()

test_cartesian_apply()

test_cartesian_apply_parallel()

class pyhrf.test.toolsTest.**CropTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testBasic()

class pyhrf.test.toolsTest.**DiagBlockTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testAll2D()

testFrom1D()

testFromNdarray()

testRepFrom1D()

testRepFrom2D()

testRepFromBlocks()

class pyhrf.test.toolsTest.**DictToStringTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testBasic()

testOnHierachicDict()

testOnNumpyArray()

testOnSpmMat()

class pyhrf.test.toolsTest.**GeometryTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_convex_hull()

test_distance()

class pyhrf.test.toolsTest.**MiscTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_decorator_do_if_file_exist()

test_decorator_do_if_file_exist2()

test_decorator_do_if_file_exist_force()

class pyhrf.test.toolsTest.**PeelVolumeTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testPeel()

class pyhrf.test.toolsTest.**PipelineTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

testBadDepTreeInit()

testGoodDepTreeInit()

testRepr()

test_cached()

test_func_default_args()

test_multiple_output_values()

class pyhrf.test.toolsTest.**ResampleTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testLargerTargetGrid()

testResampleToGrid()

class pyhrf.test.toolsTest.**TableStringTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test1Darray()

test2Darray()

test2Darray_latex()

test3Darray()

test4Darray()

pyhrf.test.toolsTest.**computeB**(*a*, *e*)

pyhrf.test.toolsTest.**computeC**(*a*)

pyhrf.test.toolsTest.**computeD**(*f*, *b*, *c*)

pyhrf.test.toolsTest.**computeF**(*g*, *e*)

pyhrf.test.toolsTest.**computeJ**(*i*, *l*)

pyhrf.test.toolsTest.**computeK**(*j*)

pyhrf.test.toolsTest.**computeL**(*k*)

pyhrf.test.toolsTest.**foo**(*a*, *b*, *c=1*, *d=2*)

pyhrf.test.toolsTest.**foo_a**(*c=1*)

pyhrf.test.toolsTest.**foo_default_arg**(*a*, *d=1*)

pyhrf.test.toolsTest.**foo_func**(*a*, *b*)

pyhrf.test.toolsTest.**foo_multiple_returns**(*e*)

pyhrf.test.toolsTest.**slow_func**(*a*, *b*)

class pyhrf.test.toolsTest.**treeToolsTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_get_leaf()

test_rearrange()

test_set_leaf()

test_stack_trees()

test_walk_branches()

1.6 pyhrf.tools package

1.6.1 Submodules

pyhrf.tools.aexpression module

class pyhrf.tools.aexpression.**ArithmeticExpression**(*expression*, ***variables*)
Bases: object

Mathematical Expression Evaluator class. You can set the expression member, set the functions, variables and then call evaluate() function that will return you the result of the mathematical expression given as a string.

addDefaultFunctions()
Add the following Python functions to be used in a mathemtical expression: acos asin atan atan2 ceil cos cosh degrees exp fabs floor fmod frexp hypot ldexp log log10 modf pow radians sin sinh sqrt tan tanh

addDefaultVariables()
Add e and pi to the list of defined variables.

call_if_func(*x*)

check()

evaluate()
Evaluate the mathematical expression given as a string in the expression member variable.

functions = None
Dictionary of variables that can be used in the expression.

getFunctionNames()
Return a List of defined function names in sorted order.

getVariableNames()
Return a List of defined variables names in sorted order.

setVariable(*name*, *value*)
Define the value of a variable defined by name

variables = None

exception pyhrf.tools.aexpression.**ArithmeticExpressionNameError**
Bases: exceptions.Exception

exception pyhrf.tools.aexpression.**ArithmeticExpressionSyntaxError**
Bases: exceptions.Exception

pyhrf.tools.backports module

class pyhrf.tools.backports.**OrderedDict**(**args*, ***kwds*)
Bases: dict

Dictionary that remembers insertion order

clear() → None. Remove all items from od.

copy() → a shallow copy of od

classmethod fromkeys(*S*[, *v*]) → New ordered dictionary with keys from S
and values equal to v (which defaults to None).

items() → list of (key, value) pairs in od

iteritems()
od.iteritems -> an iterator over the (key, value) items in od

iterkeys() → an iterator over the keys in od

itervalues()
od.itervalues -> an iterator over the values in od

keys() → list of keys in od

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), return and remove a (key, value) pair.
Pairs are returned in LIFO order if last is true or FIFO order if false.

setdefault(*k*[, *d*]) → od.get(k,d), also set od[k]=d if k not in od

update(*E*, ***F*) → None. Update od from dict/iterable E and F.
If E is a dict instance, does: for k in E: od[k] = E[k] If E has a .keys() method, does: for k in E.keys(): od[k] = E[k] Or if E is an iterable of items, does: for k, v in E: od[k] = v In either case, this is followed by: for k, v in F.items(): od[k] = v

values() → list of values in od

viewitems() → a set-like object providing a view on od's items

viewkeys() → a set-like object providing a view on od's keys

viewvalues() → an object providing a view on od's values

pyhrf.tools.cpus module

This module implements function to detect the number of allowed cpus available to the python process.

This is licensed under the CC-BY-SA 3.0 and written by Bakuriu (https://stackoverflow.com/users/510937/bakuriu), ohspite (https://stackoverflow.com/users/891129/ohspite), and Philipp Hagemeister (https://stackoverflow.com/users/35070/phihag). See https://stackoverflow.com/a/1006301

pyhrf.tools.cpus.**available_cpu_count**()
Number of available virtual or physical CPUs on this system, i.e. user/real as output by time(1) when called with an optimally scaling userspace-only program

pyhrf.tools.message module

This package provide a mean to print colored message to a standard terminal if color is available else message are print in black and white mode. If stdout is redirected in a file or piped to an other program, the output is made black and white to avoid issus with strange caracters that defined colors in terminals. Remember that all messages are print to stdout.

This package exists in 3 places, for some very good arguments : datamind.tools.message soma.wip.message pyhrf.tools.message

To use these functionalities, play with 'msg' instance. Here, some classical uses :

msg.info('something cool happend'): msg.error(self, 'too bad, an error'): msg.warning(self, 'something strange but not fatal'): msg.write_list(('no color', ('color in red', 'red'))): msg.write('simple colored write function') msg.string('string to colored string')

class pyhrf.tools.message.**Message**
Bases: object

sys = <module 'sys' (built-in)>

class pyhrf.tools.message.**MessageColor**
Bases: object

classmethod error(*msg*)

haveColor()

classmethod info(*msg*)

classmethod string(*msg*, *color='back'*)

classmethod warning(*msg*)

classmethod write(*msg*, *color='back'*)

classmethod write_list(*msg_list*)

class pyhrf.tools.message.**MessageNoColor**
Bases: *pyhrf.tools.message.MessageColor*

classmethod error(*msg*)

haveColor()

classmethod info(*msg*)

classmethod string(*msg*, *color='back'*)

classmethod warning(*msg*)

class pyhrf.tools.message.**NoMessage**
Bases: object

error(*msg*)

haveColor()

info(*msg*)

string(*msg*, *color='back'*)

warning(*msg*)

write(*msg*, *color='back'*)

write_list(*msg_list*)

pyhrf.tools.misc module

class pyhrf.tools.misc.**AnsiColorizer**
Format strings with an ANSI escape sequence to encode color

BEGINC = '\x1b['

COLORS = {'purple': '95', 'blue': '94', 'green': '92', 'red': '91', 'yellow': '93

ENDC = '\x1b[0m'

disable()

enable()

no_tty_check()

tty_check()

pyhrf.tools.misc.**Extract_TTP_whM_from_group**(*hrfs_pck_file*, *dt*, *model*, *Path_data*, *acq*)
Extract TTP and whM from a group of hrfs whose values are saved in a .pck (size nb_subjects * nb_coeff_hrf)

pyhrf.tools.misc.**Extract_TTP_whM_hrf**(*hrf*, *dt*)
Extract TTP and whM from an hrf

pyhrf.tools.misc.**PPMcalculus_jde**(*threshold_value*, *apost_mean_activ_fn*, *apost_var_activ_fn*, *apost_mean_inactiv_fn*, *apost_var_inactiv_fn*, *labels_activ_fn*, *labels_inactiv_fn*, *nrls_fn*, *mask_file*, *null_hyp=True*)
Function to calculate the probability that the nrl in voxel j, condition m, is superior to a given hreshold_value Computation for all voxels Compute Tvalue according to null hypothesis

class pyhrf.tools.misc.**PickleableStaticMethod**(*fn*, *cls=None*)
Bases: object

class pyhrf.tools.misc.**Pipeline**(*quantities*)

THE_ROOT = 0

add_root(*label*)

checkGraph()
Check the rightness of the builded graph (acyclicity, uniqueness and no short-circuits)

detectCyclity(*viewedNodes*)

detectShortCircuit(*curRoot*, *curDepth*, *depths*)
Recursive method which detects and corrects short-circuits

get_func(*f*)

get_value(*label*)
Return the value associated with 'label'

get_values()
Return all computed values. Perform a full update if not done yet.

init_dependencies(*quantities*)

removeShortCircuits(*label*, *depths*)

reportChange(*rootLabel*)
Trigger update of the sub graph starting at the given root

reprAllDeps()
Build a string representing the while graph : a concatenation of representations of all nodes (see reprDep)

reprDep(*label*)
Build a string representing all dependencies and dependers of the variable *label*. The returned string is in the form

```
       label
depee1 <-
depee2 <-
       -> deper1
       -> deper2
```

resolve()

save_graph_plot(*image_filename*, *images=None*)

setDepths(*label*, *depths*, *curDepth*)

update_all()

update_quantity(*label*, *updated*)

update_subgraph(*root*)

pyhrf.tools.misc.**add_prefix**(*fn*, *prefix*)
Add a prefix at the beginning of a file name.

```
>>> add_prefix('./my_file.txt', 'my_prefix_')
'./my_prefix_my_file.txt'
```

pyhrf.tools.misc.**add_suffix**(*fn*, *suffix*)
Add a suffix before file extension.

```
>>> add_suffix('./my_file.txt', '_my_suffix')
'./my_file_my_suffix.txt'
```

pyhrf.tools.misc.**apply_to_leaves**(*tree*, *func*, *funcArgs=None*, *funcKwargs=None*)
Apply function 'func' to all leaves in given 'tree' and return a new tree.

pyhrf.tools.misc.**array_summary**(*a*, *precision=4*)

pyhrf.tools.misc.**assert_file_exists**(*fn*)

pyhrf.tools.misc.**assert_path_not_in_src**(*p*)

pyhrf.tools.misc.**attrs_to_string**(*attrs*)

pyhrf.tools.misc.**buildPolyMat**(*paramLFD*, *n*, *dt*)

pyhrf.tools.misc.**cache_exists**(*func*, *args=None*, *prefix=None*, *path='./'*, *digest_code=False*)

pyhrf.tools.misc.**cache_filename**(*func*, *args=None*, *prefix=None*, *path='./'*, *digest_code=False*)

pyhrf.tools.misc.**cached_eval**(*func*, *args=None*, *new=False*, *save=True*, *prefix=None*, *path='./'*, *return_file=False*, *digest_code=False*, *gzip_mode='cmd'*)

pyhrf.tools.misc.**calc_nc2D**(*a*, *b*)

pyhrf.tools.misc.**cartesian**(**sequences*)

Generate the "cartesian product" of all 'sequences'. Each member of the product is a list containing an element taken from each original sequence.

Note: equivalent to itertools.product, which is at least 2x faster !!

pyhrf.tools.misc.**cartesian_apply**(*varying_args*, *func*, *fixed_args=None*, *nb_parallel_procs=1*, *joblib_verbose=0*)

Apply function *func* iteratively on the cartesian product of *varying_args* with fixed args *fixed_args*. Produce a tree (nested dicts) mapping arg values to the corresponding evaluation of function *func*

Parameters

- **varying_args** (OrderedDict) – a dictionary mapping argument names to a list of values. The *Orderdict* is used to keep track of argument order in the result. *WARNING:* all argument values must be hashable
- **func** (*function*) – the function to be applied on the cartesian product of given arguments
- **fixed_args** (*dict*) – arguments that are fixed (do not enter cartesian product)

Returns **nested dicts** – where each node is an argument value from varying args and each leaf is the result of the evaluation of the function. The order to the tree levels corresponds the order in the input *OrderedDict* of varying arguments.

Return type *tree*

Examples

```
>>> from pyhrf.tools import cartesian_apply
>>> from pyhrf.tools.backports import OrderedDict
>>> def foo(a,b,c): return a + b + c
>>> v_args = OrderedDict( [('a',[0,1]), ('b',[1,2])] )
>>> fixed_args = {'c': 3}
>>> cartesian_apply(v_args, foo, fixed_args) == { 0 : { 1:4, 2:5}, 1 : { 1:5, 2:6}
↪ }
True
```

pyhrf.tools.misc.**cartesian_combine_args**(*varying_args*, *fixed_args=None*)

Construct the cartesian product of varying_args and append fixed_args to it.

Parameters

- **varying_args** – Specify varying arguments as a dict mapping arg names to iterables of arg values. e.g:

```
{'my_arg1' : ['a','b','c'],
 'my_arg2' : [2, 5, 10]}
```

- **fixed_args** – Specify constant arguments as a dict mapping arg names to arg values. e.g:

```
{ 'my_arg3' : ['fixed_value'] }
```

Examples

```
>>> from pyhrf.tools import cartesian_combine_args
>>> vargs = {'my_arg1' : ['a','b','c'],'my_arg2' : [2, 5, 10],}
>>> fargs = { 'my_arg3' : 'fixed_value' }
>>> res = cartesian_combine_args(vargs, fargs)
>>> res ==          [{'my_arg1': 'a', 'my_arg2': 2, 'my_arg3': 'fixed_value'},
...  {'my_arg1': 'b', 'my_arg2': 2, 'my_arg3': 'fixed_value'},
...  {'my_arg1': 'c', 'my_arg2': 2, 'my_arg3': 'fixed_value'},
...  {'my_arg1': 'a', 'my_arg2': 5, 'my_arg3': 'fixed_value'},
...  {'my_arg1': 'b', 'my_arg2': 5, 'my_arg3': 'fixed_value'},
...  {'my_arg1': 'c', 'my_arg2': 5, 'my_arg3': 'fixed_value'},
...  {'my_arg1': 'a', 'my_arg2': 10, 'my_arg3': 'fixed_value'},
...  {'my_arg1': 'b', 'my_arg2': 10, 'my_arg3': 'fixed_value'},
...  {'my_arg1': 'c', 'my_arg2': 10, 'my_arg3': 'fixed_value'}]
True
```

pyhrf.tools.misc.**cartesian_eval**(*func*, *varargs*, *fixedargs=None*)

pyhrf.tools.misc.**cartesian_params**(***kwargs*)

pyhrf.tools.misc.**cartesian_test**()

pyhrf.tools.misc.**check_files_series**(*fseries*, *verbose=False*)

pyhrf.tools.misc.**closestsorted**(*a*, *val*)

pyhrf.tools.misc.**condense_series**(*numbers*)

pyhrf.tools.misc.**convex_hull_mask**(*mask*)
Compute the convex hull of the positions defined by the given binary mask

Parameters **mask** (–) – binary mask of positions to build the chull from

Returns a numpy.ndarray binary mask of positions within the convex hull

pyhrf.tools.misc.**crop_array**(*a*, *m=None*, *extension=0*)
Return a sub array where as many zeros as possible are discarded Increase bounding box of mask by *extension*

pyhrf.tools.misc.**cuboidPrinter**(*c*)

pyhrf.tools.misc.**describeRois**(*roiMask*)

pyhrf.tools.misc.**diagBlock**(*mats*, *rep=0*)
Construct a diagonal block matrix from blocks which can be 1D or 2D arrays. 1D arrays are taken as column vectors. If 'rep' is a non null positive number then blocks are diagonaly 'rep' times

pyhrf.tools.misc.**distance**(*c1*, *c2*, *coord_system=None*)

pyhrf.tools.misc.**do_if_nonexistent_file**(**dargs*, ***kwargs*)

pyhrf.tools.misc.**extractRoiMask**(*nmask*, *roiId*)

pyhrf.tools.misc.**extract_file_series**(*files*)
group all file names sharing a common prefix followed by a number, ie: <prefix><number><extension> Return a dictionnary with two levels (<tag>,<extension>), mapped to all corresponding series index found.

pyhrf.tools.misc.**foo**(**args*, ***kwargs*)

pyhrf.tools.misc.**format_duration**(*dt*)

pyhrf.tools.misc.**format_serie**(*istart*, *iend*)

pyhrf.tools.misc.**gaussian_blur**(*a*, *shape*)

pyhrf.tools.misc.**gaussian_kernel**(*shape*)
Returns a normalized ND gauss kernel array for convolutions

pyhrf.tools.misc.**get_2Dtable_string**(*val*, *rownames*, *colnames*, *precision=4*, *col_sep='|'*, *line_end=''*, *line_start=''*, *outline_char=None*)
Return a nice tabular string representation of a 2D numeric array #TODO : make colnames and rownames optional

pyhrf.tools.misc.**get_cache_filename**(*args*, *path='./'*, *prefix=None*, *gz=True*)

pyhrf.tools.misc.**get_leaf**(*element*, *branch*)
Return the nested leaf element corresponding to all dictionnary keys in branch from element

pyhrf.tools.misc.**group_file_series**(*series*, *group_rules=None*)

pyhrf.tools.misc.**has_ext**(*fn*, *ext*)

pyhrf.tools.misc.**hash_func_input**(*func*, *args*, *digest_code*)

pyhrf.tools.misc.**html_body**(*s*)

pyhrf.tools.misc.**html_cell**(*s*, *cell_type='d'*, *attrs=None*)

pyhrf.tools.misc.**html_div**(*s*, *attrs=None*)

pyhrf.tools.misc.**html_doc**(*s*)

pyhrf.tools.misc.**html_head**(*s*)

pyhrf.tools.misc.**html_img**(*fn*, *attrs=None*)

pyhrf.tools.misc.**html_list_to_row**(*l*, *cell_types*, *attrs*)

pyhrf.tools.misc.**html_row**(*s*)

pyhrf.tools.misc.**html_style**(*s*)

pyhrf.tools.misc.**html_table**(*s*, *border=None*)

pyhrf.tools.misc.**icartesian_combine_args**(*varying_args*, *fixed_args=None*)
Same as cartesian_combine_args but return an iterator over the list of argument combinations

pyhrf.tools.misc.**inspect_default_args**(*args*, *defaults*)

pyhrf.tools.misc.**is_importable**(*module_name*, *func_name=None*)
Return True if given *module_name* (str) is importable

pyhrf.tools.misc.**map_dict**(*func*, *d*)

pyhrf.tools.misc.**montecarlo**(*datagen*, *festim*, *nbit=None*)
Perform a Monte Carlo loop with data generator 'datagen' and estimation function 'festim'. 'datagen' have to be iterable. 'festim' must return an object on which ** and + operators can be applied. If 'nbit' is provided then use it for the maximum iterations else loop until datagen stops.

pyhrf.tools.misc.**my_func**(***kwargs*)

pyhrf.tools.misc.**nc2DGrid**(*maxSize*)

pyhrf.tools.misc.**non_existent_canditate**(*f*, *start_idx=1*)

pyhrf.tools.misc.**non_existent_file**(*f*)

pyhrf.tools.misc.**now**()

pyhrf.tools.misc.**peelVolume3D**(*volume*, *backgroundLabel=0*)

exception pyhrf.tools.misc.**polyError**(*expression*, *message*)
Bases: exceptions.Exception

pyhrf.tools.misc.**polyFit**(*signal*, *tr=1.0*, *order=5*)
Polynomial fit of signals. 'signal' is a 2D matrix with first axis being time and second being position. 'tr' is the time resolution (dt). 'order' is the order of the polynom. Return the orthogonal polynom basis matrix (P) and fitted coefficients (l), such that P.l yields fitted polynoms.

pyhrf.tools.misc.**rebin**(*a*, *newshape*)
Rebin an array to a new shape. Can be used to rebin a func image onto a anat image

pyhrf.tools.misc.**replace_ext**(*fn*, *ext*)

pyhrf.tools.misc.**report_arrays_in_obj**(*o*)

pyhrf.tools.misc.**resampleSignal**(*s*, *osf*)

pyhrf.tools.misc.**resampleToGrid**(*x*, *y*, *xgrid*)

pyhrf.tools.misc.**rescale_values**(*a*, *v_min=0.0*, *v_max=1.0*, *axis=None*)

pyhrf.tools.misc.**root3**(*listCoeffs*)

pyhrf.tools.misc.**set_leaf**(*tree*, *branch*, *leaf*, *branch_classes=None*)
Set the nested *leaf* element corresponding to all dictionnary keys defined in *branch*, within *tree*

pyhrf.tools.misc.**stack_trees**(*trees*, *join_func=None*)
Stack trees (python dictionnaries) with identical structures into one tree so that one leaf of the resulting tree is a list of the corresponding leaves across input trees. 'trees' is a list of dict

pyhrf.tools.misc.**swap_layers**(*t*, *labels*, *l1*, *l2*)
Create a new tree from t where layers labeled by l1 and l2 are swapped. labels contains the branch labels of t.

pyhrf.tools.misc.**swapaxes**(*array*, *a1*, *a2*)

pyhrf.tools.misc.**time_diff_str**(*diff*)

pyhrf.tools.misc.**tree**(*branched_leaves*)

pyhrf.tools.misc.**treeBranches**(*tree*, *branch=None*)

pyhrf.tools.misc.**treeBranchesClasses**(*tree*, *branch=None*)

pyhrf.tools.misc.**tree_items**(*tree*)

pyhrf.tools.misc.**tree_leaves**(*tree*)

pyhrf.tools.misc.**tree_rearrange**(*t*, *oldlabels*, *newlabels*)
Create a new tree from t where layers are rearranged following newlabels. oldlabels contains the branch labels of t.

pyhrf.tools.misc.**undrift**(*signal*, *tr*, *order=5*)
Remove the low frequency trend from 'signal' by a polynomial fit. Assume axis 3 of 'signal' is time.

pyhrf.tools.misc.**unstack_trees**(*tree*)
Return a list of tree from a tree where leaves are all lists with the same number of items

1.7 pyhrf.ui package

1.7.1 Submodules

pyhrf.ui.analyser_ui module

class pyhrf.ui.analyser_ui.**FMRIAnalyser**(*outputPrefix=''*, *roiAverage=False*, *pass_error=True*, *gzip_outputs=False*)

Bases: *pyhrf.xmlio.Initable*

P_OUTPUT_PREFIX = 'outputPrefix'

P_ROI_AVERAGE = 'averageRoiBold'

analyse(*data*, *output_dir=None*)

Launch the wrapped analyser onto the given data

Parameters

- **data** (FmriData) – the input fMRI data set (there may be multi parcels)
- **output_dir** (*str*) – the path where to store parcel-specific fMRI data sets (after splitting according to the parcellation mask)

Returns a list of analysis results -> (list of tuple(FmriData, Noneloutput of analyse_roi, str)) = (list of tuple(parcel data, analysis results, analysis report)) See method analyse_roi_wrap

analyse_roi(*roiData*)

analyse_roi_wrap(*roiData*)

Wrap the analyse_roi method to catch potential exception

analyse_roi_wrap_bak(*roiData*)

clean_output_files(*output_dir*)

enable_draft_testing()

filter_crashed_results(*results*)

get_label()

joinOutputs(*cuboids*, *roiIds*, *mappers*)

make_outputs_multi_subjects(*data_rois*, *irois*, *all_outputs*, *targetAxes*, *ext*, *meta_data*, *output_dir*)

make_outputs_single_subject(*data_rois*, *irois*, *all_outputs*, *targetAxes*, *ext*, *meta_data*, *output_dir*)

outputResults(*results*, *output_dir*, *filter='.\\A'*)

Return: a tuple (dictionary of outputs, output file names)

outputResults_back_compat(*results*, *output_dir*, *filter='.\\A'*)

parametersComments = {'averageRoiBold': 'Average BOLD signals within each ROI before

parametersToShow = ['averageRoiBold', 'outputPrefix']

set_gzip_outputs(*gzip_outputs*)

set_pass_errors(*pass_error*)

split_data(*fdata*, *output_dir=None*)

pyhrf.ui.glm_analyser module

class pyhrf.ui.glm_analyser.**GLMAnalyser**(*outputPrefix='glm_'*)
Bases: *pyhrf.ui.analyser_ui.FMRIAnalyser*

analyse_roi(*fdata*)

get_label()

pyhrf.ui.glm_ui module

class pyhrf.ui.glm_ui.**GLMAnalyser**(*contrasts={'dummy_contrast_example': '3*audio-video/3'}, contrast_test_baseline=0.0, hrf_model='Canonical', drift_model='Cosine', hfcut=128.0, residuals_model='spherical', fit_method='ols', outputPrefix='glm_', rescale_results=False, rescale_factor_file=None, fir_delays=[0], output_fit=False*)
Bases: *pyhrf.ui.analyser_ui.FMRIAnalyser*

analyse_roi(*fdata*)

get_label()

parametersComments = {'fit_method': 'Either "ols" or "kalman"', 'residuals_model': '

parametersToShow = []

pyhrf.ui.jde module

class pyhrf.ui.jde.**JDEAnalyser**(*outputPrefix='jde_', pass_error=True*)
Bases: *pyhrf.ui.analyser_ui.FMRIAnalyser*

get_label()

class pyhrf.ui.jde.**JDEMCMCAnalyser**(*sampler=<pyhrf.jde.models.BOLDGibbsSampler object>, osfMax=4, dtMin=0.4, dt=0.6, driftParam=4, driftType='polynomial', outputPrefix='jde_mcmc_', randomSeed=None, pass_error=True, copy_sampler=True*)
Bases: *pyhrf.ui.jde.JDEAnalyser*

Class that wraps a JDE Gibbs Sampler to launch an fMRI analysis TODO: remove parameters about dt and osf (should go in HRF Sampler class), drift (should go in Drift Sampler class)

P_DRIFT_LFD_PARAM = 'driftParam'

P_DRIFT_LFD_TYPE = 'driftType'

P_DT = 'dt'

P_DTMIN = 'dtMin'

P_OSFMAX = 'osfMax'

P_RANDOM_SEED = 'randomSeed'

P_SAMPLER = 'sampler'

analyse_roi(*atomData*)
Launch the JDE Gibbs Sampler on a parcel-specific data set *atomData* :param - atomData: parcel-specific data :type - atomData: pyhrf.core.FmriData

Returns JDE sampler object

enable_draft_testing()

packSamplerInput(*roiData*)

parametersComments = {'driftType': 'Either "cosine" or "polynomial" or "None"', 'dtMi

parametersToShow = ['dtMin', 'dt', 'driftType', 'driftParam', 'sampler']

pyhrf.ui.jde.**jde_analyse**(*data=None*, *nbIterations=3*, *hrfModel='estimated'*, *hrfNorm=1.0*, *hrfTrick=False*, *sampleHrfVar=True*, *hrfVar=1e-05*, *keepSamples=False*, *samplesHistPace=1*)

pyhrf.ui.jde.**runEstimationBetaEstim**(*params*)

pyhrf.ui.jde.**runEstimationSupervised**(*params*)

pyhrf.ui.rfir_ui module

class pyhrf.ui.rfir_ui.**RFIRAnalyser**(*HrfEstimator=<pyhrf.rfir.RFIREstim object>*, *outputPrefix='hrf_'*)

Bases: *pyhrf.ui.analyser_ui.FMRIAnalyser*

analyse_roi(*atomData*)

parametersToShow = ['HrfEstimator']

pyhrf.ui.treatment module

class pyhrf.ui.treatment.**FMRITreatment**(*fmri_data=FmriData(onsets=OrderedDict([('audio', [array([15. , 20.7, 29.7, 35.4, 44.7, 48. , 83.4, 89.7, 108. , 119.4, 135. , 137.7, 146.7, 173.7, 191.7, 236.7, 251.7, 284.4, 293.4, 296.7])]), ('video', [array([0. , 2.4, 8.7, 33. , 39. , 41.7, 56.4, 59.7, 75. , 96. , 122.7, 125.4, 131.4, 140.4, 149.4, 153. , 156. , 159. , 164.4, 167.7, 176.7, 188.4, 195. , 198. , 201. , 203.7, 207. , 210. , 218.7, 221.4, 224.7, 234. , 246. , 248.4, 260.4, 264. , 266.7, 269.7, 278.4, 288.])])]),bold=array([[126.23066711, 115.65962982, 122.31136322, ..., 116.81171417, 116.13331604, 111.56729126], [127.20968628, 114.09086609, 119.19332886, ..., 117.53836823, 115.68979645, 111.80612183], [125.94182587, 115.54189301, 122.27174377, ..., 113.3349762 , 114.11522675, 107.98921967], ..., [122.55458069, 110.07359314, 120.74815369, ..., 118.938797 , 117.44371796, 113.07602692], [120.98430634, 110.80142212, 118.64087677, ..., 117.53933716, 115.92312622, 110.41734314], [124.12675476, 114.32633209, 121.71891785, ..., 117.40695953, 117.4828949 , 110.78852081]], dtype=float32),tr=2.4,sessionsScans=[array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124])],roiMask=array([[[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]], [[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]], [[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]], ..., [[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]], [[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]], [[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]]], dtype=int32),graph=None,stimDurations=OrderedDict([('audio', [array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])]), ('video', [array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])])]),meta_obj=(array([[-3. , 0. , 0. , 78.], [0. , 3. , 0. , -93.], [0. , 0. , 3. , -67.5], [0. , 0. , 0. , 1.]]), <nibabel.nifti1.Nifti1Header object>)), analyser=<pyhrf.ui.jde.JDEMCMCAnalyser object>, output_dir='./', make_outputs=True, result_dump_file=None*)

already_done()

clean_output_files()

dump_roi_datasets(*dry=False*, *output_dir=None*)

enable_draft_testing()

execute()

get_data_files()

output(*result*, *dump_result=True*, *outputs=True*)

parametersComments = {'result_dump_file': 'File to save the analyser result (uses pic

parametersToShow = ['fmri_data', 'output_dir', 'analyser']

pickle_result(*result*)

replace_data_dir(*d*)

run(*parallel=None*, *n_jobs=None*)
Run the analysis: load data, run estimation, output results

split(*dump_sub_results=None*, *make_sub_outputs=None*, *output_dir=None*, *output_file_list=None*)

xmlComment = "Group all parameters for a within-subject analysis.\nTwo main parts:\n -

pyhrf.ui.treatment.**append_common_treatment_options**(*parser*)

pyhrf.ui.treatment.**create_treatment**(*boldFiles*, *parcelFile*, *dt*, *tr*, *paradigmFile*, *nbIterations=4000*, *writeXmlSetup=True*, *parallelize=False*, *outputDir=None*, *outputSuffix=None*, *outputPrefix=None*, *contrasts=None*, *beta=0.6*, *estimBeta=True*, *pfMethod='ps'*, *estimHrf=True*, *hrfVar=0.01*, *roiIds=None*, *nbClasses=2*, *gzip_rdump=False*, *make_outputs=True*, *vbjde=False*, *simulation_file=None*)

pyhrf.ui.treatment.**create_treatment_surf**(*boldFiles*, *parcelFile*, *meshFile*, *dt*, *tr*, *paradigmFile*, *nbIterations=4000*, *writeXmlSetup=True*, *parallelize=False*, *outputDir=None*, *outputSuffix=None*, *outputPrefix=None*, *contrasts=';'*, *beta=0.6*, *estimBeta=True*, *pfMethod='ps'*, *estimHrf=True*, *hrfVar=0.01*, *roiIds=None*, *nbClasses=2*, *gzip_rdump=False*, *simulation_file=None*, *make_outputs=True*)

pyhrf.ui.treatment.**exec_t**(*t*)

pyhrf.ui.treatment.**jde_surf_from_files**(*boldFiles=['/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/la packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/real_data_surf_tiny_bold.gii'], parcelFile='/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/la packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/real_data_surf_tiny_parcellation.gii', meshFile='/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/lat packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/real_data_surf_tiny_mesh.gii', dt=0.6, tr=2.4, paradigm_csv_file='/home/docs/checkouts/readthedocs.org/u packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/paradigm_loc_av.csv', nbIterations=4000, writeXmlSetup=True, parallelize=None, outputDir=None, outputSuffix=None, outputPrefix=None, contrasts=None, beta=0.6, estimBeta=True, pfMethod='ps', estimHrf=True, hrfVar=0.01, roiIds=None, force_relaunch=False, nbClasses=2, gzip_rdump=False, dry=False, simulation_file=None*)

pyhrf.ui.treatment.**jde_vol_from_files**(*boldFiles=['/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/lat packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/subj0_bold_session0.nii.gz'], parcelFile='/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/lat packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/subj0_parcellation.nii.gz', dt=0.6, tr=2.4, paradigm_csv_file='/home/docs/checkouts/readthedocs.org/use packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/paradigm_loc_av.csv', nbIterations=4000, writeXmlSetup=True, parallelize=None, outputDir=None, outputSuffix=None, outputPrefix=None, contrasts=None, beta=0.6, estimBeta=True, pfMethod='ps', estimHrf=True, hrfVar=0.01, roiIds=None, force_relaunch=False, nbClasses=2, gzip_rdump=False, dry=False, make_outputs=True, vbjde=False, simulation_file=None*)

pyhrf.ui.treatment.**make_outfile**(*fn*, *path*, *pre=''*, *suf=''*)

pyhrf.ui.treatment.**parse_data_options**(*options*)

Return an FmriData object corresponding to input options

pyhrf.ui.treatment.**run_pyhrf_cmd_treatment**(*cfg_cmd*, *exec_cmd*, *default_cfg_file*, *default_profile_file*, *label_for_cluster*)

pyhrf.ui.vb_jde_analyser module

class pyhrf.ui.vb_jde_analyser.**JDEVEMAnalyser**(*hrfDuration=25.0*, *sigmaH=0.1*, *fast=True*, *computeContrast=True*, *nbClasses=2*, *PLOT=False*, *nItMax=100*, *nItMin=1*, *scale=False*, *beta=1.0*, *estimateSigmaH=True*, *estimateHRF=True*, *TrueHrfFlag=False*, *HrfFilename='hrf.nii'*, *estimateDrifts=True*, *hyper_prior_sigma_H=1000*, *dt=0.6*, *estimateBeta=True*, *contrasts=None*, *simulation=False*, *estimateLabels=True*, *LabelsFilename=None*, *MFapprox=False*, *estimateMixtParam=True*, *constrained=False*, *InitVar=0.5*, *InitMean=2.0*, *MiniVemFlag=False*, *NbItMiniVem=5*, *zero_constraint=True*, *output_drifts=False*, *drifts_type='poly'*)

Bases: *pyhrf.ui.jde.JDEAnalyser*

Class that handles parcel-wise which is done according to the input data parcellation by default, and also takes care of merging parcel-specific outputs at the end of the JDE VEM analysis.

Parameters

- **hrfDuration** (*float*) – duration of the HRF in seconds.
- **sigmaH** (*float*) – initial HRF variance.
- **fast** (*bool*) – running fast VEM with C extensions.
- **computeContrast** (*bool*) – compute the contrasts defined in contrasts input.
- **nbClasses** (*int*) – number of classes for the response levels.
- **PLOT** (*bool*) – plotting flag for convergence curves.
- **nItMax** (*int*) – maximum number of iterations.
- **nItMin** (*int*) – minimum number of iterations.
- **scale** (*bool*) – divide the data fidelity term during m_h step by the number of voxels.
- **beta** (*float*) – initial value of spatial Potts regularization parameter.
- **estimateSigmaH** (*bool*) – estimate or not the HRF variance.
- **estimateHRF** (*bool*) – estimate or not the HRF.
- **TrueHrfFlag** (*bool*) – if true, HRF will be fixed to the simulated value.
- **HrfFilename** (*str*) – filename of the output HRF.
- **estimateDrifts** (*bool*) – if true drifts are estimated, otherwise drifts are marginalized.
- **hyper_prior_sigma_H** (*int*) – hyper-prior on sigma_H. If zero, no prior is applied.
- **dt** (*float*) – time resolution of the estimated HRF (in seconds).
- **estimateBeta** (*bool*) – estimate or not the Potts spatial regularization parameter.
- **contrasts** (OrderedDict) – contrasts to be evaluated.
- **simulation** (*bool*) – indicates whether the run corresponds to a simulation example or not.

- **estimateLabels** (*bool*) – estimate or not the labels.
- **LabelsFilename** (*str*) – filename containing the labels.
- **MFapprox** (*bool*) – using the Mean Field approximation in labels estimation.
- **estimateMixtParam** (*bool*) – estimate or not the mixture parameters.
- **constrained** (*bool*) – if true, the following constraints are added: positivity and norm = 1.
- **InitVar** (*float*) – initial value of active and inactive gaussian variances.
- **InitMean** (*float*) – initial value of active gaussian means.
- **MiniVemFlag** (*bool*) – estimate or not the best initialization of MixtParam and gamma_h.
- **NbItMiniVem** (*int*) – number of iterations in Mini VEM algorithm.
- **zero_constraint** (*bool*) – if true, add zeros at the beginning and the end of the HRF.
- **output_drifts** (*bool*) – save the estimated drifts.
- **drifts_type** (*str*) – type of the drift basis ('poly' or 'cos').

analyse_roi(*roiData*)
ROI analysis of the fMRI data.

Parameters **roiData** (*FmriData*) – fMRI data to be analyzed.

Returns packed outputs.

Return type dict of *xndarray*

parametersComments = {'estimateDrifts': 'explicit drift estimation (if false, drifts

parametersToShow = ['dt', 'hrfDuration', 'nItMax', 'nItMin', 'estimateSigmaH', 'estima

pyhrf.ui.vb_jde_analyser.**change_dim**(*labels*)
Change labels dimension from (ncond, nclass, nvox) to (nclass, ncond, nvox).

pyhrf.ui.vb_jde_analyser.**run_analysis**(***params*)
Function to run the JDE VEM analyzer with parallel computation

pyhrf.ui.vb_jde_analyser_asl_fast module

class pyhrf.ui.vb_jde_analyser_asl_fast.**JDEVEMAnalyser**(*hrfDuration=25.0*, *dt=0.5*, *fast=True*, *constrained=False*, *nbClasses=2*, *PLOT=False*, *nItMax=1*, *nItMin=1*, *scale=False*, *beta=1.0*, *simulation=None*, *fmri_data=None*, *computeContrast=True*, *estimateH=True*, *estimateG=True*, *use_hyperprior=False*, *estimateSigmaH=True*, *estimateSigmaG=True*, *positivity=False*, *sigmaH=0.0001*, *sigmaG=0.0001*, *sigmaMu=0.0001*, *physio=True*, *gammaH=1000*, *gammaG=1000*, *zero_constrained=False*, *estimateLabels=True*, *estimateMixtParam=True*, *contrasts=None*, *InitVar=0.5*, *InitMean=2.0*, *estimateA=True*, *estimateC=True*, *estimateBeta=True*, *estimateNoise=True*, *estimateLA=True*, *phy_params={'E0': 0.34, 'tau_m': 0.98, 'r0': 100, 'linear': False, 'vt0': 80.6, 'tau_f': 2.46, 'eps': 0.54, 'tau_s': 1.54, 'V0': 1, 'alpha_w': 0.33, 'model': 'RBM', 'obata': False, 'buxton': False, 'e': 1.43, 'eps_max': 10.0, 'TE': 0.018, 'model_name': 'Khalidov11'}*, *prior='omega'*, *n_session=1*)

Bases: *pyhrf.ui.jde.JDEAnalyser*

analyse_roi(*roiData*)

finalizeEstimation(*true_labels*, *labels*, *nvox*, *true_brf*, *estimated_brf*, *true_prf*, *estimated_prf*, *true_brls*, *brls*, *true_prls*, *prls*, *true_drift*, *PL*, *L*, *true_noise*, *noise*)

parametersComments = {'hrfDuration': 'duration of the HRF in seconds', 'nbClasses':

parametersToShow = ['dt', 'hrfDuration', 'nItMax', 'nItMin', 'estimateSigmaH', 'estima

pyhrf.ui.vb_jde_analyser_asl_fast.**change_dim**(*labels*)

Change labels dimension from (ncond, nclass, nvox) to (nclass, ncond, nvox)

pyhrf.ui.vb_jde_analyser_asl_fast.**run_analysis**(***params*)

pyhrf.ui.vb_jde_analyser_bold_fast module

class pyhrf.ui.vb_jde_analyser_bold_fast.**JDEVEMAnalyser**(*hrfDuration=25.0*, *dt=0.5*, *fast=True*, *constrained=False*, *nbClasses=2*, *PLOT=False*, *nItMax=1*, *nItMin=1*, *scale=False*, *beta=1.0*, *simulation=None*, *fmri_data=None*, *computeContrast=True*, *estimateH=True*, *estimateG=True*, *use_hyperprior=False*, *estimateSigmaH=True*, *estimateSigmaG=True*, *positivity=False*, *sigmaH=0.0001*, *sigmaG=0.0001*, *sigmaMu=0.0001*, *physio=True*, *gammaH=1000*, *gammaG=1000*, *zero_constrained=False*, *estimateLabels=True*, *estimateMixtParam=True*, *contrasts=None*, *InitVar=0.5*, *InitMean=2.0*, *estimateA=True*, *estimateC=True*, *estimateBeta=True*, *estimateNoise=True*, *estimateLA=True*, *phy_params={'E0': 0.34, 'tau_m': 0.98, 'r0': 100, 'linear': False, 'vt0': 80.6, 'tau_f': 2.46, 'eps': 0.54, 'tau_s': 1.54, 'V0': 1, 'alpha_w': 0.33, 'model': 'RBM', 'obata': False, 'buxton': False, 'e': 1.43, 'eps_max': 10.0, 'TE': 0.018, 'model_name': 'Khalidov11'}*, *prior='no'*, *n_session=1*)

Bases: *pyhrf.ui.jde.JDEAnalyser*

analyse_roi(*roiData*)

finalizeEstimation(*true_labels*, *labels*, *nvox*, *true_brf*, *estimated_brf*, *true_prf*, *estimated_prf*, *true_brls*, *brls*, *true_prls*, *prls*, *true_drift*, *PL*, *L*, *true_noise*, *noise*)

parametersComments = {'hrfDuration': 'duration of the HRF in seconds', 'nbClasses':

parametersToShow = ['dt', 'hrfDuration', 'nItMax', 'nItMin', 'estimateSigmaH', 'estima

pyhrf.ui.vb_jde_analyser_bold_fast.**change_dim**(*labels*)
Change labels dimension from (ncond, nclass, nvox) to (nclass, ncond, nvox)

pyhrf.ui.vb_jde_analyser_bold_fast.**run_analysis**(***params*)

1.8 pyhrf.validation package

pyhrf.validation.**randn**(*d0*, *d1*, ..., *dn*)
Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

1.8.1 Submodules

pyhrf.validation.config module

pyhrf.validation.config.**clean_cache**()

pyhrf.validation.config.**figfn**(*fn*)
Append the figure file extension to fn

pyhrf.validation.config.**is_tmp_file**(*fn*)

pyhrf.validation.valid_beta_estim module

class pyhrf.validation.valid_beta_estim.**ObsField2DTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test estimation of beta with on observed 2D fields

MC_comp_pfmethods_ML(*shape*)

MC_comp_pfmethods_ML_3C(*shape*)

setUp()

test_MC_comp_pfmethods_ML_100x100()

test_MC_comp_pfmethods_ML_10x10()

test_MC_comp_pfmethods_ML_30x30()

test_MC_comp_pfmethods_ML_3C_10x10()

test_MC_comp_pfmethods_ML_3C_20x20()

test_MC_comp_pfmethods_ML_3C_30x30()

test_MC_comp_pfmethods_ML_3C_50x50()

test_single_Onsager_MAP()
PF method: Onsager. MAP on p(label|beta).

test_single_Onsager_ML()
PF method: Onsager. ML on p(beta|label).

test_single_PFES_MAP()
PF estimation method : extrapolation scheme. MAP on p(beta|label).

test_single_PFES_ML()
PF estimation method : extrapolation scheme. ML on p(label|beta).

test_single_PFPS_MAP()
PF estimation method : path sampling. MAP on p(beta|label).

test_single_PFPS_ML()
PF estimation method : path sampling. ML on p(label|beta).

test_single_surface_PFPS_ML()
PF estimation method : path sampling. ML on p(label|beta). topology from a surfacic RDI

pyhrf.validation.valid_beta_estim.**beta_estim_obs_field_mc**(*graph*, *nbClasses*, *beta*, *gridLnz*, *mcit=1*, *cachePotts=False*)

pyhrf.validation.valid_beta_estim.**dist**(*x*, *y*)

pyhrf.validation.valid_beta_estim.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns **Z** – A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

Return type *ndarray* or float

See also:

random.standard_normal() Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

pyhrf.validation.valid_beta_estim.**test_beta_estim_obs_fields**(*graphs*, *betas*, *nbLabels*, *pfmethod*, *mcit=1*)

pyhrf.validation.valid_jde_asl module

class pyhrf.validation.valid_jde_asl.**ASLTest**(*methodName='runTest'*)

Bases: unittest.case.TestCase

setUp()

tearDown()

test_all()

Validate estimation of full ASL model at high SNR

test_brf()
Validate estimation of BRF at high SNR

test_brls()
Validate estimation of BRLs at high SNR

test_drift()
Validate estimation of drift at high SNR

test_labels()
Validate estimation of labels at high SNR

test_noise_var()
Validate estimation of noise variances at high SNR

test_prf()
Validate estimation of PRF

test_prls()
Validate estimation of PRLs at high SNR

pyhrf.validation.valid_jde_asl_physio module

class pyhrf.validation.valid_jde_asl_physio.**ASLPhysioHierarchicalTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_brf()
Validate estimation of BRF

test_mu()
Validate estimation of mu

test_mu_brf()
Validate estimation of mu and brf

test_mu_brf_prf()
Validate estimation of mu, brf and prf

test_mu_prf()
Validate estimation of mu, brf and prf

test_prf()
Validate estimation of PRF

class pyhrf.validation.valid_jde_asl_physio.**ASLTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_all()
Validate estimation of full ASL model at high SNR

test_brf_basic_reg()
Validate estimation of BRF at high SNR

test_brf_physio_det()
Validate estimation of BRF at high SNR

test_brf_physio_nonreg()
Validate estimation of BRF at high SNR

test_brf_physio_reg()
Validate estimation of BRF at high SNR

test_brf_var()
Validate estimation of BRF at high SNR

test_brls()
Validate estimation of BRLs at high SNR

test_drift()
Validate estimation of drift at high SNR

test_drift_var()
Validate estimation of drift at high SNR

test_labels()
Validate estimation of labels at high SNR

test_noise_var()
Validate estimation of noise variances at high SNR

test_perf_baseline()
Validate estimation of drift at high SNR

test_perf_baseline_var()
Validate estimation of drift at high SNR

test_prf_basic_reg()
Validate estimation of BRF at high SNR

test_prf_physio_det()
Validate estimation of BRF at high SNR

test_prf_physio_nonreg()
Validate estimation of BRF at high SNR

test_prf_physio_reg()
Validate estimation of PRF

test_prf_var()
Validate estimation of PRF

test_prls()
Validate estimation of PRLs at high SNR

pyhrf.validation.valid_jde_asl_physio_alpha module

class pyhrf.validation.valid_jde_asl_physio_alpha.**ASLTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_all()
Validate estimation of full ASL model at high SNR

test_brf_basic_reg()
Validate estimation of BRF at high SNR

test_brf_physio_det()
Validate estimation of BRF at high SNR

test_brf_physio_nonreg()
Validate estimation of BRF at high SNR

test_brf_physio_reg()
Validate estimation of BRF at high SNR

test_brf_var()
Validate estimation of BRF at high SNR

test_brls()
Validate estimation of BRLs at high SNR

test_drift()
Validate estimation of drift at high SNR

test_drift_var()
Validate estimation of drift at high SNR

test_labels()
Validate estimation of labels at high SNR

test_noise_var()
Validate estimation of noise variances at high SNR

test_perf_baseline()
Validate estimation of drift at high SNR

test_perf_baseline_var()
Validate estimation of drift at high SNR

test_prf_basic_reg()
Validate estimation of BRF at high SNR

test_prf_physio_det()
Validate estimation of BRF at high SNR

test_prf_physio_nonreg()
Validate estimation of BRF at high SNR

test_prf_physio_reg()
Validate estimation of PRF

test_prf_var()
Validate estimation of PRF

test_prls()
Validate estimation of PRLs at high SNR

pyhrf.validation.valid_jde_bold_mono_subj_multi_sess module

class pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.**MultiSessTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_bmixt_sampler()

test_default_jde_small_simulation()
Test JDE multi-sessions sampler on small simulation with small nb of iterations. Estimation accuracy is not tested.

test_drift_and_var_sampler()

test_drift_sampler()

test_drift_var_sampler()

test_full_sampler()
Test JDE Multi-sessions sampler on simulation with normal size. Estimation accuracy is tested.

test_hrf_sampler()

test_hrf_var_sampler()

test_label_sampler()

test_noise_var_sampler()

test_nrl_bar_only_sampler()

test_nrl_bar_sampler()

test_nrl_by_session_hrf()

test_nrl_by_session_sampler()

test_nrl_by_session_var_sampler()

test_simulation()

pyhrf.validation.valid_jde_bold_mono_subj_sess module

class pyhrf.validation.valid_jde_bold_mono_subj_sess.**JDETest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_full_sampler()
Test JDE on simulation with normal size. Estimation accuracy is tested.

test_hrf_var_sampler()

test_hrf_var_sampler_2()

test_hrf_with_var_sampler()

test_hrf_with_var_sampler_2()

test_noise_var_sampler()

pyhrf.validation.valid_jde_vem_asl module

class pyhrf.validation.valid_jde_vem_asl.**ASLTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_E_step()
Validate estimation of perfusion component at high SNR

test_all()
Validate estimation of full ASL model at high SNR

test_beta()
Validate estimation of drift at high SNR

test_bold()
Validate estimation of bold component at high SNR

test_brf()
Validate estimation of BRF at high SNR

test_brls()
Validate estimation of BRLs at high SNR

test_la()
Validate estimation of drift at high SNR

test_labels()
Validate estimation of labels at high SNR

test_mp()
Validate estimation of drift at high SNR

test_noise_var()
Validate estimation of noise variances at high SNR

test_perfusion()
Validate estimation of perfusion component at high SNR

test_prf()
Validate estimation of PRF

test_prls()
Validate estimation of PRLs at high SNR

test_sigmaG()
Validate estimation of drift at high SNR

test_sigmaH()
Validate estimation of drift at high SNR

pyhrf.validation.valid_rfir module

class pyhrf.validation.valid_rfir.**RFIRTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test the Regularized FIR (RFIR)-based methods implemented in pyhrf.rfir

setUp()

tearDown()

test_results_small_simulation()
TODO: move to validation

test_rfir_on_small_simulation()
Check if pyhrf.rfir runs properly and that returned outputs contains the expected items

pyhrf.validation.valid_rndm_field module

class pyhrf.validation.valid_rndm_field.**PartitionFunctionTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_comparison()

test_extrapolation()

test_onsager()

test_onsager1()

test_path_sampling()

class pyhrf.validation.valid_rndm_field.**PottsTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_SW_nrj()

test_SW_nrj_2C_3C()

test_gibbs()

test_sw_nrj()

test_sw_sampling()

class pyhrf.validation.valid_rndm_field.**field_energy_calculator**(*graph*)

pyhrf.validation.valid_sandbox_parcellation module

class pyhrf.validation.valid_sandbox_parcellation.**FeatureExtractionTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_feature_extraction()

test_generate_features()

class pyhrf.validation.valid_sandbox_parcellation.**ParcellationTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

save_parcellation_outputs(*pobj*, *mask*)

setUp()

tearDown()

test_gmm_from_forged_features()
Test spatial Ward with uncertainty on forged features

test_hemodynamic_parcellation_GMM_2D_high_SNR()
test GMM-based parcellation on features extracted from a 2D artificial fMRI data set, at high SNR

test_hemodynamic_parcellation_wpu_2D_high_SNR()
test WPU on features extracted from a 2D artificial fMRI data set, at high SNR

test_mixtdist()
Check that merge is in favour of non-activ at the same feature level, starting from singleton clusters.

test_parcellation_history()

test_parcellation_mmp_act_level_1D()
Test the ability of MMP to 'jump' non-activating positions (1D case).

test_parcellation_mmp_act_level_2D()
Test the ability of MMP to 'jump' non-activating positions (2D case).

test_parcellation_spatialWard_2()
Test WPU on a simple case.

test_parcellation_spatialWard_400_nonoise()

test_parcellation_spatialWard_400_variance()

test_parcellation_spatialWard_5_sklearn()

test_parcellation_spatialWard_act_level_1D()
Test the ability of WPU to 'jump' non-activating positions (1D case).

test_parcellation_spatialWard_act_level_2D()
Test the ability of WPU to 'jump' non-activating positions (2D case).

test_parcellation_spatialWard_variance_1D()
Test the ability of WPU to 'jump' non-activating positions (1D case).

test_parcellation_spatialWard_variance_2D()
Test the sensibility to variance (2D case).

test_render_ward_tree()

test_spatialward_against_modelbasedspatialward()
Check that pyhrf's spatial Ward parcellation is giving the same results as scikit's spatial Ward parcellation

test_spatialward_against_ward_sk()
Check that pyhrf's spatial Ward parcellation is giving the same results as scikit's spatial Ward parcellation

test_spatialward_from_forged_features()
Test spatial Ward on forged features

test_uspatialward_formula()
Check that pyhrf's Uncertain spatial Ward parcellation is giving the same results as Uncertain spatial Ward parcellation modified formula

test_uward_tree_save()

test_ward_distance_1D_v1()

test_ward_distance_1D_v2()

test_ward_distance_2D()

test_ward_tree_save()

test_wpu_from_forged_features()
Test spatial Ward with uncertainty on forged features

class pyhrf.validation.valid_sandbox_parcellation.**StatTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_gmm_known_alpha0()
Test biGMM update with posterior weights equal to 0

test_gmm_known_weights_difvars_noise()
Test biGMM fit with known post weights, from biGMM samples (no noise) 1D case.

test_gmm_known_weights_difvars_noisea()
Test biGMM fit with known post weights, from biGMM samples (no noise) 1D case.

test_gmm_known_weights_noise()
Test biGMM fit with known post weights, from biGMM samples (no noise) 1D case.

test_gmm_known_weights_noisea()
Test biGMM fit with known post weights, from biGMM samples (no noise) 1D case.

test_gmm_known_weights_simu_1D()
Test biGMM fit with known post weights, from biGMM samples (no noise) 1D case.

test_gmm_likelihood()
Test the log likelihood computation

test_informedGMM_parameters()
Check that merge is in favour of non-activ at the same feature level, starting from singleton clusters.

test_norm_bc()

pyhrf.validation.valid_sandbox_parcellation.**create_features**(*size='2D'*, *feat_contrast='high'*, *noise_var=0.0*, *n_features=2*)

pyhrf.validation.valid_sandbox_parcellation.**simulate_fmri_data**(*scenario='high_snr'*, *output_path=None*)

1.9 pyhrf.vbjde package

1.9.1 Submodules

pyhrf.vbjde.vem_asl_models_fast module

VEM BOLD Constrained

File that contains function for BOLD data analysis with positivity and l2-norm=1 constraints.

It imports functions from vem_tools.py in pyhrf/vbjde

pyhrf.vbjde.vem_asl_models_fast.**Main_vbjde_physio**(*graph*, *Y*, *Onsets*, *durations*, *Thrf*, *K*, *TR*, *beta*, *dt*, *scale=1*, *estimateSigmaH=True*, *estimateSigmaG=True*, *sigmaH=0.05*, *sigmaG=0.05*, *gamma_h=0*, *gamma_g=0*, *NitMax=-1*, *NitMin=1*, *estimateBeta=True*, *PLOT=False*, *contrasts=[]*, *computeContrast=False*, *idx_first_tag=0*, *simulation=None*, *sigmaMu=None*, *estimateH=True*, *estimateG=True*, *estimateA=True*, *estimateC=True*, *estimateZ=True*, *estimateNoise=True*, *estimateMP=True*, *estimateLA=True*, *use_hyperprior=False*, *positivity=False*, *constraint=False*, *phy_params={'E0': 0.34*, *'tau_m': 0.98*, *'r0': 100*, *'linear': False*, *'vt0': 80.6*, *'tau_f': 2.46*, *'eps': 0.54*, *'tau_s': 1.54*, *'V0': 1*, *'alpha_w': 0.33*, *'model': 'RBM'*, *'obata': False*, *'buxton': False*, *'e': 1.43*, *'eps_max': 10.0*, *'TE': 0.018*, *'model_name': 'Khalidov11'}*, *prior='omega'*, *zc=False*)

pyhrf.vbjde.vem_asl_models_fast_ms module

VEM BOLD Constrained

File that contains function for BOLD data analysis with positivity and l2-norm=1 constraints.

It imports functions from vem_tools.py in pyhrf/vbjde

pyhrf.vbjde.vem_asl_models_fast_ms.**Main_vbjde_physio**(*graph*, *Y*, *Onsets*, *durations*, *Thrf*, *K*, *TR*, *beta*, *dt*, *scale=1*, *estimateSigmaH=True*, *estimateSigmaG=True*, *sigmaH=0.05*, *sigmaG=0.05*, *gamma_h=0*, *gamma_g=0*, *NitMax=-1*, *NitMin=1*, *estimateBeta=True*, *PLOT=False*, *contrasts=[]*, *computeContrast=False*, *idx_first_tag=0*, *simulation=None*, *sigmaMu=None*, *estimateH=True*, *estimateG=True*, *estimateA=True*, *estimateC=True*, *estimateZ=True*, *estimateNoise=True*, *estimateMP=True*, *estimateLA=True*, *use_hyperprior=False*, *positivity=False*, *constraint=False*, *phy_params={'E0': 0.34, 'tau_m': 0.98, 'r0': 100, 'linear': False, 'vt0': 80.6, 'tau_f': 2.46, 'eps': 0.54, 'tau_s': 1.54, 'V0': 1, 'alpha_w': 0.33, 'model': 'RBM', 'obata': False, 'buxton': False, 'e': 1.43, 'eps_max': 10.0, 'TE': 0.018, 'model_name': 'Khalidov11'}*, *prior='omega'*, *zc=False*)

pyhrf.vbjde.vem_bold module

This module implements the VEM for BOLD data.

The function uses the C extension for expectation and maximization steps (see src/pyhrf/vbjde/utilsmodule.c file).

See also:

pyhrf.ui.analyser_ui, *pyhrf.ui.treatment*, *pyhrf.ui.jde*, *pyhrf.ui.vb_jde_analyser*

Notes

TODO: add some refs?

pyhrf.vbjde.vem_bold.**eps**
: *float* – mimics the machine epsilon to avoid zero values

pyhrf.vbjde.vem_bold.**logger**
: *logger* – logger instance identifying this module to log informations

pyhrf.vbjde.vem_bold.**jde_vem_bold**(*graph*, *bold_data*, *onsets*, *durations*, *hrf_duration*, *nb_classes*, *tr*, *beta*, *dt*, *estimate_sigma_h=True*, *sigma_h=0.05*, *it_max=-1*, *it_min=0*, *estimate_beta=True*, *contrasts=None*, *compute_contrasts=False*, *hrf_hyperprior=0*, *estimate_hrf=True*, *constrained=False*, *zero_constraint=True*, *drifts_type='poly'*, *seed=6537546*)

This is the main function that computes the VEM analysis on BOLD data. This function uses optimized python functions.

Parameters

- **graph** (*ndarray of lists*) – represents the neighbours indexes of each voxels index
- **bold_data** (ndarray, *shape (nb_scans, nb_voxels)*) – raw data
- **onsets** (*dict*) – dictionnary of onsets
- **durations** (*# TODO*) – # TODO
- **hrf_duration** (*float*) – hrf total time duration (in s)
- **nb_classes** (*int*) – the number of classes to classify the nrls. This parameter is provided for development purposes as most of the algorithm implies two classes
- **tr** (*float*) – time of repetition
- **beta** (*float*) – the initial value of beta
- **dt** (*float*) – hrf temporal precision
- **estimate_sigma_h** (*bool, optional*) – toggle estimation of sigma H
- **sigma_h** (*float, optional*) – initial or fixed value of sigma H
- **it_max** (*int, optional*) – maximal computed iteration number
- **it_min** (*int, optional*) – minimal computed iteration number
- **estimate_beta** (*bool, optional*) – toggle the estimation of beta
- **contrasts** (OrderedDict, *optional*) – dict of contrasts to compute
- **compute_contrasts** (*bool, optional*) – if True, compute the contrasts defined in contrasts
- **hrf_hyperprior** (*float*) – # TODO
- **estimate_hrf** (*bool, optional*) – if True, estimate the HRF for each parcel, if False use the canonical HRF
- **constrained** (*bool, optional*) – if True, add a constrains the l2 norm of the HRF to 1
- **zero_constraint** (*bool, optional*) – if True, add zeros to the beginning and the end of the estimated HRF.
- **drifts_type** (*str, optional*) – set the drifts basis type used. Can be “poly” for polynomial or “cos” for cosine
- **seed** (*int, optional*) – seed used by numpy to initialize random generator number

Returns

- **loop** (*int*) – number of iterations before convergence
- **nrls_mean** (*ndarray, shape (nb_voxels, nb_conditions)*) – Neural response level mean value
- **hrf_mean** (*ndarray, shape (hrf_len,)*) – Hemodynamic response function mean value

- **hrf_covar** (*ndarray, shape (hrf_len, hrf_len)*) – Covariance matrix of the HRF
- **labels_proba** (*ndarray, shape (nb_conditions, nb_classes, nb_voxels)*) – probability of voxels being in one class
- **noise_var** (*ndarray, shape (nb_voxels,)*) – estimated noise variance
- **nrls_class_mean** (*ndarray, shape (nb_conditions, nb_classes)*) – estimated mean value of the gaussians of the classes
- **nrls_class_var** (*ndarray, shape (nb_conditions, nb_classes)*) – estimated variance of the gaussians of the classes
- **beta** (*ndarray, shape (nb_conditions,)*) – estimated beta
- **drift_coeffs** (*ndarray, shape (# TODO)*) – estimated coefficient of the drifts
- **drift** (*ndarray, shape (# TODO)*) – estimated drifts
- **contrasts_mean** (*ndarray, shape (nb_voxels, len(contrasts))*) – Contrasts computed from NRLs
- **contrasts_var** (*ndarray, shape (nb_voxels, len(contrasts))*) – Variance of the contrasts
- **compute_time** (*list*) – computation time of each iteration
- **compute_time_mean** (*float*) – computation mean time over iterations
- **nrls_covar** (*ndarray, shape (nb_conditions, nb_conditions, nb_voxels)*) – # TODO
- **stimulus_induced_signal** (*ndarray, shape (nb_scans, nb_voxels)*) – # TODO
- **mahalanobis_zero** (*float*) – Mahalanobis distance between estimated hrf_mean and the null vector
- **mahalanobis_cano** (*float*) – Mahalanobis distance between estimated hrf_mean and the canonical HRF
- **mahalanobis_diff** (*float*) – difference between mahalanobis_cano and mahalanobis_diff
- **mahalanobis_prod** (*float*) – product of mahalanobis_cano and mahalanobis_diff
- **ppm_a_nrl** (*ndarray, shape (nb_voxels,)*) – The posterior probability map using an alpha
- **ppm_g_nrl** (*ndarray, shape (nb_voxels,)*) – # TODO
- **ppm_a_contrasts** (*ndarray, shape (nb_voxels,)*) – # TODO
- **ppm_g_contrasts** (*ndarray, shape (nb_voxels,)*) – # TODO
- **variation_coeff** (*float*) – coefficient of variation of the HRF
- **free_energy** (*list*) – # TODO

Notes

See A novel definition of the multivariate coefficient of variation article for more information about the coefficient of variation.

pyhrf.vbjde.vem_bold_constrained module

VEM BOLD Constrained

File that contains function for BOLD data analysis with positivity and l2-norm=1 constraints.

It imports functions from vem_tools.py in pyhrf/vbjde

pyhrf.vbjde.vem_bold_constrained.**Main_vbjde_Extension_constrained**(*graph*, *Y*, *Onsets*, *Thrf*, *K*, *TR*, *beta*, *dt*, *scale=1*, *estimateSigmaH=True*, *sigmaH=0.05*, *NitMax=-1*, *NitMin=1*, *estimateBeta=True*, *PLOT=False*, *contrasts=[]*, *computeContrast=False*, *gamma_h=0*, *estimateHRF=True*, *TrueHrfFlag=False*, *HrfFilename='hrf.nii'*, *estimateLabels=True*, *LabelsFilename='labels.nii'*, *MFapprox=False*, *InitVar=0.5*, *InitMean=2.0*, *MiniVEMFlag=False*, *NbItMiniVem=5*)

pyhrf.vbjde.vem_bold_constrained.**Main_vbjde_Extension_constrained_stable**(*graph*, *Y*, *Onsets*, *Thrf*, *K*, *TR*, *beta*, *dt*, *scale=1*, *estimateSigmaH=True*, *sigmaH=0.05*, *NitMax=-1*, *NitMin=1*, *estimateBeta=True*, *PLOT=False*, *contrasts=[]*, *computeContrast=False*, *gamma_h=0*)

Version modified by Lofti from Christine's version

pyhrf.vbjde.vem_bold_constrained.**Main_vbjde_Python_constrained**(*graph*, *Y*, *Onsets*, *Thrf*, *K*, *TR*, *beta*, *dt*, *scale=1*, *estimateSigmaH=True*, *sigmaH=0.1*, *NitMax=-1*, *NitMin=1*, *estimateBeta=False*, *PLOT=False*)

pyhrf.vbjde.vem_bold_models_fast module

VEM BOLD Constrained

File that contains function for BOLD data analysis with positivity and l2-norm=1 constraints.

It imports functions from vem_tools.py in pyhrf/vbjde

pyhrf.vbjde.vem_bold_models_fast.**Main_vbjde_physio**(*graph*, *Y*, *Onsets*, *durations*, *Thrf*, *K*, *TR*, *beta*, *dt*, *scale=1*, *estimateSigmaH=True*, *estimateSigmaG=True*, *sigmaH=0.05*, *sigmaG=0.05*, *gamma_h=0*, *gamma_g=0*, *NitMax=-1*, *NitMin=1*, *estimateBeta=True*, *PLOT=False*, *contrasts=[]*, *computeContrast=False*, *idx_first_tag=0*, *simulation=None*, *sigmaMu=None*, *estimateH=True*, *estimateG=True*, *estimateA=True*, *estimateC=True*, *estimateZ=True*, *estimateNoise=True*, *estimateMP=True*, *estimateLA=True*, *use_hyperprior=False*, *positivity=False*, *constraint=False*, *phy_params={'E0': 0.34, 'tau_m': 0.98, 'r0': 100, 'linear': False, 'vt0': 80.6, 'tau_f': 2.46, 'eps': 0.54, 'tau_s': 1.54, 'V0': 1, 'alpha_w': 0.33, 'model': 'RBM', 'obata': False, 'buxton': False, 'e': 1.43, 'eps_max': 10.0, 'TE': 0.018, 'model_name': 'Khalidov11'}*, *prior='omega'*)

pyhrf.vbjde.vem_bold_models_fast_ms module

VEM BOLD Constrained

File that contains function for BOLD data analysis with positivity and l2-norm=1 constraints.

It imports functions from vem_tools.py in pyhrf/vbjde

pyhrf.vbjde.vem_bold_models_fast_ms.**Main_vbjde_physio**(*graph*, *Y*, *Onsets*, *durations*, *Thrf*, *K*, *TR*, *beta*, *dt*, *scale=1*, *estimateSigmaH=True*, *estimateSigmaG=True*, *sigmaH=0.05*, *sigmaG=0.05*, *gamma_h=0*, *gamma_g=0*, *NitMax=-1*, *NitMin=1*, *estimateBeta=True*, *PLOT=False*, *contrasts=[]*, *computeContrast=False*, *idx_first_tag=0*, *simulation=None*, *sigmaMu=None*, *estimateH=True*, *estimateG=True*, *estimateA=True*, *estimateC=True*, *estimateZ=True*, *estimateNoise=True*, *estimateMP=True*, *estimateLA=True*, *use_hyperprior=False*, *positivity=False*, *constraint=False*, *phy_params={'E0': 0.34, 'tau_m': 0.98, 'r0': 100, 'linear': False, 'vt0': 80.6, 'tau_f': 2.46, 'eps': 0.54, 'tau_s': 1.54, 'V0': 1, 'alpha_w': 0.33, 'model': 'RBM', 'obata': False, 'buxton': False, 'e': 1.43, 'eps_max': 10.0, 'TE': 0.018, 'model_name': 'Khalidov11'}*, *prior='omega'*, *zc=False*)

pyhrf.vbjde.vem_tools module

TOOLS and FUNCTIONS for VEM JDE Used in different versions of VEM

pyhrf.vbjde.vem_tools.**A_Entropy**(*Sigma_A*, *M*, *J*)

pyhrf.vbjde.vem_tools.**Compute_FreeEnergy**(*y_tilde*, *m_A*, *Sigma_A*, *mu_Ma*, *sigma_Ma*, *m_H*, *Sigma_H*, *AuxH*, *R*, *R_inv*, *sigmaH*, *sigmaG*, *m_C*, *Sigma_C*, *mu_Mc*, *sigma_Mc*, *m_G*, *Sigma_G*, *AuxG*, *q_Z*, *neighboursIndexes*, *Beta*, *Gamma*, *gamma*, *gamma_h*, *gamma_g*, *sigma_eps*, *XX*, *W*, *J*, *D*, *M*, *N*, *K*, *hyp*, *Gamma_X*, *Gamma_WX*, *plot=False*, *bold=False*, *S=1*)

pyhrf.vbjde.vem_tools.**H_Entropy**(*Sigma_H*, *D*)

pyhrf.vbjde.vem_tools.**PolyMat**(*Nscans*, *paramLFD*, *tr*)
Build polynomial basis

pyhrf.vbjde.vem_tools.**Q_Entropy**(*q_Z*, *M*, *J*)

pyhrf.vbjde.vem_tools.**Q_expectation_Ptilde**(*q_Z*, *neighboursIndexes*, *Beta*, *gamma*, *K*, *M*)

pyhrf.vbjde.vem_tools.**RF_Entropy**(*Sigma_RF*, *D*)

pyhrf.vbjde.vem_tools.**RF_expectation_Ptilde**(*m_X*, *Sigma_X*, *sigmaX*, *R*, *R_inv*, *D*)

pyhrf.vbjde.vem_tools.**RL_Entropy**(*Sigma_RL*, *M*, *J*)

pyhrf.vbjde.vem_tools.**RL_expectation_Ptilde**(*m_X*, *Sigma_X*, *mu_Mx*, *sigma_Mx*, *q_Z*)

pyhrf.vbjde.vem_tools.**Z_Entropy**(*q_Z*, *M*, *J*)

pyhrf.vbjde.vem_tools.**beta_gradient**(*beta*, *labels_proba*, *labels_neigh*, *neighbours_indexes*, *gamma*, *gradient_method='m1'*)

Computes the gradient of the beta function.

The maximization of $f(\beta^m)$ needs the computation of its derivative with respect to β^m.

Method 1

$$\frac{\partial f(\beta^m)}{\partial \beta^m} = -\frac{1}{2}\sum_j \sum_{k\in N(j)} \sum_{i\in\{0,1\}} \left\{ p_{mf_j}(i)p_{mf_k}(i) - \widetilde{p}_{q_j^m}(i)\widetilde{p}_{q_k^m}(i) \right\} - \lambda_\beta$$

Method 2

$$\frac{\partial f(\beta^m)}{\partial \beta^m} = -\sum_j \sum_{k\in N(j)} \sum_{i\in\{0,1\}} \widetilde{p}_{q_k^m}(i) \left\{ p_{mf_j}(i) - \frac{1}{2}\widetilde{p}_{q_j^m}(i) \right\} - \lambda_\beta$$

where

$$p_{mf_j}(i) = \frac{\exp\left(\beta \sum_{k\in N(j)} \widetilde{p}_{q_k^m}(i)\right)}{\sum_{i\in\{0,1\}} \exp\left(\beta \sum_{k\in N(j)} \widetilde{p}_{q_k^m}(i)\right)}$$

Parameters

- **beta** (*float*) –
- **labels_proba** (ndarray) –
- **labels_neigh** (ndarray) –
- **neighbours_indexes** (ndarray) –
- **gamma** (*float*) –
- **gradient_method** (*str*) – for testing purposes

Returns **gradient** – the gradient estimated in beta

Return type float

pyhrf.vbjde.vem_tools.**beta_maximization**(*beta*, *labels_proba*, *neighbours_indexes*, *gamma*)

Computes the Beta Maximization step of the JDE VEM algorithm.

The maximization over each β^m corresponds to the M-step obtained for a standard Hiddden MRF model:

$$\hat{\beta}^m = \arg\max_{\beta^m} f(\beta^m)$$

Parameters

- **beta** (ndarray) – initial value of beta

- **labels_proba** (ndarray) –
- **neighbours_indexes** (ndarray) –
- **gamma** (*float*) –

Returns

- **beta** (*float*) – the new value of beta
- **success** (*bool*) – True if the maximization has succeeded

Notes

See *beta_gradient()* function.

pyhrf.vbjde.vem_tools.**buildFiniteDiffMatrix**(*order*, *size*, *regularization=None*)
Build the finite difference matrix used for the hrf regularization prior.

Parameters

- **order** (*int*) – difference order (see numpy.diff function)
- **size** (*int*) – size of the matrix
- **regularization** (*array like, optional*) – one dimensional vector of factors used for regularizing the hrf

Returns **diffMat** – the finite difference matrix

Return type *ndarray*, shape (size, size)

pyhrf.vbjde.vem_tools.**computeFit**(*hrf_mean*, *nrls_mean*, *X*, *nb_voxels*, *nb_scans*)
Compute the estimated induced signal by each stimulus.

Parameters

- **hrf_mean** (ndarray) –
- **nrls_mean** (ndarray) –
- **X** (OrderedDict) –
- **nb_voxels** (*int*) –
- **nb_scans** (*int*) –

Returns

Return type *ndarray*

pyhrf.vbjde.vem_tools.**computeFit_asl**(*H*, *m_A*, *G*, *m_C*, *W*, *XX*)
Compute Fit

pyhrf.vbjde.vem_tools.**compute_contrasts**(*condition_names*, *contrasts*, *m_A*, *m_C*, *Sigma_A*, *Sigma_C*, *M*, *J*)

pyhrf.vbjde.vem_tools.**compute_mat_X_2**(*nbscans*, *tr*, *lhrf*, *dt*, *onsets*, *durations=None*)

pyhrf.vbjde.vem_tools.**constraint_norm1_b**(*Ftilde*, *Sigma_F*, *positivity=False*, *perfusion=None*)
Constrain with optimization strategy

pyhrf.vbjde.vem_tools.**contrasts_mean_var_classes**(*contrasts*, *condition_names*, *nrls_mean*, *nrls_covar*, *nrls_class_mean*, *nrls_class_var*, *nb_contrasts*, *nb_classes*, *nb_voxels*)

Computes the contrasts nrls from the conditions nrls and the mean and variance of the gaussian classes of the contrasts (in the cases of all inactive conditions and all active conditions).

Parameters

- **contrasts** (OrderedDict) – TODO.
- **condition_names** (*list*) – TODO.
- **nrls_mean** (ndarray, *shape (nb_voxels, nb_conditions)*) – TODO.
- **nrls_covar** (ndarray, *shape (nb_conditions, nb_conditions, nb_voxels)*) – TODO.
- **nrls_class_mean** (ndarray, *shape (nb_conditions, nb_classes)*) – TODO.
- **nrls_class_var** (ndarray, *shape (nb_conditions, nb_classes)*) – TODO.
- **nb_contrasts** (*int*) –
- **nb_classes** (*int*) –
- **nb_voxels** (*int*) –

Returns

- **contrasts_mean** (*ndarray, shape (nb_voxels, nb_contrasts)*)
- **contrasts_var** (*ndarray, shape (nb_voxels, nb_contrasts)*)
- **contrasts_class_mean** (*ndarray, shape (nb_contrasts, nb_classes)*)
- **contrasts_class_var** (*ndarray, shape (nb_contrasts, nb_classes)*)

pyhrf.vbjde.vem_tools.**cosine_drifts_basis**(*nb_scans*, *param_lfd*, *tr*)

Build cosine drifts basis.

Parameters

- **nb_scans** (*int*) –
- **param_lfd** (*int*) – TODO
- **tr** (*float*) –

Returns **drifts_basis** – K is determined by the scipy.linalg.orth function and corresponds to the effective rank of the matrix it is applied to (see function's docstring)

Return type *ndarray*, shape (nb_scans, K)

pyhrf.vbjde.vem_tools.**covariance_matrix**(*order*, *D*, *dt*)

pyhrf.vbjde.vem_tools.**create_conditions**(*onsets*, *durations*, *nb_conditions*, *nb_scans*, *hrf_len*, *tr*, *dt*)

Generate the occurrences matrix.

Parameters

- **onsets** (*dict*) – dictionary of onsets for each condition.
- **durations** (*dict*) – dictionary of durations for each condition.

- **nb_conditions** (*int*) – number of experimental conditions.
- **nb_scans** (*int*) – number of scans.
- **hrf_len** (*int*) – number of points of the hrf
- **tr** (*float*) – time of repetition
- **dt** (*float*) – hrf temporal precision

Returns

- **X** (*dict*) – dictionary of the occurence matrix
- **occurence_matrix** (*ndarray*)
- **condition_names** (*list*)

pyhrf.vbjde.vem_tools.**create_neighbours**(*graph*)

Transforms the graph list in ndarray. This is for performances purposes. Sets the empty neighbours to -1.

Parameters **graph** (*list of ndarray*) – each graph[i] represents the list of neighbours of the ith voxel

Returns **neighbours_indexes**

Return type *ndarray*

pyhrf.vbjde.vem_tools.**drifts_coeffs_fit**(*signal*, *drift_basis*)

TODO

Parameters

- **signal** (ndarray, *shape (nb_scans, nb_voxels)*) –
- **drift_basis** (ndarray, *shape (nb_scans, int)*) –

Returns **drift_coeffs**

Return type *ndarray*, shape

pyhrf.vbjde.vem_tools.**expectation_A_asl**(*H*, *G*, *m_C*, *W*, *XX*, *Gamma*, *Gamma_X*, *q_Z*, *mu_Ma*, *sigma_Ma*, *J*, *y_tilde*, *Sigma_H*, *sigma_eps_m*)

Expectation-A step:

$p_A = argmax_h(E_pc, pq, ph, pg[logp(a|y, h, c, g, q; theta)]) \propto exp(E_pc, ph, pg[logp(y|h, a, c, g; theta)] + E_pq[logp(a|q; mu_M$

Returns

Return type m_A, Sigma_A of probability distribution p_A of the current iteration

pyhrf.vbjde.vem_tools.**expectation_A_ms**(*m_A*, *Sigma_A*, *H*, *G*, *m_C*, *W*, *XX*, *Gamma*, *Gamma_X*, *q_Z*, *mu_Ma*, *sigma_Ma*, *J*, *y_tilde*, *Sigma_H*, *sigma_eps_m*, *N*, *M*, *D*, *S*)

Expectation-A step:

$p_A = argmax_h(E_pc, pq, ph, pg[logp(a|y, h, c, g, q; theta)]) \propto exp(E_pc, ph, pg[logp(y|h, a, c, g; theta)] + E_pq[logp(a|q; mu_M$

Returns

Return type m_A, Sigma_A of probability distribution p_A of the current iteration

pyhrf.vbjde.vem_tools.**expectation_C_asl**(*G*, *H*, *m_A*, *W*, *XX*, *Gamma*, *Gamma_X*, *q_Z*, *mu_Mc*, *sigma_Mc*, *J*, *y_tilde*, *Sigma_G*, *sigma_eps_m*)

Expectation-C step:

$p_C = argmax_h(E_p a, pq, ph, pg[logp(a|y,h,a,g,q;theta)]) \propto exp(E_p a, ph, pg[logp(y|h,a,c,g;theta)] + E_p q[logp(c|q;mu_M$

Returns

Return type m_C, Sigma_C of probability distribution p_C of the current iteration

pyhrf.vbjde.vem_tools.**expectation_C_ms**(*m_C*, *Sigma_C*, *G*, *H*, *m_A*, *W*, *XX*, *Gamma*, *Gamma_X*, *q_Z*, *mu_Mc*, *sigma_Mc*, *J*, *y_tilde*, *Sigma_G*, *sigma_eps_m*, *N*, *M*, *D*, *S*)

Expectation-C step:

$p_C = argmax_h(E_p a, pq, ph, pg[logp(a|y,h,a,g,q;theta)]) \propto exp(E_p a, ph, pg[logp(y|h,a,c,g;theta)] + E_p q[logp(c|q;mu_M$

Returns

Return type m_C, Sigma_C of probability distribution p_C of the current iteration

pyhrf.vbjde.vem_tools.**expectation_G_asl**(*Sigma_C*, *m_C*, *m_A*, *H*, *XX*, *W*, *WX*, *Gamma*, *Gamma_WX*, *XW_Gamma_WX*, *J*, *y_tilde*, *cov_noise*, *R_inv*, *sigmaG*, *prior_mean_term*, *prior_cov_term*)

Expectation-G step:

$p_G = argmax_g(E_p a, pc, ph[logp(g|y,a,c,h;theta)]) \propto exp(E_p a, pc, ph[logp(y|h,a,c,g;theta) + logp(g;sigmaG)])$

Returns

Return type m_G, Sigma_G of probability distribution p_G of the current iteration

pyhrf.vbjde.vem_tools.**expectation_G_ms**(*Sigma_C*, *m_C*, *m_A*, *H*, *XX*, *W*, *WX*, *Gamma*, *Gamma_WX*, *XW_Gamma_WX*, *J*, *y_tilde*, *cov_noise*, *R_inv*, *sigmaG*, *prior_mean_term*, *prior_cov_term*, *N*, *M*, *D*, *S*)

Expectation-G step:

$p_G = argmax_g(E_p a, pc, ph[logp(g|y,a,c,h;theta)]) \propto exp(E_p a, pc, ph[logp(y|h,a,c,g;theta) + logp(g;sigmaG)])$

Returns

Return type m_G, Sigma_G of probability distribution p_G of the current iteration

pyhrf.vbjde.vem_tools.**expectation_H_asl**(*Sigma_A*, *m_A*, *m_C*, *G*, *XX*, *W*, *Gamma*, *Gamma_X*, *X_Gamma_X*, *J*, *y_tilde*, *cov_noise*, *R_inv*, *sigmaH*, *prior_mean_term*, *prior_cov_term*)

Expectation-H step:

$p_H = argmax_h(E_p a, pc, pg[logp(h|y,a,c,g;theta)]) \propto exp(E_p a, pc, pg[logp(y|h,a,c,g;theta) + logp(h;sigmaH)])$

Returns

Return type m_H, Sigma_H of probability distribution p_H of the current iteration

pyhrf.vbjde.vem_tools.**expectation_H_ms**(*Sigma_A*, *m_A*, *m_C*, *G*, *XX*, *W*, *Gamma*, *Gamma_X*, *X_Gamma_X*, *J*, *y_tilde*, *cov_noise*, *R_inv*, *sigmaH*, *prior_mean_term*, *prior_cov_term*, *N*, *M*, *D*, *S*)

Expectation-H step:

$p_H = argmax_h(E_p a, pc, pg[logp(h|y,a,c,g;theta)]) \propto exp(E_p a, pc, pg[logp(y|h,a,c,g;theta) + logp(h;sigmaH)])$

Returns

Return type m_H, Sigma_H of probability distribution p_H of the current iteration

pyhrf.vbjde.vem_tools.**expectation_H_ms_concat**(*Sigma_A*, *m_A*, *m_C*, *G*, *XX*, *W*, *Gamma*, *Gamma_X*, *X_Gamma_X*, *J*, *y_tilde*, *cov_noise*, *R_inv*, *sigmaH*, *prior_mean_term*, *prior_cov_term*, *S*)

Expectation-H step:

$p_H = argmax_h(E_p a, pc, pg[logp(h|y, a, c, g; theta)]) \propto exp(E_p a, pc, pg[logp(y|h, a, c, g; theta) + logp(h; sigmaH)])$

Returns

Return type m_H, Sigma_H of probability distribution p_H of the current iteration

pyhrf.vbjde.vem_tools.**expectation_Ptilde_Likelihood**(*y_tilde*, *m_A*, *Sigma_A*, *H*, *Sigma_H*, *m_C*, *Sigma_C*, *G*, *Sigma_G*, *XX*, *W*, *sigma_eps*, *Gamma*, *J*, *D*, *M*, *N*, *Gamma_X*, *Gamma_WX*)

pyhrf.vbjde.vem_tools.**expectation_Q_asl**(*Sigma_A*, *m_A*, *Sigma_C*, *m_C*, *sigma_Ma*, *mu_Ma*, *sigma_Mc*, *mu_Mc*, *Beta*, *p_q_t*, *p_Q*, *neighbours_indexes*, *graph*, *M*, *J*, *K*)

pyhrf.vbjde.vem_tools.**expectation_Q_async_asl**(*Sigma_A*, *m_A*, *Sigma_C*, *m_C*, *sigma_Ma*, *mu_Ma*, *sigma_Mc*, *mu_Mc*, *Beta*, *p_q_t*, *p_Q*, *neighbours_indexes*, *graph*, *M*, *J*, *K*)

pyhrf.vbjde.vem_tools.**expectation_Q_ms**(*Sigma_A*, *m_A*, *Sigma_C*, *m_C*, *sigma_Ma*, *mu_Ma*, *sigma_Mc*, *mu_Mc*, *Beta*, *p_q_t*, *p_Q*, *neighbours_indexes*, *graph*, *M*, *J*, *K*, *S*)

pyhrf.vbjde.vem_tools.**expectation_ptilde_hrf**(*hrf_mean*, *hrf_covar*, *sigma_h*, *hrf_regu_prior*, *hrf_regu_prior_inv*, *hrf_len*)

Expectation with respect to p_tilde hrf.

$$\mathrm{E}_{\widetilde{p}_h}\left[\log p(h|\sigma_h)\right] = -\frac{D+1}{2}\log 2\pi - \frac{D-1}{2}\log\sigma_h - \frac{\log|\mathbf{R}|}{2} - \frac{m_h^t\mathbf{R}^{-1}m_h + \mathrm{tr}\left(\Sigma_h\mathbf{R}^{-1}\right)}{2\sigma_h}$$

pyhrf.vbjde.vem_tools.**expectation_ptilde_labels**(*labels_proba*, *neighbours_indexes*, *beta*, *nb_conditions*, *nb_classes*)

Expectation with respect to p_tilde q (or z).

$$\mathrm{E}_{\widetilde{p}_q}\left[\log p(q|\beta)\right] = \sum_m \left\{ -\sum_j \left\{ \log\left(\sum_{i=0}^{1} \exp\left(\beta^m \sum_{k\in N(j)} \widetilde{p}_{q_k^m}(i) \right) \right) \right\} \right.$$
$$\left. -\beta^m \sum_j \sum_{k\in N(j)} \sum_{i=0}^{1} \left[p_j^{MF}(i) \left(\frac{p_k^{MF}(i)}{2} - \widetilde{p}_{q_k^m}(i) \right) - \frac{1}{2}\widetilde{p}_{q_j^m}(i)\widetilde{p}_{q_k^m}(i) \right] \right\}$$

pyhrf.vbjde.vem_tools.**expectation_ptilde_likelihood**(*data_drift*, *nrls_mean*, *nrls_covar*, *hrf_mean*, *hrf_covar*, *occurence_matrix*, *noise_var*, *noise_struct*, *nb_voxels*, *nb_scans*)

Expectation with respect to likelihood.

$$\mathrm{E}_{\widetilde{p}_a \widetilde{p}_h \widetilde{p}_q}\left[\log p(y|a,h,q;\theta)\right] = -\frac{NJ}{2}\log 2\pi + \frac{J}{2}\log|\Lambda_j| - N\sum_{j\in J}\log v_{b_j} + \frac{1}{2v_{b_j}}\sum_{j\in J} V_j$$

where

$$V_j = \widetilde{m}_{a_j}^t \mathbf{X}_h^t \Lambda_j \mathbf{X}_h \widetilde{m}_{a_j} + \mathrm{tr}\left(\Sigma_{a_j}\mathbf{X}_h^t\Lambda_j\mathbf{X}_h\right) - 2\widetilde{m}_{a_j}^t\mathbf{X}_h^t\Lambda_j\left(y_j - \mathbf{P}\ell_j\right)$$

Parameters

- **data_drift** (*ndarray, shape (nb_scans, nb_voxels)*) – This is the BOLD data minus the drifts (y_tilde in the paper)
- **nrls_mean** (*ndarray, shape (nb_voxels, nb_conditions)*) –
- **nrls_covar** (*ndarray, shape (nb_conditions, nb_conditions, nb_voxels)*) –
- **hrf_mean** (*ndarray, shape (hrf_len,)*) –
- **hrf_covar** (*ndarray, shape (hrf_len, hrf_len)*) –
- **occurence_matrix** (*ndarray, shape (nb_conditions, nb_scans, hrf_len)*) –
- **noise_var** (*ndarray, shape (nb_voxels,)*) –
- **noise_struct** (*ndarray, shape (nb_scans, nb_scans)*) –
- **nb_voxels** (*int*) –
- **nb_scans** (*int*) –

Returns ptilde_likelihood

Return type float

pyhrf.vbjde.vem_tools.**expectation_ptilde_nrls**(*labels_proba*, *nrls_class_mean*, *nrls_class_var*, *nrls_mean*, *nrls_covar*)

Expectation with respect to p_tilde a.

$$\mathrm{E}_{\widetilde{p}_a\widetilde{p}_q}[\log p(a|q,\theta_a)] = \sum_m\sum_j \left\{ \left[1 - \widetilde{p}_{q_j^m}(1)\right]\left[\log\frac{1}{\sqrt{2\pi\sigma_0^{2m}}} - \frac{\left(m_{a_j^m} - \mu_0^m\right)^2 + \Sigma_{a_j^{m,m}}}{2\sigma_0^{2m}}\right] + \widetilde{p}_{q_j^m}(1)\left[\log\frac{1}{\sqrt{2\pi\sigma_1^{2m}}} - \frac{\left(m_{a_j^m} - \mu_1^m\right)^2 + \Sigma_{a_j^{m,m}}}{2\sigma_1^{2m}}\right]\right\}$$

pyhrf.vbjde.vem_tools.**fit_hrf_two_gammas**(*hrf_mean*, *dt*, *duration*)

Fits the estimated HRF to the standard two gammas model.

Parameters

- **dt** (*float*) –
- **duration** (*float*) –
- **hrf_mean** (*ndarray, shape (hrf_len,)*) –

Returns

- **delay_of_response** (*float*)

- **delay_of_undershoot** (*float*)
- **dispersion_of_response** (*float*)
- **dispersion_of_undershoot** (*float*)
- **ratio_resp_under** (*float*)
- **delay** (*float*)

pyhrf.vbjde.vem_tools.**free_energy_computation**(*nrls_mean*, *nrls_covar*, *hrf_mean*, *hrf_covar*, *hrf_len*, *labels_proba*, *data_drift*, *occurence_matrix*, *noise_var*, *noise_struct*, *nb_conditions*, *nb_voxels*, *nb_scans*, *nb_classes*, *nrls_class_mean*, *nrls_class_var*, *neighbours_indexes*, *beta*, *sigma_h*, *hrf_regu_prior*, *hrf_regu_prior_inv*, *gamma*, *hrf_hyperprior*)

Compute the free energy functional.

$$\mathcal{F}(q,\theta) = \mathrm{E}_q\left[\log p(y, A, H, Z; \theta)\right] + \mathcal{G}(q)$$

where $E_q[\cdot]$ denotes the expectation with respect to q and $\mathcal{G}(q)$ is the entropy of q.

Returns **free_energy**

Return type float

pyhrf.vbjde.vem_tools.**fun**(*Beta*, *p_Q*, *Qtilde_sumneighbour*, *neighboursIndexes*, *gamma*)

function to minimize

pyhrf.vbjde.vem_tools.**grad_fun**(*Beta*, *p_Q*, *Qtilde_sumneighbour*, *neighboursIndexes*, *gamma*)

function to minimize

pyhrf.vbjde.vem_tools.**hrf_entropy**(*hrf_covar*, *hrf_len*)

Compute the entropy of the hemodynamic response function. The entropy of a multivariate normal distribution is

$$\ln\left(\sqrt{(2\pi e)^n\,|\Sigma|}\right)$$

where *n* is the dimensionality of the vector space and $|\Sigma|$ is the determinant of the covariance matrix.

Parameters

- **hrf_covar** (ndarray, *shape (hrf_len, hrf_len)*) – Covariance matrix of the HRF
- **hrf_len** (*int*) – size of the HRF

Returns **entropy**

Return type float

pyhrf.vbjde.vem_tools.**hrf_expectation**(*nrls_covar*, *nrls_mean*, *occurence_matrix*, *noise_struct*, *hrf_regu_prior_inv*, *sigmaH*, *nb_voxels*, *y_tilde*, *noise_var*, *prior_mean_term=0.0*, *prior_cov_term=0.0*)

Computes the VE-H step of the JDE-VEM algorithm.

$$m_H^{(r)} = \Sigma_H^{(r)} \left(\sum_{i \in \mathcal{P}} \widetilde{S}_i^t \widetilde{y}_i^{(r)} \right)$$

$$\left(\Sigma_H^{(r)} \right)^{-1} = \frac{\mathbf{R}^{-1}}{\sigma_h^{2(r)}} + \sum_{i \in \mathcal{P}} \left(\sum_{m,m'} \sigma^{(r-1)}_{A_{mi} A_{m'i}} \mathbf{X}_m^t \Gamma_i^{(r)} \mathbf{X}_{m'} + \widetilde{S}_i^t \Gamma_i^{(r)} \widetilde{S}_i \right)$$

where

$$\widetilde{S}_i = \sum_{m=1}^{M} m^{(r-1)}_{A_{mi}} \mathbf{X}_m$$

$$\widetilde{y}_i^{(r)} = \Gamma_i^{(r)} \left(y_i - \mathbf{P} \ell_i^{(r)} \right)$$

Here, $m^{(r-1)}_{A_{mi}}$ and $\sigma^{(r-1)}_{A_{mi} A_{m'i}}$ denote the m^{th} and $(m, m')^{th}$ entries of the mean vector and covariance matrix of the current $q^{(r-1)}_{A_i}$, respectively.

Parameters

- **nrls_covar** (ndarray, *shape (nb_conditions, nb_conditions, nb_voxels)*) –
- **nrls_mean** (ndarray, *shape (nb_voxels, nb_conditions)*) –
- **occurence_matrix** (ndarray, *shape (nb_conditions, nb_scans, hrf_len)*) –
- **noise_struct** (ndarray, *shape (nb_scans, nb_scans)*) –
- **hrf_regu_prior_inv** (ndarray, *shape (hrf_len, hrf_len)*) – inverse of the hrf regularization prior matrix *R*
- **sigmaH** (*float*) –
- **nb_voxels** (*int*) –
- **y_tilde** (ndarray, *shape (nb_scans, nb_voxels)*) –
- **noise_var** (ndarray, *shape (nb_voxels,)*) –
- **prior_mean_term** (*float, optional*) –
- **prior_cov_term** (*float, optional*) –

Returns

- **hrf_mean** (*ndarray, shape (hrf_len,)*)
- **hrf_covar** (*ndarray, shape (hrf_len, hrf_len)*)

pyhrf.vbjde.vem_tools.**labels_entropy**(*labels_proba*)

Compute the labels entropy.

$$-\sum_{x=0}^{1} p_Q(x) \log p_Q(x)$$

Parameters **labels_proba** (ndarray, *shape (nb_conditions, nb_classes, nb_voxels)*) – Probability of each voxel to be in one class

Returns entropy

Return type float

pyhrf.vbjde.vem_tools.**labels_expectation**(*nrls_covar*, *nrls_mean*, *nrls_class_var*, *nrls_class_mean*, *beta*, *labels_proba*, *neighbours_indexes*, *nb_conditions*, *nb_classes*, *nb_voxels=None*, *parallel=True*, *nans_init=False*)

Computes the E-Z (or E-Q) step of the JDE-VEM algorithm.

Using the *mean-field* approximation, $\widetilde{p}^{(r)}_{Q^m}(q^m)$ is approximated by a factorized density $\widetilde{p}^{(r)}_{Q^m}(q^m) = \prod_{j \in \mathcal{P}_\gamma} \widetilde{p}^{(r)}_{Q^m_j}(q^m_j)$ such that if $q^m_j = i$, then $\widetilde{p}^{(r)}_{Q^m_j}(i) \propto \mathcal{N}\left(m^{(r)}_{A^m_j}; \mu^{(r-1)}_{im}, v^{(r-1)}_{im}\right) f\left(q^m_j = i \,|\, \widetilde{q}^m_{\sim j}; \beta^{(r-1)}_m, v^{(r-1)}_m\right)$ where $\widetilde{q}^m$ is a particular configuration of q^m updated at each iteration according to a specific scheme and

$$f\left(q^m_j \,|\, \widetilde{q}^m_{\sim j}; \beta^{(r-1)}_m, v^{(r-1)}_m\right) \propto \exp\left(\alpha^{m(r)}_j(q^m_j) + \beta^{(r-1)}_m \sum_{k \sim j} I\left(\widetilde{q}^m_k = q^m_j\right)\right)$$

where

$$\left\{\alpha^{m(r)}_j = \left(-v^{(r)}_{A^m_j A^{m\prime\prime}_j}\left[\frac{1}{v^{(r-1)}_{0m}}, \frac{1}{v^{(r-1)}_{1m}}\right]\right)^t, j \in \mathcal{P}_\gamma\right\}$$

and $v^{(r)}_{A^m_j A^{m\prime\prime}_j}$ denotes the (m, m') entries of the covariance matrix $\Sigma^{(r)}_{A_j}$

Notes

The *mean-field fixed point* equation is defined in:

> Celeux, G., Forbes, F., & Peyrard, N. (2003). EM procedures using mean field-like approximations for Markov model-based image segmentation. Pattern Recognition, 36(1), 131–144. https://doi.org/10.1016/S0031-3203(02)00027-4

Parameters

- **nrls_covar** (ndarray, *shape (nb_conditions, nb_conditions, nb_voxels)*) –
- **nrls_mean** (ndarray, *shape (nb_voxels, nb_conditions)*) –
- **nrls_class_var** (ndarray, *shape (nb_conditions, nb_classes)*) –
- **nrls_class_mean** (ndarray, *shape (nb_conditions, nb_classes)*) –
- **beta** (ndarray, *shape*) –
- **labels_proba** (ndarray, *shape (nb_conditions, nb_classes, nb_voxels)*) –
- **neighbours_indexes** (ndarray, *shape (nb_voxels, max(len(a) for a in graph))*) – This is the version of graph array where arrays from graph smaller than the maximum ones are filled with -1
- **nb_conditions** (*int*) –
- **nb_classes** (*int*) –
- **nb_voxels** (*int*) –

- **parallel** (*bool*) –
- **nans_init** (*bool*) –

Returns **labels_proba**

Return type *ndarray*, shape (nb_conditions, nb_classes, nb_voxels)

pyhrf.vbjde.vem_tools.**maximization_LA_asl**(*Y*, *m_A*, *m_C*, *XX*, *WP*, *W*, *WP_Gamma_WP*, *H*, *G*, *Gamma*)

pyhrf.vbjde.vem_tools.**maximization_Mu_asl**(*H*, *G*, *matrix_covH*, *matrix_covG*, *sigmaH*, *sigmaG*, *sigmaMu*, *Omega*, *R_inv*)

pyhrf.vbjde.vem_tools.**maximization_beta_m2_asl**(*beta*, *p_Q*, *Qtilde_sumneighbour*, *Qtilde*, *neighboursIndexes*, *maxNeighbours*, *gamma*, *MaxItGrad*, *gradientStep*)

pyhrf.vbjde.vem_tools.**maximization_beta_m2_scipy_asl**(*Beta*, *p_Q*, *Qtilde_sumneighbour*, *Qtilde*, *neighboursIndexes*, *maxNeighbours*, *gamma*, *MaxItGrad*, *gradientStep*)

Maximize beta

pyhrf.vbjde.vem_tools.**maximization_beta_m4_asl**(*beta*, *p_Q*, *Qtilde_sumneighbour*, *Qtilde*, *neighboursIndexes*, *maxNeighbours*, *gamma*, *MaxItGrad*, *gradientStep*)

pyhrf.vbjde.vem_tools.**maximization_class_proba**(*labels_proba*, *nrls_mean*, *nrls_covar*)

Computes the M-(mu, sigma) step of the JDE-VEM algorithm.

$$\bar{q}_{mk}^{(r)} = \sum_{i \in \mathcal{P}} q_{Z_{mi}}^{(r)}(k)$$

$$\mu_{mk}^{(r+1)} = \frac{\sum_{i \in \mathcal{P}} q_{Z_{mi}}^{(r)}(k) m_{A_{mi}}^{(r)}}{\bar{q}_{mk}^{(r)}}$$

$$\sigma_{mk}^{2(r+1)} = \frac{\sum_{i \in \mathcal{P}} q_{Z_{mi}}^{(r)}(k) \left(\left(m_{A_{mi}}^{(r)} - \mu_{mk}^{(r+1)} \right)^2 + \sigma_{A_{im}A_{im}}^{(r)} \right)}{\bar{q}_{mk}^{(r)}}$$

pyhrf.vbjde.vem_tools.**maximization_drift_coeffs**(*data*, *nrls_mean*, *occurence_matrix*, *hrf_mean*, *noise_struct*, *drift_basis*)

Computes the M-(l, Gamma) step of the JDE-VEM algorithm. In the AR(1) case:

$$\ell_j^{(r)} = \left(P^\intercal \Lambda_j^{(r)} P \right)^{-1} P^\intercal \Lambda_j^{(r)} \left(y_j - \widetilde{S}_j m_H^{(r)} \right)$$

pyhrf.vbjde.vem_tools.**maximization_mu_sigma_asl**(*q_Z*, *m_X*, *Sigma_X*)

pyhrf.vbjde.vem_tools.**maximization_mu_sigma_ms**(*q_Z*, *m_X*, *Sigma_X*, *M*, *J*, *S*, *K*)

pyhrf.vbjde.vem_tools.**maximization_noise_var**(*occurence_matrix*, *hrf_mean*, *hrf_covar*, *nrls_mean*, *nrls_covar*, *noise_struct*, *data_drift*, *nb_scans*)

Computes the M-sigma_epsilon step of the JDE-VEM algorithm.

$$\sigma_j^{2(r)} = \frac{1}{N} \left(\mathrm{E}_{\widetilde{p}_{A_j}^{(r)}} \left[a_j^t \widetilde{\Lambda}_j^{(r)} a_j \right] - 2 \left(m_{A_j}^{(r)} \right)^t \widetilde{G}_j^{(r)} y_j^{(r)} + \left(y_j^{(r)} \right)^t \Lambda_j^{(r)} y_j^{(r)} \right)$$

where matrix $\widetilde{\Lambda}_j^{(r)} = \mathrm{E}_{\widetilde{p}_H^{(r)}}\left[G^t \Lambda_j^{(r)} G\right]$ is a $M \times M$ whose element (m, m') is given by

$$\widetilde{g}_m^t \Lambda_j^{(r)} \widetilde{g}_{m'} + \mathrm{tr}\left(\Lambda_j^{(r)} X_m \Sigma_H^{(r)} X_{m'}^t\right)$$

pyhrf.vbjde.vem_tools.**maximization_sigmaH**(*D*, *Sigma_H*, *R*, *m_H*)
Computes the M-sigma_h step of the JDE-VEM algorithm.

$$\sigma_h^{2(r+1)} = \frac{\mathrm{tr}((\Sigma_H^{(r)} + m_H^{(r)}(m_H^{(r)})^\intercal)\mathbf{R}^{-1})}{D-1}$$

pyhrf.vbjde.vem_tools.**maximization_sigmaH_prior**(*D*, *Sigma_H*, *R*, *m_H*, *gamma_h*)
Computes the M-sigma_h step of the JDE-VEM algorithm with a prior.

$$\sigma_h^{(r)} = \frac{(D-1) + \sqrt{8\lambda_{\sigma_h} C + (D-1)^2}}{4\lambda_{\sigma_h}}$$

where

$$C = \mathrm{tr}\left(\left(\Sigma_H^{(r)} + m_H^{(r)}(m_H^{(r)})^\intercal\right)\mathbf{R}^{-1}\right)$$

pyhrf.vbjde.vem_tools.**maximization_sigma_asl**(*D*, *Sigma_H*, *R_inv*, *m_H*, *use_hyp*, *gamma_h*)

pyhrf.vbjde.vem_tools.**maximization_sigma_noise_asl**(*XX*, *m_A*, *Sigma_A*, *H*, *m_C*, *Sigma_C*, *G*, *Sigma_H*, *Sigma_G*, *W*, *y_tilde*, *Gamma*, *Gamma_X*, *Gamma_WX*, *N*)
Maximization sigma_noise

pyhrf.vbjde.vem_tools.**maximum**(*iterable*)
Return the maximum and the indice of the maximum of an iterable.

Parameters **iterable** (*iterable or numpy array*) –

Returns

- **iter_max** (*the maximum*)
- **iter_max_indice** (*the indice of the maximum*)

pyhrf.vbjde.vem_tools.**mult**(*v1*, *v2*)
Multiply two vectors.

The first vector is made vertical and the second one horizontal. The result will be a matrix of size len(v1), len(v2).

Parameters

- **v1** (ndarray) – unidimensional
- **v2** (ndarray) – unidimensional

Returns **x**

Return type *ndarray*, shape (*len*(v1), *len*(v2))

pyhrf.vbjde.vem_tools.**norm1_constraint**(*function*, *variance*)
Returns the function constrained with optimization strategy.

Parameters

- **function** (*array_like*) – function to optimize under norm1 constraint

- **variance** (*array_like*) – variance of the *function*, must be the same size

Returns optimized_function

Return type numpy array

Raises ValueError – If *len(variance) != len(function)*

pyhrf.vbjde.vem_tools.**normpdf**(*x*, *mu*, *sigma*)

pyhrf.vbjde.vem_tools.**nrls_entropy**(*nrls_covar*, *nb_conditions*)

Compute the entropy of neural response levels. The entropy of a multivariate normal distribution is

$$\ln\left(\sqrt{(2\pi e)^n\,|\Sigma|}\right)$$

where *n* is the dimensionality of the vector space and $|\Sigma|$ is the determinant of the covariance matrix.

Parameters

- **nrls_covar** (ndarray, *shape (nb_conditions, nb_conditions, nb_voxels)*) – Covariance of the NRLs
- **nb_conditions** (*int*) –

Returns entropy

Return type float

pyhrf.vbjde.vem_tools.**nrls_expectation**(*hrf_mean*, *nrls_mean*, *occurence_matrix*, *noise_struct*, *labels_proba*, *nrls_class_mean*, *nrls_class_var*, *nb_conditions*, *y_tilde*, *nrls_covar*, *hrf_covar*, *noise_var*)

Computes the VE-A step of the JDE-VEM algorithm.

$$\Sigma_{A_j}^{(r)} = \left(\sum_{i=1}^{I} \Delta_{ij} + \widetilde{H}_j\right)^{-1}$$

$$m_{A_j}^{(r)} = \Sigma_{Aj}^{(r)}\left(\sum_{i=1}^{I} \Delta_{ij}\mu_i^{(r-1)} + \widetilde{G}^t\Gamma_j^{(r-1)}\left(y_j - P\ell^{(r-1)}\right)\right)$$

where:

The mth column of $\widetilde{G}$ is denote by $\widetilde{g}_m = X_m m_H^{(r)} \in \mathbb{R}^N$

$$\Delta_{ij} = \mathrm{diag}_M\left[\frac{q_{Z_{mj}}^{(r-1)}(i)}{\sigma_{mi}^{2(r)}}\right]$$

$$\widetilde{H}_j = \widetilde{g}_m^t\Gamma_j^{(r-1)}\widetilde{g}_{m'} + \mathrm{tr}\left(\Gamma_j^{(r-1)}X_m\Sigma_H^{(r)}X_{m'}^t\right)$$

Parameters

- **hrf_mean** (ndarray, *shape (hrf_len,)*) –
- **nrls_mean** (ndarray, *shape (nb_voxels, nb_conditions)*) –
- **occurence_matrix** (ndarray, *shape (nb_conditions, nb_scans, hrf_len)*) –
- **noise_struct** (ndarray, *shape (nb_scans, nb_scans)*) –

- **labels_proba** (ndarray, *shape (nb_conditions, nb_classes, nb_voxels)*) –
- **nrls_class_mean** (ndarray, *shape (nb_conditions, nb_classes)*) –
- **nrls_class_var** (ndarray, *shape (nb_conditions, nb_classes)*) –
- **nb_conditions** (*int*) –
- **y_tilde** (ndarray, *shape (nb_scans, nb_voxels)*) – BOLD data minus drifts
- **nrls_covar** (ndarray, *shape (nb_conditions, nb_conditions, nb_voxels)*) –
- **hrf_covar** (ndarray, *shape (hrf_len, hrf_len)*) –
- **noise_var** (ndarray, *shape (nb_voxels,)*) –

Returns

- **nrls_mean** (*ndarray, shape (nb_voxels, nb_conditions)*)
- **nrls_covar** (*ndarray, shape (nb_conditions, nb_conditions, nb_voxels)*)

pyhrf.vbjde.vem_tools.**plot_convergence**(*ni*, *M*, *cA*, *cC*, *cH*, *cG*, *cAH*, *cCG*, *SUM_q_Z*, *mua1*, *muc1*, *FE*)

pyhrf.vbjde.vem_tools.**plot_response_functions_it**(*ni*, *NitMin*, *M*, *H*, *G*, *Mu=None*, *prior=None*)

pyhrf.vbjde.vem_tools.**polyFit**(*signal*, *tr*, *order*, *p*)

pyhrf.vbjde.vem_tools.**poly_drifts_basis**(*nb_scans*, *param_lfd*, *tr*)

Build polynomial drifts basis.

Parameters

- **nb_scans** (*int*) –
- **param_lfd** (*int*) – TODO
- **tr** (*float*) –

Returns **drifts_basis** – K is determined by the scipy.linalg.orth function and corresponds to the effective rank of the matrix it is applied to (see function's docstring)

Return type *ndarray*, shape (nb_scans, K)

pyhrf.vbjde.vem_tools.**ppm_contrasts**(*contrasts_mean*, *contrasts_var*, *contrasts_class_mean*, *contrasts_class_var*, *threshold_a='std_inact'*, *threshold_g=0.95*)

Computes the ppm for the given contrast using either the standard deviation of the "all inactive conditions" class gaussian (default) or the intersection of the [all inactive conditions] and [all active conditions] classes gaussians as threshold for the PPM_a and 0.95 (default) for the PPM_g. Be carefull, this computation considers the mean of the inactive class as zero.

Parameters

- **contrasts_mean** (ndarray, *shape (nb_voxels, nb_contrasts)*) –
- **contrasts_var** (ndarray, *shape (nb_voxels, nb_contrasts)*) –
- **contrasts_class_mean** (ndarray, *shape (nb_contrasts, nb_classes)*) –

- **contrasts_class_var** (ndarray, *shape (nb_contrasts, nb_classes)*) –
- **threshold_a** (*str, optional*) – if "std_inact" (default) uses the standard deviation of the [all inactive conditions] gaussian class as PPM_a threshold, if "intersect" uses the intersection of the [all inactive/all active conditions] gaussian classes
- **threshold_g** (*float, optional*) – the threshold of the PPM_g

Returns

- **ppm_a_contrasts** (*ndarray, shape (nb_voxels, nb_contrasts)*)
- **ppm_g_contrasts** (*ndarray, shape (nb_voxels, nb_contrasts)*)

pyhrf.vbjde.vem_tools.**ppms_computation**(*elements_mean*, *elements_var*, *class_mean*, *class_var*, *threshold_a='std_inact'*, *threshold_g=0.9*)

Considering the *elements_mean* and *elements_var* from a gaussian distribution, commutes the posterior probability maps considering for the alpha threshold, either the standard deviation of the [all inactive conditions] gaussian class or the intersection of the [all (in)active conditions] gaussian classes; and for the gamma threshold 0.9 (default).

The posterior probability maps (PPM) per experimental condition is computed as $p(a_j^m > \delta | y_j) > \alpha$. Note that we have to thresholds to set. We set δ to get a posterior probability distribution, and α is the threshold that we set to see a certain level of significance. As default, we chose a threshold δ for each experimental condition *m* as the intersection of the two Gaussian densities of the Gaussian Mixture Model (GMM) that represent active and non-active voxel.

$$\frac{(\delta - \mu_1^m)^2}{v_1^m} - \frac{(\delta - \mu_0^m)^2}{v_0^m} = \log\left(\frac{v_1^m}{v_0^m}\right)$$

μ_i^m and v_i^m being the parameters of the GMM in a_j^m corresponding to active (*i=0*) and non-active (*i=1*) voxels for experimental condition *m*.

$$\delta = \frac{\mu_1\sigma_0^2 \pm \sigma_0\sigma_1\sqrt{\mu_1^2 + 2\left(\sigma_0^2 - \sigma_1^2\right)\log\left(\frac{\sigma_0}{\sigma_1}\right)}}{\sigma_0^2 - \sigma_1^2}$$

Be careful, this computation considers the mean of the inactive class as zero.

Notes

nb_elements refers either to the number of contrasts (for the PPMs contrasts computation) or for the number of conditions (for the PPMs nrls computation).

Parameters

- **elements_mean** (ndarray, *shape (nb_voxels, nb_elements)*) –
- **elements_var** (ndarray, *shape (nb_voxels, nb_elements)*) –
- **class_mean** (ndarray, *shape (nb_elements, nb_classes)*) –
- **class_var** (ndarray, *shape (nb_elements, nb_classes)*) –
- **threshold_a** (*str, optional*) – if "std_inact" (default) uses the standard deviation of the [all inactive conditions] gaussian class as PPM_a threshold. If "intersect" uses the intersection of the [all inactive/all active conditions] gaussian classes.
- **threshold_g** (*float, optional*) – the threshold of the PPM_g

Returns

- **ppm_a** (*ndarray, shape (nb_voxels, nb_elements)*)
- **ppm_g** (*ndarray, shape (nb_voxels, nb_elements)*)

pyhrf.vbjde.vem_tools.**roc_curve**(*dvals*, *labels*, *rocN=None*, *normalize=True*)
Compute ROC curve coordinates and area

- *dvals* - a list with the decision values of the classifier
- *labels* - list with class labels, in {0, 1}

returns (FP coordinates, TP coordinates, AUC)

pyhrf.vbjde.vem_tools.**sum_over_neighbours**(*neighbours_indexes*, *array_to_sum*)
Sums the *array_to_sum* over the neighbours in the graph.

1.10 pyhrf.xmliobak package

1.10.1 Submodules

pyhrf.xmliobak.xmlbase module

class pyhrf.xmliobak.xmlbase.**FuncWrapper**(*func*, *params=None*)

class pyhrf.xmliobak.xmlbase.**TypedXMLHandler**(*write_callback=None*)
Class handling the xml format with the following generic document structure:

```
<root>
    <tagName 'type'=tagType>
        tagContent
    </tagName>
</root>
```

The root tag is mandatory, so is the 'type' attribute for every other tag. This class can parse an xml string and build a dictionary of python objects according to each tag (see parseXMLString()). Conversely, it can build the xml string corresponding to a list or dictionary of objects (see to_xml()). This class is based on the xml.dom python module and relies on the DOM structure to parse XML. XML input/output is handled via a mapping between a type attribute of a tag and a static handling function. This class handles the following basic python types: string, int, float, array.array, list, dict. One can add other type-specific read or write handlers with the functions addDOMTagReader() and addDOMWriter().

Reading XML:

- specific handlers are added with method addDOMTagReader(stype, myReadHandler) which maps the function myReadHandler to the string stype.
- a tag reading handler must have the following prototype:

```
myReadHandler(domTreeWalker):
    # interpret and process tag content
    # return built python object
```

, where domTreeWalker is an instance of the _xmlplus.dom.TreeWalker.TreeWalker class.

- useful things to use the domTreeWalker and implement a handler:

  - node = domTreeWalker.currentNode -> current node in the tree parsing, of class Node, corresponding to the current tag.
  - node.getAttribute('my attribute') -> return the string corresponding to 'my attribute' in the tag definition
  - node.childNodes[0].data -> the tag content data (string type), to be parse and build the python object from
  - node.tagName -> the name of the tag
  - node.parentNode -> the parent tag node

Writing XML:

- handlers are added with method addDOMTagWriter(pythonType, myWriteHandler), where pythonType is of python type 'type' and myWriteHandler a function.
- a tag writing handler must have the following prototype:

```
myWriteHandler(domDocument, node, pyObj):
```

  where:

  - domDocument is overall encapsulating structure (_xmlplus.dom.Document instance)
  - node (Node instance) is the current tag node to append data to
  - pyObj is the python object to convert into a 'human-readable' string.

- useful things to write handlers :

ATTRIBUTE_LABEL_META = 'meta'

ATTRIBUTE_LABEL_PYTHON_CLASS = 'pythonClass'

ATTRIBUTE_LABEL_PYTHON_CLASS_INIT_MODE = 'pythonInitMode'

ATTRIBUTE_LABEL_PYTHON_FUNCTION = 'pythonFunction'

ATTRIBUTE_LABEL_PYTHON_MODULE = 'pythonModule'

ATTRIBUTE_LABEL_TYPE = 'type'

TYPE_LABEL_ARRAY = 'array'

TYPE_LABEL_BOOL = 'bool'

TYPE_LABEL_DICT = 'struct'

TYPE_LABEL_FLOAT = 'double'

TYPE_LABEL_INT = 'int'

TYPE_LABEL_KEY_VAL_PAIR = 'dictItem'

TYPE_LABEL_LIST = 'list'

TYPE_LABEL_NONE = 'none'

TYPE_LABEL_ODICT = 'ordered_struct'

TYPE_LABEL_STRING = 'char'

TYPE_LABEL_TUPLE = 'tuple'

TYPE_LABEL_XML_INCLUDE = 'include'

static arrayDOMWriter(*doc*, *node*, *arrayObj*, *xmlHandler*)

static arrayTagDOMReader(*walker*, *xmlHandler*)

static boolDOMWriter(*doc*, *node*, *boolObj*, *xmlHandler*)

static boolTagDOMReader(*walker*, *xmlHandler*)

buildXMLString(*obj*, *label=None*, *pretty=False*)

createDocument()

static dictDOMWriter(*doc*, *node*, *dictObj*, *xmlHandler*, *atype=None*)

static dictTagDOMReader(*walker*, *xmlHandler*, *init_class=None*)

static floatDOMWriter(*doc*, *node*, *floatObj*, *xmlHandler*)

static floatTagDOMReader(*walker*, *xmlHandler*)

static includeTagDOMReader(*walker*, *xmlHandler*)

inspect_and_append_to_DOM_tree(*doc*, *node*, *obj*)

inspectable(*obj*)

static intDOMWriter(*doc*, *node*, *intObj*, *xmlHandler*)

static intTagDOMReader(*walker*, *xmlHandler*)

static listDOMWriter(*doc*, *node*, *listObj*, *xmlHandler*)

static listTagDOMReader(*walker*, *xmlHandler*)

mountDefaultHandlers()

static noneDOMWriter(*doc*, *node*, *noneObj*, *xmlHandler*)

static noneTagDOMReader(*walker*, *xmlHandler*)

static odictDOMWriter(*doc*, *node*, *dictObj*, *xmlHandler*)

static odictTagDOMReader(*walker*, *xmlHandler*)

packHandlers()

parseXMLString(*xml*)

readDOMData(*walker*)

rootTagDOMReader(*walker*)

static stringDOMWriter(*doc*, *node*, *stringObj*, *xmlHandler*)

static stringTagDOMReader(*walker*, *xmlHandler*)

static tupleDOMWriter(*doc*, *node*, *tupleObj*, *xmlHandler*)

static tupleTagDOMReader(*walker*, *xmlHandler*)

writeDOMData(*doc*, *node*, *obj*, *label*, *comment=None*, *meta=None*)

class pyhrf.xmliobak.xmlbase.**XMLParamDrivenClass**(*parameters=None*, *xmlHandler=<pyhrf.xmliobak.xmlbase.TypedXMLHandler instance>*, *xmlLabel=None*, *xmlComment=None*)

Base "abstract" class to handle parameters with clear specification and default values. Recursive aggregation is availaible to handle aggregated variables which also require parameter specifications.

appendParametersToDOMTree(*doc*, *node*)

defaultParameters = {}

fetchDefaultParameters()

parametersComments = {}

parametersMeta = {}

parametersToXml(*tagName=None*, *pretty=False*)

updateParameters(*newp*)

class pyhrf.xmliobak.xmlbase.**XMLParamDrivenClassInitException**

class pyhrf.xmliobak.xmlbase.**XMLable**(***kwargs*)

get_parameters_comments()

get_parameters_meta()

get_parameters_to_show()

override_init(*param_name*, *init_obj*, *init_params=None*)

override_value(*param_name*, *value*)

set_init(*init_func*, *init_params=None*)

class pyhrf.xmliobak.xmlbase.**XMLable2**
Bases: object

check_init_func(*params=None*)

get_parameters_comments()

get_parameters_meta()

get_parameters_to_show()

override_param_init(*init_func*, ***params*)
TODO (if needed)

set_init(*init_func*, ***init_params*)

set_init_param(*param_name*, *param_value*)

pyhrf.xmliobak.xmlbase.**from_xml**(*s*, *handler=<pyhrf.xmliobak.xmlbase.TypedXMLHandler instance>*)

pyhrf.xmliobak.xmlbase.**getargspec**(*func*)

pyhrf.xmliobak.xmlbase.**match_init_args**(*c*, *argsDict*)

pyhrf.xmliobak.xmlbase.**read_xml**(*fn*)

pyhrf.xmliobak.xmlbase.**to_xml**(*o*, *handler=<pyhrf.xmliobak.xmlbase.TypedXMLHandler instance>*, *objName='anonymObject'*, *pretty=False*)

pyhrf.xmliobak.xmlbase.**write_xml**(*obj*, *fn*)

pyhrf.xmliobak.xmlmatlab module

class pyhrf.xmliobak.xmlmatlab.**MatlabXMLHandler**
Bases: *pyhrf.xmliobak.xmlbase.TypedXMLHandler*

TYPE_LABEL_CELL = 'cell'

TYPE_LABEL_DOUBLE = 'double'

static cellDOMWriter(*doc*, *node*, *arrayObj*, *xmlHandler*)

static cellTagDOMReader(*walker*, *xmlHandler*)

static doubleDOMWriter(*doc*, *node*, *arrayObj*, *xmlHandler*)

static doubleTagDOMReader(*walker*, *xmlHandler*)

packHandlers()

pyhrf.xmliobak.xmlnumpy module

class pyhrf.xmliobak.xmlnumpy.**NumpyXMLHandler**(*write_callback=None*)

Bases: *pyhrf.xmliobak.xmlbase.TypedXMLHandler*

NUMPY_ARRAY_TAG_NAME = 'numpy.ndarray'

NUMPY_INT16_TAG_NAME = 'numpy.int16'

NUMPY_INT32_TAG_NAME = 'numpy.int32'

static arrayDOMWriter(*doc*, *node*, *arrayObj*, *xmlHandler*)

static arrayTagDOMReader(*walker*, *xmlHandler*)

static int16DOMWriter(*doc*, *node*, *intObj*, *xmlHandler*)

static int16TagDOMReader(*walker*, *xmlHandler*)

static numpyObjectTagDOMReader(*walker*, *xmlHandler*)

static numpyObjectTagDOMWriter(*doc*, *node*, *obj*, *xmlHandler*)

packHandlers()

CHAPTER 2

Submodules

2.1 pyhrf.configuration module

Loads and allows configuration of PyHRF.

exception pyhrf.configuration.**ConfigurationError**
Bases: exceptions.Exception

Exception class for configuration parsing errors.

pyhrf.configuration.**cfg_error_report**(*cfg*, *refcfg*)

pyhrf.configuration.**load_configuration**(*filename*, *refcfg*, *mode='file_only'*)
Load configuration file from 'filename' and check it against 'refcfg'. If mode is 'file_only' then only configuration in filename is returned. If mode is 'update' then the loaded configuration is updated with 'refcfg' to load defaults for unprovided parameters.

pyhrf.configuration.**write_configuration**(*cfg_dict*, *filename*, *section_order=None*)

2.2 pyhrf.core module

class pyhrf.core.**AttrClass**(***kwargs*)
Bases: object

Base class to display attributes.

class pyhrf.core.**Condition**(***kwargs*)
Bases: *pyhrf.core.AttrClass*

Represents an activation condition

class pyhrf.core.**FMRISessionSimulationData**(*onsets=OrderedDict([('audio', array([15., 20.7, 29.7, 35.4, 44.7, 48., 83.4, 89.7, 108., 119.4, 135., 137.7, 146.7, 173.7, 191.7, 236.7, 251.7, 284.4, 293.4, 296.7])), ('video', array([0., 2.4, 8.7, 33., 39., 41.7, 56.4, 59.7, 75., 96., 122.7, 125.4, 131.4, 140.4, 149.4, 153., 156., 159., 164.4, 167.7, 176.7, 188.4, 195., 198., 201., 203.7, 207., 210., 218.7, 221.4, 224.7, 234., 246., 248.4, 260.4, 264., 266.7, 269.7, 278.4, 288.]))]), durations=OrderedDict([('audio', array([], dtype=float64)), ('video', array([], dtype=float64))]), simulation_file='/home/docs/checkouts/readthedocs.org/user_builds/py packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/simu.pck'*)

Bases: *pyhrf.xmlio.Initable*

to_dict()

class pyhrf.core.**FMRISessionSurfacicData**(*onsets=OrderedDict([('audio', array([15., 20.7, 29.7, 35.4, 44.7, 48., 83.4, 89.7, 108., 119.4, 135., 137.7, 146.7, 173.7, 191.7, 236.7, 251.7, 284.4, 293.4, 296.7])), ('video', array([0., 2.4, 8.7, 33., 39., 41.7, 56.4, 59.7, 75., 96., 122.7, 125.4, 131.4, 140.4, 149.4, 153., 156., 159., 164.4, 167.7, 176.7, 188.4, 195., 198., 201., 203.7, 207., 210., 218.7, 221.4, 224.7, 234., 246., 248.4, 260.4, 264., 266.7, 269.7, 278.4, 288.]))]), durations=OrderedDict([('audio', array([], dtype=float64)), ('video', array([], dtype=float64))]), bold_file='/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/ packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/real_data_surf_tiny_bold.gii'*)

Bases: *pyhrf.xmlio.Initable*

to_dict()

class pyhrf.core.**FMRISessionVolumicData**(*onsets=OrderedDict([('audio', array([15., 20.7, 29.7, 35.4, 44.7, 48., 83.4, 89.7, 108., 119.4, 135., 137.7, 146.7, 173.7, 191.7, 236.7, 251.7, 284.4, 293.4, 296.7])), ('video', array([0., 2.4, 8.7, 33., 39., 41.7, 56.4, 59.7, 75., 96., 122.7, 125.4, 131.4, 140.4, 149.4, 153., 156., 159., 164.4, 167.7, 176.7, 188.4, 195., 198., 201., 203.7, 207., 210., 218.7, 221.4, 224.7, 234., 246., 248.4, 260.4, 264., 266.7, 269.7, 278.4, 288.]))]), durations=OrderedDict([('audio', array([], dtype=float64)), ('video', array([], dtype=float64))]), bold_file='/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/la packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/subj0_bold_session0.nii.gz'*)

Bases: *pyhrf.xmlio.Initable*

parametersComments = {'onsets': 'Onsets of experimental simtuli in seconds. \nDiction

to_dict()

class pyhrf.core.**FmriData**(*onsets*, *bold*, *tr*, *sessionsScans*, *roiMask*, *graphs=None*, *stimDurations=None*, *meta_obj=None*, *simulation=None*, *backgroundLabel=0*, *data_files=None*, *data_type=None*, *edge_lengths=None*, *mask_loaded_from_file=False*, *extra_data=None*)

Bases: *pyhrf.xmlio.Initable*

onsets
a dictionary mapping a stimulus name to a list of session onsets. Each item of this list is a 1D numpy float array of onsets for a given session.

stimDurations
same as 'onsets' but stores durations of stimuli

roiMask
numpy int array of roi labels (0 stands for the background). Shape depends on the data form (3D volumic or 1D surfacic)

bold
either a 4D numpy float array with axes [sag,cor,ax,scan] and then spatial axes must have the same shape as roiMask, or a 2D numpy float array with axes [scan, position] and position axis must have the same length as the number of positions within roiMask (without background). Sessions are stacked in the scan axis

sessionsScans
a list of session indexes along scan axis.

tr
Time of repetition of the BOLD signal

simulation
if not None then it should be a list of simulation instance.

meta_obj
extra information associated to data

average(*flag=True*)

build_graphs(*force=False*)

compute_average()

discard_rois(*roi_ids*)

discard_small_rois(*min_size*)

classmethod from_simu_ui(*sessions_data=None*)

classmethod from_simulation_dict(*simulation*, *mask=None*)

classmethod from_surf_files(*paradigm_csv_file='/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/late packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/paradigm_loc_av.csv'*, *bold_files=None*, *tr=2.4*, *mesh_file='/home/docs/checkouts/readthedocs.org/user_build packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/real_data_surf_tiny_mesh.gii'*, *mask_file=None*)

Return FmriData representation from surf files

classmethod from_surf_ui(*sessions_data=None*, *tr=2.4*, *mask_file='/home/docs/checkouts/readthedocs.org/user_build packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/real_data_surf_tiny_parcellati mesh_file='/home/docs/checkouts/readthedocs.org/user_builds/pyhrf/envs/latest/lib/python2 packages/pyhrf-0.4.3-py2.7-linux-x86_64.egg/pyhrf/datafiles/real_data_surf_tiny_mesh.gii'*

Convenient creation function intended to be used for XML I/O. 'session_data' is a list of FMRISession-VolumicData objects. 'tr' is the time of repetition. 'mask_file' is a path to a functional mask file.

This represents the following hierarchy:

```
- FMRIData:
   - list of session data:
       [ * data for session 1:
                - onsets for session 1,
                - durations for session 1,
                - fmri data file for session 1 (gii)
         * data for session 2:
                - onsets for session 2,
                - durations for session 2,
                - fmri data file for session 2 (gii)
       ],
   - time of repetition
   - mask file
   - mesh file
```

from_vol_files

from_vol_files_rel

from_vol_ui

getSummary(*long=False*)

get_condition_names()

get_data_files()

get_extra_data(*label*, *default*)

get_graph()

get_joined_durations()

get_joined_onsets()

get_nb_rois()
Return the number of parcels (background id is discarded)

get_nb_vox_in_mask()

get_roi_id()
In case of FMRI data containing only one ROI, return the id of this ROI. If data contains several ROIs then raise an exception

get_roi_mask()

keep_only_rois(*roiIds*)

parametersComments = {'mask_file': 'Input n-ary mask file (= parcellation). Only posi

parametersToShow = ['tr', 'sessions_data', 'mask_file']

roiMask

roi_split(*mask=None*)

save(*output_dir*)
Save paradigm to output_dir/paradigm.csv, BOLD to output_dir/bold.nii, mask to output_dir/mask.nii #TODO: handle multi-session

Return: tuple of file names in this order: (paradigm, bold, mask)

set_extra_data(*label*, *value*)

store_mask_sparse(*roiMask*)

class pyhrf.core.**FmriGroupData**(*list_subjects*)
Bases: *pyhrf.xmlio.Initable*

Used for group level hemodynamic analysis Encapsulates FmriData objects for all subjects All subjects must habe the same number of ROIs

Inputs: list_subjects: contains list of FmriData object for each subject

build_graphs(*force=False*)

getSummary(*long=False*)

get_roi_id()

roi_split()
Retrieve a list of FmriGroupData object, each containing the data for all subject, in one ROI

class pyhrf.core.**Object**
Bases: object

pyhrf.core.**get_data_file_name**(*filename*)
Return the path of a given filename.

pyhrf.core.**get_roi_simulation**(*simu_sessions*, *mask*, *roi_id*)
Extract the ROI from the given simulation dict. :param - simu: dictionnary of simulated quantities :type - simu: dict :param - mask: binary mask defining the spatial extent of the ROI :type - mask: np.ndarray :param - roi_id: the id of the roi to extract :type - roi_id: int

Returns dict of roi-specific simulation items

pyhrf.core.**get_src_doc_path**()
Return the documentation path of pyhrf.

pyhrf.core.**get_src_path**()
Return the source path of pyhrf.

pyhrf.core.**get_tmp_path**(*tag='pyhrf_'*)
Return a temporary path.

pyhrf.core.**list_data_file_names**()
List all the data filenames.

pyhrf.core.**load_surf_bold_mask**(*bold_files*, *mesh_file*, *mask_file=None*)

pyhrf.core.**load_vol_bold_and_mask**(*bold_files*, *mask_file*)

pyhrf.core.**merge_fmri_sessions**(*fmri_data_sets*)
fmri_data_sets: list of FmriData objects. Each FmriData object is assumed to contain only one session

pyhrf.core.**merge_fmri_subjects**(*fmri_data_sets*, *roiMask*, *backgroundLabel=0*)
fmri_data_sets: list of FmriData objects, for different subjects. In case of multisession data, merging of fmri data over sessions must be done for each subject before using this function. roiMask: multi_subject parcellation (nparray)

2.3 pyhrf.glm module

pyhrf.glm.**glm_nipy**(*fmri_data, contrasts=None, hrf_model='Canonical', drift_model='Cosine', hfcut=128, residuals_model='spherical', fit_method='ols', fir_delays=[0], rescale_results=False, rescale_factor=None*)

Perform a GLM analysis on fMRI data using the implementation of Nipy.

Parameters

- **fmri_data** (pyhrf.core.FmriData) – the input fMRI data defining the paradigm and the measured 3D+time signal.
- **contrasts** (*dict*) – keys are contrast labels and values are arithmetic expressions involving regressor names. Valid names are: * names of experimental conditions as defined in fmri_data * constant
- **hrf_model** – "Canonical", "Canonical with Derivative", "FIR"
- **residuals_model** – "spherical", "ar1"
- **fit_method** – "ols", "kalman" (If residuals_model is "ar1" then method is set to "kalman" and this argument is ignored)
- **fir_delays** – list of integers indicating the delay of each FIR coefficient (in terms of scans). Eg if TR = 2s. and we want a FIR duration of 20s.: fir_delays=range(10)

Returns (glm instance, design matrix, dict of contrasts of objects)

Examples: >>> from pyhrf.core import FmriData >>> from pyhrf.glm import glm_nipy >>> g,dmtx,con = glm_nipy(FmriData.from_vol_ui()) >>> g,dmtx,con = glm_nipy(FmriData.from_vol_ui(), contrasts={'A-V':'audio-video'})

pyhrf.glm.**glm_nipy_from_files**(*bold_file, tr, paradigm_csv_file, output_dir, mask_file, session=0, contrasts=None, con_test_baseline=0.0, hrf_model='Canonical', drift_model='Cosine', hfcut=128, residuals_model='spherical', fit_method='ols', fir_delays=[0]*)

#TODO: handle surface data hrf_model : Canonical | Canonical with Derivative | FIR

2.4 pyhrf.graph module

Module to handle graphs. Base structures : - undirected, unweighted graph: a list of neighbours index (list of numpy array).

pyhrf.graph.**bfs_set_label**(*g*, *root*, *data*, *value*, *radius*)

pyhrf.graph.**bfs_sub_graph**(*g*, *root*, *radius*)

pyhrf.graph.**breadth_first_search**(*graph*, *root=0*, *visitable=None*)

Traverses a graph in breadth-first order.

The first argument should be the tree root; visitable should be an iterable with all searchable nodes;

pyhrf.graph.**center_mask_at**(*mask*, *pos*, *indexes*, *toroidal=False*)

pyhrf.graph.**center_mask_at_v01**(*mask*, *pos*, *shape*)

pyhrf.graph.**center_mask_at_v02**(*mask*, *pos*, *shape*)

pyhrf.graph.**connected_components**(*g*)

pyhrf.graph.**connected_components_iter**(*g*)

pyhrf.graph.**flatten_and_graph**(*data*, *mask=None*, *kerMask=None*, *depth=1*, *toroidal=False*)

pyhrf.graph.**graph_from_lattice**(*mask*, *kerMask=None*, *depth=1*, *toroidal=False*)
Creates a graph from a n-dimensional lattice 'mask' define valid positions to build the graph over. 'kerMask' is numpy array mask (tuple of arrays) which defines the neighbourhood system, ie the relative positions of neighbours for a given position in the lattice.

pyhrf.graph.**graph_from_lattice3D**(*mask*, *kerMask=None*, *depth=1*, *toroidal=False*)
Creates a graph from a n-dimensional lattice 'mask' define valid positions to build the graph over. 'kerMask' is numpy array mask (tuple of arrays) which defines the neighbourhood system, ie the relative positions of neighbours for a given position in the lattice.

pyhrf.graph.**graph_from_mesh**(*polygonList*)
Return the list of neighbours indexes for each position, from a list of polygons. Each polygon is a triplet.

pyhrf.graph.**graph_is_sane**(*g*, *toroidal=False*)
Check the structure of graph 'g', which is a list of neighbours index. Return True if check is ok, else False. -> every nid in g[id] should verify id in g[nid] -> any neighbours list must have unique elements -> no isolated node

pyhrf.graph.**graph_nb_cliques**(*graph*)

pyhrf.graph.**graph_pool_indexes**(*g*)

pyhrf.graph.**graph_pygraph**(*g*)

pyhrf.graph.**graph_to_sparse_matrix**(*graph*)
Creates a connectivity sparse matrix from the adjacency graph (list of neighbors list)

pyhrf.graph.**parcels_to_graphs**(*parcellation*, *kerMask*, *toKeep=None*, *toDiscard=None*)
Compute graphs for each parcel in parcels. A graph is simply defined as a list of neighbour indexes.

Parameters

- **parcellation** – is a n-ary numpy array.
- **kerMask** – defines the connectivity
- **toKeep** –
- **toDiscard** –

Returns

Return type a dictionary mapping a roi ID to its graph

pyhrf.graph.**split_mask_into_cc_iter**(*mask*, *min_size=0*, *kerMask=None*)
Return an iterator over all connected components (CC) within input mask. CC which are smaller than *min_size* are discarded. *kerMask* defines the connectivity, e.g., kerMask3D_6n for 6-neighbours in 3D.

Examples

```
vol = np.array( [[1,1,0,1,1],
                 [1,1,0,1,1],
                 [0,0,0,0,0],
                 [1,0,1,1,0],
                 [0,0,1,1,0]], dtype=int )
for cc in split_mask_into_cc_iter(vol):
    print cc
```

Should output:

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

pyhrf.graph.**sub_graph**(*graph*, *nodes*)

2.5 pyhrf.grid module

Module to distribute shell commands across a network

Original Author: Mathieu Perrot Extended by: Thomas Vincent

class pyhrf.grid.**DispatchedTasksManager**(**args*, ***kwargs*)
Bases: *pyhrf.grid.TasksManager*

start()

wait_for_end_or_cmd(*print_number*, *tasks_number*, *cmd*)

class pyhrf.grid.**HierarchicalTasksManager**(**args*, ***kwargs*)
Bases: *pyhrf.grid.DispatchedTasksManager*

start()

class pyhrf.grid.**Host**(*name*, *status*)
Bases: object

name : hostname. status : set host status :

class pyhrf.grid.**HostsManager**(*list*)
Bases: object

available_status = 0

isup(*hostname*)

not_available_status = 1

probe(*hostname*)

unknown_host_status = 2

unknown_status = 3

update_all_hosts()

update_host_status(*host*, *status*)

class pyhrf.grid.**OneTaskManager**(**args*, ***kwargs*)
Bases: *pyhrf.grid.TasksManager*

abnormal_stop(*task*)

start()

class pyhrf.grid.**ProbeHost**(*hosts_manager*, *hostname*)
Bases: threading.Thread

run()

class pyhrf.grid.**RepeatedTasksManager**(**args*, ***kwargs*)
Bases: *pyhrf.grid.TasksManager*

abnormal_stop(*task*)

start()

class pyhrf.grid.**Task**(*task*)
Bases: object

Only one task that will be computed on only one host.

get()

class pyhrf.grid.**TaskHierarchical**(*rule*, *tasks_dic*)
Bases: *pyhrf.grid.Task*

Hiearchic dependencies of TaskList.

init()

next()

class pyhrf.grid.**TaskList**(*tasks*)
Bases: *pyhrf.grid.Task*

List of independent tasks. Each one can be computed on a different task.

append(*task*)

next()

class pyhrf.grid.**TasksManager**(*timeslot*, *user*, *tasks*, *hosts_manager*, *log*, *brokenfd*, *time_limit=86400*)
Bases: object

abnormal_stop(*task*)

print_status(*n*, *size*)

wait_to_be_ready()

class pyhrf.grid.**TasksStarter**(*tasks_manager*, *host*, *task*, *time_limit=86400*)
Bases: threading.Thread

kill()

run()

class pyhrf.grid.**TimeSlot**(*start*, *end*)
Bases: object

Define a contiguous timeslot.

start, end : in second since day beggining.

is_inside(*time*)

is_inside_now()

class pyhrf.grid.**TimeSlotList**(*list*)
Bases: *pyhrf.grid.TimeSlot*

Define uncontiguous timslots.

list : list of timeslots.

is_inside(*time*)

class pyhrf.grid.**User**(*name=None*, *passwd=None*, *keytype=None*)
Bases: object

Define user launching task and identification process.

Parameters

- **name** – username.
- **passwd** – user passwd or if None, try to get *~/.ssh/id_dsa* dsa key for key connection.
- **keytype** – *rsa* or *dsa*.

key()

pyhrf.grid.**broken_help**(*cmd*)

pyhrf.grid.**create_options**(*argv*)

pyhrf.grid.**hosts_help**(*cmd*)

pyhrf.grid.**kill_threads**()

pyhrf.grid.**log_help**(*cmd*)

pyhrf.grid.**main**()

pyhrf.grid.**main_safe**()

pyhrf.grid.**mode_help**(*cmd*)

pyhrf.grid.**parse_options**(*parser*)

pyhrf.grid.**quit**(*signal*, *frame*)

pyhrf.grid.**read_hierarchic_tasks**(*tasks_file*)

pyhrf.grid.**read_hosts**(*hosts*)

pyhrf.grid.**read_tasks**(*tasks*, *mode*)

pyhrf.grid.**read_timeslot**(*timeslot*)

pyhrf.grid.**remote_dir_is_writable**(*user*, *hosts*, *path*)
Test if *path* is writable from each host in *hosts*. Sending bash commands to each host via ssh using the given *user* login.

Args:

pyhrf.grid.**run_grid**(*mode*, *hosts_list*, *keytype*, *tasks*, *timeslot*, *brokenfile=None*, *logfile=None*, *user=None*, *passwd=None*, *time_limit=86400*)

pyhrf.grid.**tasks_help**(*cmd*)

pyhrf.grid.**timeslot_help**(*cmd*)

2.6 pyhrf.ndarray module

This module provides classes and functions to handle multi-dimensionnal numpy array (ndarray) objects and extend them with some semantics (axes labels and axes domains). See xndarray class. (TODO: make xndarray inherit numpy.ndarray?)

exception pyhrf.ndarray.**ArrayMappingError**
Bases: exceptions.Exception

pyhrf.ndarray.**expand_array_in_mask**(*flat_data*, *mask*, *flat_axis=0*, *dest=None*, *m=None*)
Map the *flat_axis* of *flat_data* onto the region within mask. *flat_data* is then reshaped so that flat_axis is replaced with *mask.shape*.

Notes

m is the result of *np.where(mask)* -> can be passed to speed up if already done before.

Examples

```
>>> a = np.array([1,2,3])
>>> m = np.array([[0,1,0], [0,1,1]] )
>>> expand_array_in_mask(a,m)
array([[0, 1, 0],
       [0, 2, 3]])
```

```
>>> a = np.array([[1,2,3],[4,5,6]])
>>> m = np.array([[0,1,0], [0,1,1]] )
>>> expand_array_in_mask(a,m,flat_axis=1)
array([[[0, 1, 0],
        [0, 2, 3]],

       [[0, 4, 0],
        [0, 5, 6]]])
```

pyhrf.ndarray.**merge**(*arrays*, *mask*, *axis*, *fill_value=0*)
Merge the given arrays into a single array according to the given mask, with the given axis being mapped to those of mask. Assume that arrays[id] corresponds to mask==id and that all arrays are in the same orientation.

Arg:

- arrays (dict of xndarrays):
- **mask (xndarray): defines the mapping between the flat axis in the** arrays to merge and the target expanded axes.
- axis (str): flat axis for the

pyhrf.ndarray.**split_and_save**(*cub*, *axes*, *fn*, *meta_data=None*, *set_MRI_orientation=False*, *output_dir=None*, *format_dvalues=False*)

pyhrf.ndarray.**stack_cuboids**(*c_list*, *axis*, *domain=None*, *axis_pos='first'*)
Stack xndarray instances in list *c_list* along a new axis label *axis*. If *domain* (numpy array or list) is provided, it is associated to the new axis. All cuboids in *c_list* must have the same orientation and domains. *axis_pos* defines the position of the new axis: either *first* or *last*.

Examples

```
>>> import numpy as np
>>> from pyhrf.ndarray import xndarray, stack_cuboids
>>> c1 = xndarray(np.arange(4*3).reshape(4,3), ['x','y'])
>>> c1
```

```
axes: ['x', 'y'], array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
>>> c2 = xndarray(np.arange(4*3).reshape(4,3)*2, ['x','y'])
>>> c2
axes: ['x', 'y'], array([[ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16],
       [18, 20, 22]])
>>> c_stacked = stack_cuboids([c1,c2], 'stack_axis', ['c1','c2'])
>>> print c_stacked.descrip()
* shape : (2, 4, 3)
* dtype : int64
* orientation: ['stack_axis', 'x', 'y']
* value label: value
* axes domains:
  'stack_axis': array(['c1', 'c2'],
      dtype='|S2')
  'x': arange(0,3,1)
  'y': arange(0,2,1)
```

TODO: enable broadcasting (?)

pyhrf.ndarray.**tree_to_xndarray**(*tree*, *level_labels=None*)

Stack all arrays within input tree into a single array.

Parameters

- **tree** (*dict*) – nested dictionaries of xndarray objects. Each level of the tree correspond to a target axis, each key of the tree correspond to an element of the domain associated to that axis.
- **level_labels** (*list of str*) – axis labels corresponding to each level of the tree

Returns

Return type xndarray object

Examples

```
>>> from pyhrf.ndarray import xndarray, tree_to_xndarray
>>> d = { 1 : { .1 : xndarray([1,2], axes_names=['inner_axis']),
↪    .2 : xndarray([3,4], axes_names=['inner_axis']),                    },
↪                                                    2 : { .1 : xndarray([1,
↪2], axes_names=['inner_axis']),                    .2 : xndarray([3,4], axes_
↪names=['inner_axis']),                  }
↪                         }
>>> tree_to_xndarray(d, ['level_1', 'level_2'])
axes: ['level_1', 'level_2', 'inner_axis'], array([[[1, 2],
        [3, 4]],

       [[1, 2],
        [3, 4]]])
```

class pyhrf.ndarray.**xndarray**(*narray*, *axes_names=None*, *axes_domains=None*, *value_label='value'*, *meta_data=None*)

Handles a multidimensional numpy array with axes that are labeled and mapped to domain values.

Examples

```
>>> c = xndarray( [ [4,5,6],[8,10,12] ], ['time','position'], {'time':[0.1,0.2]} )
```

Will represent the following situation:

```
position
------->
4  5  6 | t=0.1   |time
8 10 12 | t=0.2   v
```

add(*c*, *dest=None*)

astype(*t*)

cexpand(*cmask*, *axis*, *dest=None*)
Same as expand but mask is a cuboid

TODO: + unit test

cflatten(*cmask*, *new_axis*)

copy(*copy_meta_data=False*)
Return copy of the current cuboid. Domains are copied with a shallow dictionnary copy.

descrip()
Return a printable string describing the cuboid.

descrip_shape()

divide(*c*, *dest=None*)

expand(*mask*, *axis*, *target_axes=None*, *target_domains=None*, *dest=None*, *do_checks=True*, *m=None*)
Create a new xndarray instance (or store into an existing *dest* cuboid) where *axis* is expanded and values are mapped according to *mask*.

- *target_axes* is a list of the names of the new axes replacing *axis*.
- *target_domains* is a dict of domains for the new axes.

Examples

```
>>> import numpy as np
>>> from pyhrf.ndarray import xndarray
>>> c_flat = xndarray(np.arange(2*6).reshape(2,6).astype(np.int64),
↪                   ['condition', 'voxel'],                                {
↪'condition' : ['audio','video']})
>>> print c_flat.descrip()
* shape : (2, 6)
* dtype : int64
* orientation: ['condition', 'voxel']
* value label: value
* axes domains:
  'condition': array(['audio', 'video'],
      dtype='|S5')
  'voxel': arange(0,5,1)
>>> mask = np.zeros((4,4,4), dtype=int)
>>> mask[:3,:2,0] = 1
>>> c_expanded = c_flat.expand(mask, 'voxel', ['x','y','z'])
```

```
>>> print c_expanded.descrip()
* shape : (2, 4, 4, 4)
* dtype : int64
* orientation: ['condition', 'x', 'y', 'z']
* value label: value
* axes domains:
  'condition': array(['audio', 'video'],
      dtype='|S5')
  'x': arange(0,3,1)
  'y': arange(0,3,1)
  'z': arange(0,3,1)
```

explode(*cmask*, *new_axis='position'*)
Explode array according to the given n-ary *mask* so that axes matchin those of *mask* are flatten into *new_axis*.

Parameters

- **mask** (–) – n-ary mask that defines "regions" used to split data
- **new_axis** (–) – target flat axis

Returns dict of xndarray that maps a mask value to a xndarray.

explode_a(*mask*, *axes*, *new_axis*)
Explode array according to given n-ary *mask* so that *axes* are flatten into *new_axis*.

Parameters

- **mask** (–) – n-ary mask that defines "regions" used to split data
- **axes** (–) – list of axes in the current object that are mapped onto the mask
- **new_axis** (–) – target flat axis

Returns dict of xndarray that maps a mask value to a xndarray.

fill(*c*)

flatten(*mask*, *axes*, *new_axis*)
flatten cudoid.

TODO: +unit test

get_axes_domains()
Return domains associated to axes as a dict (axis_name:domain array)

get_axes_ids(*axes_names*)
Return the index of all axes in given axes_names

get_axis_id(*axis_name*)
Return the id of an axis from the given name.

get_axis_name(*axis_id*)
Return the name of an axis from the given index 'axis_id'.

get_domain(*axis_id*)
Return the domain of the axis *axis_id*

Examples

```
>>> from pyhrf.ndarray import xndarray
>>> c = xndarray(np.random.randn(10,2), axes_names=['x','y'],
↪       axes_domains={'y' : ['plop','plip']})
>>> (c.get_domain('y') == np.array(['plop', 'plip'], dtype='|S4')).all()
True
>>> c.get_domain('x') #default domain made of slice indexes
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

get_domain_idx(*axis*, *value*)
Get slice index from domain value for axis 'axis'.

get_dshape()
Return the shape of the array as dict mapping an axis name to the corresponding size

get_extra_info(*fmt='dict'*)

get_ndims()

get_new_axis_name()
Return an axis label not already in use. Format is: dim%d

get_orientation()

get_voxel_size(*axis*)
Return the size of a voxel along 'axis', only if meta data is available.

has_axes(*axes*)

has_axis(*axis*)

len(*axis*)

static load(*file_name*)
Load cuboid from file. Supported format: nifti1. Extra axis information is retrieved from a nifti extension if available. If it's not available, label the axes as: (sagittal, coronal, axial[, time]).

TODO: gifti.

map_onto(*xmapping*)
Reshape the array by mapping the axis corresponding to xmapping.value_label onto the shape of xmapping.

Parameters **xmapping** (xndarray) – array whose attribute value_label matches an axis of the current array

Returns

Return type a new array (*xndarray*) where values from the current array have been mapped according to xmapping

Examples

```
>>> from pyhrf.ndarray import xndarray
>>> import numpy as np
>>> # data with a region axis:
>>> data = xndarray(np.arange(2*4).reshape(2,4).T * .1,
↪         ['time', 'region'],                                    {
↪'time':np.arange(4)*.5, 'region':[2, 6]})
```

```
>>> data
axes: ['time', 'region'], array([[ 0. ,  0.4],
       [ 0.1,  0.5],
       [ 0.2,  0.6],
       [ 0.3,  0.7]])
>>> # 2D spatial mask of regions:
>>> region_map = xndarray(np.array([[2,2,2,6], [6,6,6,0], [6,6,0,0]]),
↪                          ['x','y'], value_label='region')
>>> # expand region-specific data into region mask
>>> # (duplicate values)
>>> data.map_onto(region_map)
axes: ['x', 'y', 'time'], array([[[ 0. ,  0.1,  0.2,  0.3],
        [ 0. ,  0.1,  0.2,  0.3],
        [ 0.4,  0.5,  0.6,  0.7]],

       [[ 0.4,  0.5,  0.6,  0.7],
        [ 0.4,  0.5,  0.6,  0.7],
        [ 0.4,  0.5,  0.6,  0.7],
        [ 0. ,  0. ,  0. ,  0. ]],

       [[ 0.4,  0.5,  0.6,  0.7],
        [ 0.4,  0.5,  0.6,  0.7],
        [ 0. ,  0. ,  0. ,  0. ],
        [ 0. ,  0. ,  0. ,  0. ]]])
```

max(*axis=None*)

mean(*axis=None*)

min(*axis=None*)

multiply(*c*, *dest=None*)

ptp(*axis=None*)

reorient(*orientation*)
Return a cuboid with new orientation. If cuboid is already in the right orientation, then return the current cuboid. Else, create a new one.

repeat(*n*, *new_axis*, *domain=None*)
Return a new cuboid with self's data repeated 'n' times along a new axis labelled 'new_axis'. Associated 'domain' can be provided.

rescale_values(*v_min=0.0*, *v_max=1.0*, *axis=None*)

roll(*axis*, *pos=-1*)
Roll xndarray by making 'axis' the last axis. 'pos' is either 0 or -1 (first or last, respectively) TODO: handle all pos.

save(*file_name*, *meta_data=None*, *set_MRI_orientation=False*)
Save cuboid to a file. Supported format: Nifti1. 'meta_data' shoud be a 2-elements tuple: (affine matrix, Nifti1Header instance). If provided, the meta_data attribute of the cuboid is ignored. All extra axis information is stored as an extension.

set_MRI_orientation()
Set orientation to sagittal,coronal,axial,[time|iteration|condition] Priority for the 4th axis: time > condition > iteration. The remaining axes are sorted in alphatical order

set_axis_domain(*axis_id*, *domain*)
Set the value domain mapped to *axis_id* as *domain*

Parameters

- **axis_id** (–) – label of the axis
- **domain** (–) – value domain

Returns None

set_orientation(*axes*)
Set the cuboid orientation (inplace) according to input axes labels

split(*axis*)
Split a cuboid along given axis. Return an OrderedDict of cuboids.

squeeze(*axis=None*)
Remove all dims which have length=1. 'axis' selects a subset of the single-dimensional axes.

squeeze_all_but(*axes*)

std(*axis=None*)

sub_cuboid(*orientation=None*, ***kwargs*)
Return a sub cuboid. 'kwargs' allows argument in the form: axis=slice_value.

sub_cuboid_from_slices(*orientation=None*, ***kwargs*)
Return a sub cuboid. 'kwargs' allows argument in the form: axis=slice_index.

substract(*c*, *dest=None*)

sum(*axis=None*)

swapaxes(*a1*, *a2*)
Swap axes *a1* and *a2*

Parameters

- **a1** (–) – identifier of the 1st axis
- **a2** (–) – identifier of the 2nd axis

Returns A new cuboid wrapping a swapped view of the numpy array

to_html_table(*row_axes*, *col_axes*, *inner_axes*, *cell_format='txt'*, *plot_dir=None*, *rel_plot_dir=None*, *plot_fig_prefix='xarray_'*, *plot_style='image'*, *plot_args=None*, *tooltip=False*, *border=None*)
Render the array as an html table whose column headers correspond to domain values and axis names defined by *col_axes*, row headers defined by *row_axes* and inner cell axes defined by *inner_axes* Data within a cell can be render as text or as a plot figure (image files are produced)

Parameters **–** –

Returns html code (str)

to_latex(*row_axes=None*, *col_axes=None*, *inner_axes=None*, *inner_separator=' | '*, *header_styles=None*, *hval_fmt=None*, *val_fmt='%1.2f'*, *col_align=None*)

to_tree(*level_axes*, *leaf_axes*)
Convert nested dictionary mapping where each key is a domain value and each leaf is an array or a scalar value if *leaf_axes* is empty.

Returns {dv_axis1 : {dv_axis2 : {... : xndarray|scalar_type}

Return type OrderedDict such as

Example: >>> from pyhrf.ndarray import xndarray >>> import numpy as np >>> c = xndarray(np.arange(4).reshape(2,2), axes_names=['a1','ia'], axes_domains={'a1':['out_dv1', 'out_dv2'],

'ia':['in_dv1', 'in_dv2']}) >>> c.to_tree(['a1'], ['ia']) OrderedDict([('out_dv1', axes: ['ia'], array([0, 1])), ('out_dv2', axes: ['ia'], array([2, 3]))])

unstack(*outer_axes*, *inner_axes*)
Unstack the array along outer_axes and produce a xndarray of xndarrays

Parameters

- **outer_axes** (–) – list of axis names defining the target unstacked xndarray
- **inner_axes** (–) – list of axis names of any given sub-array of the target unstacked xndarray

Returns xndarray object

Example: >>> from pyhrf.ndarray import xndarray >>> import numpy as np >>> c = xndarray(np.arange(4).reshape(2,2), axes_names=['a1','ia'], axes_domains={'a1':['out_dv1', 'out_dv2'], 'ia':['in_dv1', 'in_dv2']}) >>> c.unstack(['a1'], ['ia']) axes: ['a1'], array([axes: ['ia'], array([0, 1]), axes: ['ia'], array([2, 3])], dtype=object)

var(*axis=None*)

static xndarray_like(*c*, *data=None*)
Return a new cuboid from data with axes, domains and value label copied from 'c'. If 'data' is provided then set it as new cuboid's data, else a zero array like c.data is used.

TODO: test

pyhrf.ndarray.**xndarray_like**(*c*, *data=None*)

2.7 pyhrf.paradigm module

class pyhrf.paradigm.**Paradigm**(*stimOnsets*, *sessionDurations=None*, *stimDurations=None*)

delete_condition(*cond*)

classmethod from_csv(*csvFile*, *delim=None*)
Create a Paradigm object from a CSV file which columns are: session, task name, stimulation onset, stimulation duration, [amplitude]

classmethod from_session_dict(*d*, *sessionDurations=None*)

classmethod from_spm_mat(*spm_mat_file*)
TODO: handle session durations

get_info(*long=True*)

get_joined_and_rastered(*dt*)

get_joined_durations()
For each condition, join stimulus durations of all sessions.

get_joined_durations_dim()
For each condition, join stimulus durations of all sessions.

get_joined_onsets()
For each condition, join onsets of all sessions.

get_joined_onsets_dim()
For each condition, join onsets of all sessions.

get_nb_trials()

get_rastered(*dt*, *tMax=None*)
Return binary sequences of stimulus arrivals. Each stimulus event is approximated to the closest time point on the time grid defined by dt. eg return

```
{ 'cond1' : [np.array([ 0 0 0 1 0 0 1 1 1 0 1]),
             np.array([ 0 1 1 1 0 0 1 0 1 0 0])] },
  'cond2' : [np.array([ 0 0 0 1 0 0 1 1 1 0 0]),
             np.array([ 1 1 0 1 0 1 0 0 0 0 0])] },
```

Parameters

- **dt** (*float*) – temporal resolution of the target grid
- **tMax** (*float*) – total duration of the paradigm. If None, then use the session lengths

get_stimulus_names()

get_t_max()

join_sessions()

save_csv(*csvFile*)

save_spm_mat_for_1st_level_glm(*mat_file*, *session=0*)

to_nipy_paradigm()

pyhrf.paradigm.**check_stim_durations**(*stim_onsets*, *stimDurations*)
If no durations specified (stimDurations is None or empty np.array) then assume spiked stimuli: return a sequence of zeros with same shape as onsets sequence. Check that durations have same shape as onsets.

pyhrf.paradigm.**contrasts_to_spm_vec**(*condition_list*, *contrasts*)

pyhrf.paradigm.**extend_sampled_events**(*sampled_events*, *sampled_durations*)
Add events to encode stimulus duration

pyhrf.paradigm.**merge_onsets**(*onsets*, *new_condition*, *criterion=None*, *durations=None*, *discard=None*)
Convention for definition of onsets or durations.

```
OrderedDict({
    'condition_name': [ <array of timings for sess1>,
                        <array of timings for sess2>,
                        ...]
    }
```

pyhrf.paradigm.**restarize_events**(*events*, *durations*, *dt*, *t_max*)
build a binary sequence of events. Each event start is approximated to the nearest time point on the time grid defined by dt and t_max.

2.8 pyhrf.parallel module

exception pyhrf.parallel.**RemoteException**
Bases: exceptions.Exception

pyhrf.parallel.**dump_func**(*func*, *fn*)

pyhrf.parallel.**merge_default_kwargs**(*func*, *kwargs*)

pyhrf.parallel.**prepare_treatment_jobs**(*treatment*, *tmp_local_dir*, *local_result_path*, *local_user*, *local_host*, *remote_host*, *remote_user*, *remote_path*, *label_for_cluster*)

Prepare soma-workflow jobs to perform one treatment (i.e., one subject).

Parameters

- **treatment** (FMRITreatment) – the treatment defining the analysis
- **tmp_local_dir** (*str*) – a path where to store the temporary config file before sending it to the remote host
- **local_result_path** (*str*) – path where to store the final result
- **local_user** (*str*) – the user on the local host who enables SHH connection from the remote cluster
- **local_host** (*str*) – local host (used to send back the result)
- **remote_host** (*str*) – remote machine where the treatment will be run
- **remote_user** (*str*) – user login on the remote machine.
- **remote_path** (*str*) – path on the remote machine where to store ROI data and analysis results
- **label_for_cluster** (*str*) – label prefix to name job in soma-workflow

Returns

- *a tuple (job_split, jobs, dependencies, mainGroup)*
- *job_split (Job)* – job handling splitting of input data into ROI data
- *jobs (list of Job)* – all jobs except the splitting jobs -> roi analyses, result merge, scp of result back to local host, data cleaning
- *dependencies (list of job pairs)* – define the pipeline structure
- *mainGroup (Group)* – top-level object gathering all jobs for this treatment.

pyhrf.parallel.**remote_map**(*func*, *largs=None*, *lkwargs=None*, *mode='serial'*)

Execute a function in parallel on a list of arguments.

Parameters

- ***func*** (*function*) – function to apply on each item. **this function must be importable on the remote side**
- ***largs*** (*list of tuple*) – each item in the list is a tuple containing all positional argument values of the function
- ***lkwargs*** (*list of dict*) – each item in the list is a dict containing all named arguments of the function mapped to their value.
- ***mode*** (*str*) – indicates how execution is distributed. Choices are:
  - ”serial”: single-thread loop on the local machine
  - **”local”** [use joblib to run tasks in parallel.] The number of simultaneous jobs is defined in the configuration section [‘parallel-local’][‘nb_procs’] see ~/.pyhrf/config.cfg
  - **”remote_cluster: use somaworkflow to run tasks in parallel.** The connection setup has to be defined in the configuration section [‘parallel-cluster’] of ~/.pyhrf/config.cfg.
  - **”local_with_dumps”: testing purpose only, run each task serially as** a subprocess.

Returns a list of results

Raises RemoteException if any remote task has failed

Example: >>> from pyhrf.parallel import remote_map >>> def foo(a, b=2): return a + b >>> remote_map(foo, [(2,),(3,)], [{'b':5}, {'b':7}]) [7, 10]

pyhrf.parallel.**remote_map_marshal**(*func*, *largs=None*, *lkwargs=None*, *mode='local'*)

pyhrf.parallel.**run_soma_workflow**(*treatments*, *exec_cmd*, *tmp_local_dirs*, *server_id*, *remote_host*, *remote_user*, *remote_pathes*, *local_result_pathes*, *label_for_cluster*, *wait_ending=False*)

Dispatch treatments using soma-workflow.

Parameters

- **treatments** – it is a dict mapping a treatment name to a treatment object
- **exec_cmd** – it is the command to run on each ROI data.
- **tmp_local_dirs** – it is a dict mapping a treatment name to a local tmp dir (used to store a temporary configuration file)
- **server_id** – it is the server ID as expected by WorkflowController
- **remote_host** – it is the remote machine where treatments are treated in parallel
- **remote_user** – it is used to log in remote_host
- **remote_pathes** – it is a dict mapping a treatment name to an existing remote dir which will be used to store ROI data and result files
- **local_result_pathes** – it is a dict mapping a treatment name to a local path where final results will be sorted (host will send it there by scp)
- **label_for_cluster** – it is the base name used to label workflows and sub jobs

pyhrf.parallel.**save_treatment**(*t*, *f*)

2.9 pyhrf.parcellation module

class pyhrf.parcellation.**Ant**(*a_id*, *greed*, *graph*, *labels*, *path_marks*, *site_marks*, *pressures*, *world*, *verbosity=0*)

Bases: *pyhrf.parcellation.Talker*

action(*time*)

fix_explosion()

to_conquer()

to_patrol()

class pyhrf.parcellation.**Talker**(*talker_string_id*, *verbosity=0*)

verbose(*level*, *msg*)

verbose_array(*level*, *array*)

pyhrf.parcellation.**Visit_graph_noeud**(*noeud*, *graphe*, *Visited=None*)

class pyhrf.parcellation.**World**(*graph*, *nb_ants*, *greed=0.05*, *time_min=100*, *time_max=None*, *tolerance=1*, *verbosity=0*, *stop_when_all_controlled=False*)

Bases: *pyhrf.parcellation.Talker*

balanced()

force_end()

get_final_labels()

resolve()

site_taken(*site*)

pyhrf.parcellation.**init_edge_data**(*g*, *init_value=0*)

pyhrf.parcellation.**make_parcellation_cubed_blobs_from_file**(*parcellation_file*, *output_path*, *roi_ids=None*, *bg_parcel=0*, *skip_existing=False*)

pyhrf.parcellation.**make_parcellation_from_files**(*betaFiles*, *maskFile*, *outFile*, *nparcels*, *method*, *dry=False*, *spatial_weight=10.0*)

pyhrf.parcellation.**make_parcellation_surf_from_files**(*beta_files*, *mesh_file*, *parcellation_file*, *nbparcel*, *method*, *mu=10.0*, *verbose=0*)

pyhrf.parcellation.**parcellate_balanced_vol**(*mask*, *nb_parcels*)

Performs a balanced partitioning on the input mask using a balloon patroling algorithm [Eurol 2009]. Values with 0 are discarded position in the mask.

Parameters

- **mask** (–) – binary 3D array of valid position to parcellate
- **nb_parcels** (–) – the required number of parcels

Returns a 3D array of integers

Return type

- the parcellation (numpy.ndarray)

pyhrf.parcellation.**parcellate_voronoi_vol**(*mask*, *nb_parcels*, *seeds=None*)

Produce a parcellation from a Voronoi diagram built on random seeds. The number of seeds is equal to the nb of parcels. Seed are randomly placed within the mask, expect on edge positions

Parameters

- **mask** (–) – binary 3D array of valid position to parcellate
- **nb_parcels** (–) – the required number of parcels
- **seeds** (–) – TODO

Returns a 3D array of integers -

Return type

- the parcellation (numpy.ndarray)

pyhrf.parcellation.**parcellation_dist**(*p1*, *p2*, *mask=None*)

Compute the distance between the two parcellation p1 and p2 as the minimum number of positions to remove in order to obtain equal partitions. "mask" may be a binary mask to limit the distance computation to some specific positions. Important convention: parcel label 0 is treated as background and corresponding positions are discarded if no mask is provided.

Returns (distance value, parcellation overlap)

pyhrf.parcellation.**parcellation_for_jde**(*fmri_data*, *avg_parcel_size=250*, *output_dir=None*, *method='gkm'*, *glm_drift='Cosine'*, *glm_hfcut=128*)

method: gkm, ward, ward_and_gkm

pyhrf.parcellation.**parcellation_report**(*d*)

pyhrf.parcellation.**parcellation_ward_spatial**(*func_data*, *n_clusters*, *graph=None*)

Make parcellation based upon ward hierarchical clustering from scikit-learn

Parameters

- **func_data** (*array of shape (nb_positions, dim_feature_1, [dim_feature2, ..])*) – functional data:
- **n_clusters** – chosen number of clusters to create
- **graph** – adjacency list defining neighbours. if None, no connectivity defined: clustering is spatially independent

Returns

Return type parcellation labels

pyhrf.parcellation.**permutation**(*x*)

Randomly permute a sequence, or return a permuted range.

If *x* is a multi-dimensional array, it is only shuffled along its first index.

Parameters **x** (*int or array_like*) – If *x* is an integer, randomly permute `np.arange(x)`. If *x* is an array, make a copy and shuffle the elements randomly.

Returns **out** – Permuted sequence or array range.

Return type *ndarray*

Examples

```
>>> np.random.permutation(10)
array([1, 7, 4, 3, 0, 9, 2, 5, 8, 6])
```

```
>>> np.random.permutation([1, 4, 9, 12, 15])
array([15,  1,  9,  4, 12])
```

```
>>> arr = np.arange(9).reshape((3, 3))
>>> np.random.permutation(arr)
array([[6, 7, 8],
       [0, 1, 2],
       [3, 4, 5]])
```

pyhrf.parcellation.**rand**(*d0*, *d1*, ..., *dn*)

Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters **d1, .., dn** (*d0,*) – The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.

Returns **out** – Random values.

Return type ndarray, shape `(d0, d1, ..., dn)`

See also:

`random()`

Notes

This is a convenience function. If you want an interface that takes a shape-tuple as the first argument, refer to np.random.random_sample .

Examples

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

pyhrf.parcellation.**randint**(*low*, *high=None*, *size=None*, *dtype='l'*)

Return random integers from *low* (inclusive) to *high* (exclusive).

Return random integers from the "discrete uniform" distribution of the specified dtype in the "half-open" interval [*low*, *high*). If *high* is None (the default), then results are from [0, *low*).

Parameters

- **low** (*int*) – Lowest (signed) integer to be drawn from the distribution (unless `high=None`, in which case this parameter is the *highest* such integer).
- **high** (*int, optional*) – If provided, one above the largest (signed) integer to be drawn from the distribution (see above for behavior if `high=None`).
- **size** (*int or tuple of ints, optional*) – Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. Default is None, in which case a single value is returned.
- **dtype** (*dtype, optional*) – Desired dtype of the result. All dtypes are determined by their name, i.e., 'int64', 'int', etc, so byteorder is not available and a specific precision may have different C types depending on the platform. The default value is 'np.int'.

  New in version 1.11.0.

Returns **out** – *size*-shaped array of random integers from the appropriate distribution, or a single such random int if *size* not provided.

Return type int or ndarray of ints

See also:

random.random_integers() similar to *randint*, only for the closed interval [*low*, *high*], and 1 is the lowest value if *high* is omitted. In particular, this other one is the one to use to generate uniformly distributed discrete non-integers.

Examples

```
>>> np.random.randint(2, size=10)
array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0])
>>> np.random.randint(1, size=10)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Generate a 2 x 4 array of ints between 0 and 4, inclusive:

```
>>> np.random.randint(5, size=(2, 4))
array([[4, 0, 2, 1],
       [3, 2, 2, 0]])
```

pyhrf.parcellation.**random_pick**(*a*)

pyhrf.parcellation.**round_nb_parcels**(*n*)

pyhrf.parcellation.**split_big_parcels**(*parcel_file*, *output_file*, *max_size=400*)

pyhrf.parcellation.**split_parcel**(*labels*, *graphs*, *id_parcel*, *n_parcels*, *inplace=False*, *verbosity=0*, *balance_tolerance='exact'*)

balance_tolerance : exact or draft

2.10 pyhrf.plot module

pyhrf.plot.**autocrop**(*img_fn*)

Remove extra background within figure (inplace). Use ImageMagick (convert)

pyhrf.plot.**flip**(*img_fn*, *direction='horizontal'*)

Mirror the figure (inplace). Use ImageMagick (convert) 'horizontal' direction -> use -flop. 'vertical' direction -> use -flip.

pyhrf.plot.**mix_cmap**(*img1*, *cmap1*, *img2*, *cmap2*, *norm1=None*, *norm2=None*, *blend_r=0.5*)

pyhrf.plot.**plot_anat_parcel_func_fusion**(*anat*, *func*, *parcel*, *parcel_col='white'*, *parcels_line_width=0.5*, *func_cmap=None*, *func_norm=None*, *anat_norm=None*, *func_mask=None*, *highlighted_parcels_col=None*, *highlighted_parcels_line_width=1.5*)

pyhrf.plot.**plot_cub_as_curve**(*c*, *colors=None*, *plot_kwargs=None*, *legend_prefix=''*, *show_axis_labels=True*, *show_legend=False*, *axes=None*, *axis_label_fontsize=12*)

Plot a cuboid (ndims <= 2) as curve(s).

- If the input is 1D: one single curve.
- **If the input is 2D:**
  - multiple curves are plotted: one for each domain value on the 1st axis.
  - legends are shown to display which domain value is associated to which curve.

Parameters

- **colors** (*dict <domain value>: <matplotlib color>*) – associate domain values of the 1st axis to color curves
- **plot_kwargs** (*dict <arg name>:<arg value>*) – dictionary of named argument passed to the plot function
- **legend_prefix** (*str*) – prefix to prepend to legend labels.

Returns

Return type None

pyhrf.plot.**plot_cub_as_image**(*c*, *cmap=None*, *norm=None*, *show_axes=True*, *show_axis_labels=True*, *show_tick_labels=True*, *show_colorbar=False*, *axes=None*)

pyhrf.plot.**plot_func_slice**(*func_slice_data*, *anatomy=None*, *parcellation=None*, *parcel_col='white'*, *parcels_line_width=2.5*, *func_cmap=None*, *func_norm=None*, *anat_norm=None*, *func_mask=None*, *highlighted_parcels_col=None*, *highlighted_parcels_line_width=2.5*, *resolution=None*, *crop_extension=None*, *blend=0.5*)

pyhrf.plot.**plot_gaussian_mixture**(*values*, *props=None*, *color='k'*, *lw=1.75*)
axes of values : (component,class)

pyhrf.plot.**plot_gaussian_pdf**(*bins*, *m*, *v*, *prop=None*, *plotArgs={}*)

pyhrf.plot.**plot_palette**(*cmap*, *norm=None*, *fontsize=None*)

pyhrf.plot.**plot_spm_mip**(*img_fn*, *mip_fn*)

pyhrf.plot.**rotate**(*img_fn*, *angle*)
Rotate figure (inplace). Use ImageMagick (convert)

pyhrf.plot.**set_int_tick_labels**(*axis*, *labels*, *fontsize=None*, *rotation=None*)
Redefine labels of visible ticks at integer positions for the given axis.

pyhrf.plot.**set_ticks_fontsize**(*fontsize*, *colbar=None*)
Change the fontsize of the tick labels for the current figure. If colorbar (Colorbar instance) is provided then change the fontsize of its tick labels as well.

pyhrf.plot.**set_xticklabels**(*labels*, *positions=None*, *rotation=None*)
Set ticks labels of the xaxis in the current figure to *labels* If positions is provided then enforce tick position.

2.11 pyhrf.rfir module

class pyhrf.rfir.**RFIREstim**(*hrf_nb_coeffs=42*, *hrf_dt=0.6*, *drift_type='cosine'*, *stop_crit1=0.0001*, *stop_crit2=1e-05*, *nb_its_max=5*, *nb_iterations=500*, *nb_its_min=1*, *average_bold=False*, *taum=0.01*, *lambda_reg=100.0*, *fixed_taum=False*, *discarded_scan_indexes=None*, *output_fit=False*)

Bases: *pyhrf.xmlio.Initable*

Class handling the estimation of HRFs from fMRI data. Analysis is voxel-wise and can be multissession (heteroscedastic noise and session dependent drift). Simultaneous analysis of several conditions is treated. One HRF is considered at each voxel.

Compute_INV_R_and_R_and_DET_R()
Both computes *self.InvR* and *self.DetR*

Requires:

- K-1
- InvR initialized

Compute_onset_matrix3()
Computes the onset matrix. Each stimulus onset is considered over a period of *LengthOnsets* seconds if (*LengthOnsets* > *DetlaT*) and a time step otherwise.

Requires:

- X initialized
- OnsetList
- TR
- DeltaT
- K
- LengthOnsets

where *self.X[i][m,n,k]* is such that:

- session i (in 0:I-1)
- condition m (in 0:M-1)
- data nb n (in 0:Ni[i]-1)
- hrf coef nb k (in 0:K-2)

CptFctQ(*CptType*)
Computes the function $Q(\Theta', \tilde{\Theta}; y)$ at a given iteration

Notes

It requires:

- All parameters and hyperparameters
- Sigma
- InvR
- CptType = 'K_Km1' or 'K_K'

CptSigma()
Computes the Sigma at a given iteration.

*self.Sigma[m*SBS:(m+1)*SBS,n*SBS:(n+1)*SBS]] -> (m,n)^th* block of Sigma in session *i*.

EM_solver(*POI*)
requires: * everything in the class is supposed initialized

InitMatrixAndVectors(*POI*)
Initialize to zeros: X, y, P, l, h, InvR, Sigma. Initialize to ones: TauM, rb (<-scalar).

InitStorageMat()
Initialization of the matrices that will store all voxel results.

Notes

Input signals must have been read (in ReadRealSignal)

ReadPointOfInterestData(*POI*)
Initialize the parameters for a voxel analysis. The voxel ID is 'POI' in 'ConsideredCoord' initialized in 'ReadRealSignal'

Notes

Input signals must have been read (in ReadRealSignal)

StoreRes(*POI*)
Store results computed in the voxel defined in POI.

Notes

The estimation at this voxel must have been performed

buildCosMat(*fctNb*, *tr*, *ny*)
Build a cosine low frequency basis in P (adapted from *samplerbase.py*)

Parameters

- **fctNb** – columns number in the current session
- **tr** – the time resolution of the BOLD data (in second)
- **ny** – number of data for the current session

buildLowFreqMat()
Build the low frequency basis matrix P.

buildPolyMat(*fctNb*, *tr*, *ny*)
Build a polynomial low frequency basis in P (adapted from *samplerbase.py*)

Parameters

- **fctNb** – columns number in the current session
- **tr** – the time resolution of the BOLD data (in second)
- **ny** – number of data for the current session

Notes

- there may have no constant column in the orthogonal matrix (the algorithm suppose there is one such column)
- the columns number is not always as expected

clean_memory()
Clean all objects that are useless for outputs

compute_fit(*POI*)

cpt_XSigmaX(*tempTerm2i*, *SBS*, *i*)

default_nb_its = 500

default_stop_crit1 = 0.0001

default_stop_crit2 = 1e-05

getOutputs()

linkToData(*data*)

parametersComments = {'hrf_dt': 'Required HRF temporal resolution', 'hrf_nb_coeffs':

parametersToShow = ['hrf_nb_coeffs', 'hrf_dt', 'drift_type', 'nb_iterations']

run()
function to launch the analysis

pyhrf.rfir.**init_dict**()

pyhrf.rfir.**rfir**(*func_data*, *fir_duration=42*, *fir_dt=0.6*, *nb_its_max=100*, *nb_its_min=5*, *fixed_taum=False*, *lambda_reg=100.0*)
Fit a Regularized FIR on functional data *func_data*:

- multisession voxel-based fwd model: y = sum Xh + Pl + b
- heteroscedastic noise
- session dependent drift coefficients
- one HRF per condition
- solved by Expectation-Minimization (EM) (iterative scheme)

Reference: "Unsupervised robust non-parametric estimation of the hemodynamic response function for any fMRI experiment." Ciuciu, J.-B. Poline, G. Marrelec, J. Idier, Ch. Pallier, and H. Benali. IEEE Trans. Med. Imag., 22(10):1235-1251, Oct. 2003.

Parameters

- **func_data** (pyhrf.core.FmriData) –
- **fir_duration** (*float*) – FIR duration in seconds
- **fir_dt** (*float*) – FIR temporal resolution
- **fixed_taum** (*bool*) – enable faster (drafter) RFIR version where the HRF variance hyper-parameter is fixed.
- **lambda_reg** (*float*) – amount of temporal regularization for the HRF. Only used if *fixed_taum* is true.
- **nb_its_min** – minimum number of iterations for the EM
- **nb_its_max** – maximum number of iterations for the EM

Returns

Return type dict of xndarray instances

2.12 pyhrf.surface module

pyhrf.surface.**create_projection_kernels**(*input_mesh*, *output_kernel*, *resolution*, *geod_decay=5.0*, *norm_decay=2.0*, *size=7*)

pyhrf.surface.**extract_sub_mesh**(*cor*, *tri*, *center_node*, *radius*)

pyhrf.surface.**extract_sub_mesh_with_files**(*input_mesh*, *center_node*, *radius*, *output_mesh=None*)

pyhrf.surface.**mesh_contour**(*coords*, *triangles*, *labels*)

pyhrf.surface.**mesh_contour_with_files**(*input_mesh*, *input_labels*, *output_mesh=None*, *output_labels=None*)
TODO: use nibabel here

pyhrf.surface.**project_fmri**(*input_mesh*, *data_file*, *output_tex_file*, *output_kernels_file=None*, *data_resolution=None*, *geod_decay=5.0*, *norm_decay=2.0*, *kernel_size=7*, *tex_bin_threshold=None*)

pyhrf.surface.**project_fmri_from_kernels**(*input_mesh*, *kernels_file*, *fmri_data_file*, *output_tex*, *bin_threshold=None*)

2.13 pyhrf.usemode module

2.14 pyhrf.version module

2.15 pyhrf.xmlio module

exception pyhrf.xmlio.**DeprecatedXMLFormatException**
Bases: exceptions.Exception

class pyhrf.xmlio.**Initable**
Bases: object

Abstract class to keep track of how an object is initialised. To do so, it stores the init function and its parameters. The aim is to use it in a user interface or to serialize objects. It also allows to add comments and meta info on init parameters.

assert_is_initialized()

check_init_obj(*params=None*)
check if the function used for init can be used in this API -> must allow **kwargs and *args. All arguments must have a value: either a default one or specified in the input dict *params*

from_ui_node

get_arg_for_ui(*a*)

get_arg_from_ui(*a*)

get_init_func()

get_parameters_comments()

get_parameters_meta()

get_parameters_to_show()

init_new_obj()
Creates a new instance

set_arg_translation(*a*, *t*)
Set the display name of argument *a* as *t*

set_init(*init_obj*, ***init_params*)
Override init function with *init_obj* and use *init_params* as new init parameters. *init_obj* must return an instance of the same class as the current object. Useful when the object is not instanciated via its __init__ function but eg a static method.

set_init_param(*param_name*, *param_value*)

to_ui_node(*label*, *parent=None*)

class pyhrf.xmlio.**UiNode**(*label*, *parent=None*, *attributes=None*)
Bases: object

Store data hierarchically to be used in a Qt model/tree view setting. Also store additional node-specific data as attributes (in a dict). Attributes must only contain strings.

The resulting data structure is:

```
   col 0         | col 1
|- <node_label> | <node_attributes>                    #row 0
    |
    |  col 0               | col 1
    |- <child_node_label> | <child_node_attributes>   #row 0
        |
        |...
...
```

This structure is similar to DOM.

Features:

- can be instantiated from any python object that can be serialized into human-readable strings: bool, int, float, string, numpy scalars, numpy arrays. It also support container types (list, dict, OrderedDict) as long as their items are also serializable into human-readable strings.

See static method *from_py_object*.

add_child(*child*)

child(*row*)

childCount()

from_py_object

from_xml

get_attribute(*attr_name*)

get_children()

has_attribute(*attr_name*)

is_leaf_node()

label()

log(*tabLevel=-1*)

serialize_attributes()

set_attribute(*attr_name*, *attr_value*)

set_label(*label*)

to_xml(*pretty=False*)
Return an XML representation (str) of the Node and its children.

type_info()

unserialize_attributes

pyhrf.xmlio.**XmlInitable**
alias of *Initable*

pyhrf.xmlio.**from_xml**(*sxml*)

pyhrf.xmlio.**numpy_array_from_string**(*s*, *sdtype*, *sshape=None*)

pyhrf.xmlio.**protect_xml_attr**(*sa*)

pyhrf.xmlio.**read_xml**(*fn*)

pyhrf.xmlio.**to_xml**(*obj*, *label='anonym'*, *pretty=False*)
Return an XML representation of the init state of the given object *obj*.

pyhrf.xmlio.**unprotect_xml_attr**(*sa*)

pyhrf.xmlio.**write_xml**(*obj*, *fn*)

Python Module Index

p

pyhrf, 3
pyhrf.boldsynth, 3
pyhrf.boldsynth.field, 7
pyhrf.boldsynth.hrf, 8
pyhrf.boldsynth.pottsfield, 3
pyhrf.boldsynth.pottsfield.swendsenwang, 3
pyhrf.boldsynth.scenarios, 9
pyhrf.boldsynth.spatialconfig, 14
pyhrf.configuration, 229
pyhrf.core, 229
pyhrf.glm, 234
pyhrf.graph, 234
pyhrf.grid, 236
pyhrf.jde, 18
pyhrf.jde.asl, 35
pyhrf.jde.asl_2steps, 40
pyhrf.jde.asl_physio, 41
pyhrf.jde.asl_physio_1step, 47
pyhrf.jde.asl_physio_1step_params, 53
pyhrf.jde.asl_physio_det_fwdm, 60
pyhrf.jde.asl_physio_hierarchical, 68
pyhrf.jde.asl_physio_joint, 75
pyhrf.jde.beta, 80
pyhrf.jde.drift, 86
pyhrf.jde.hrf, 87
pyhrf.jde.jde_multi_sess, 91
pyhrf.jde.jde_multi_sujets, 99
pyhrf.jde.jde_multi_sujets_alpha, 106
pyhrf.jde.models, 115
pyhrf.jde.noise, 122
pyhrf.jde.nrl, 18
pyhrf.jde.nrl.ar, 18
pyhrf.jde.nrl.base, 19
pyhrf.jde.nrl.bigaussian, 19
pyhrf.jde.nrl.bigaussian_drift, 25
pyhrf.jde.nrl.gammagaussian, 31
pyhrf.jde.nrl.habituation, 32
pyhrf.jde.nrl.trigaussian, 34
pyhrf.jde.samplerbase, 124
pyhrf.jde.wsampler, 126
pyhrf.ndarray, 238
pyhrf.paradigm, 246
pyhrf.parallel, 247
pyhrf.parcellation, 249
pyhrf.plot, 253
pyhrf.rfir, 254
pyhrf.sandbox, 127
pyhrf.sandbox.data_parser, 127
pyhrf.sandbox.func_BMA_consensus_clustering, 128
pyhrf.sandbox.make_parcellation, 129
pyhrf.sandbox.parcellation, 129
pyhrf.sandbox.physio, 135
pyhrf.sandbox.physio_params, 140
pyhrf.sandbox.stats, 143
pyhrf.stats, 145
pyhrf.stats.misc, 145
pyhrf.stats.random, 146
pyhrf.surface, 257
pyhrf.test, 149
pyhrf.test.analysertest, 150
pyhrf.test.boldsynthTest, 150
pyhrf.test.commandTest, 152
pyhrf.test.core_test, 153
pyhrf.test.graphtest, 153
pyhrf.test.iotest, 153
pyhrf.test.jdetest, 155
pyhrf.test.rfir_test, 156
pyhrf.test.seppotest, 156
pyhrf.test.statsTest, 156
pyhrf.test.test, 156
pyhrf.test.test_glm, 158
pyhrf.test.test_jde_multi_subj, 158
pyhrf.test.test_jde_vem_asl, 158
pyhrf.test.test_jde_vem_bold, 159
pyhrf.test.test_jde_vem_tools, 159
pyhrf.test.test_jde_vem_tools_asl, 160

pyhrf.test.test_jde_vem_tools_UtilsC, 160
pyhrf.test.test_ndarray, 161
pyhrf.test.test_paradigm, 162
pyhrf.test.test_parallel, 162
pyhrf.test.test_parcellation, 163
pyhrf.test.test_plot, 163
pyhrf.test.test_rfir, 164
pyhrf.test.test_sampler, 164
pyhrf.test.test_sandbox_physio, 164
pyhrf.test.test_treatment, 165
pyhrf.test.test_xml, 165
pyhrf.test.toolsTest, 167
pyhrf.tools, 169
pyhrf.tools.aexpression, 169
pyhrf.tools.backports, 170
pyhrf.tools.cpus, 170
pyhrf.tools.message, 171
pyhrf.tools.misc, 172
pyhrf.ui, 178
pyhrf.ui.analyser_ui, 178
pyhrf.ui.glm_analyser, 179
pyhrf.ui.glm_ui, 179
pyhrf.ui.jde, 179
pyhrf.ui.rfir_ui, 180
pyhrf.ui.treatment, 182
pyhrf.ui.vb_jde_analyser, 185
pyhrf.ui.vb_jde_analyser_asl_fast, 187
pyhrf.ui.vb_jde_analyser_bold_fast, 188
pyhrf.usemode, 258
pyhrf.validation, 189
pyhrf.validation.config, 190
pyhrf.validation.valid_beta_estim, 190
pyhrf.validation.valid_jde_asl, 191
pyhrf.validation.valid_jde_asl_physio, 192
pyhrf.validation.valid_jde_asl_physio_alpha, 193
pyhrf.validation.valid_jde_bold_mono_subj_multi_sess, 194
pyhrf.validation.valid_jde_bold_mono_subj_sess, 195
pyhrf.validation.valid_jde_vem_asl, 195
pyhrf.validation.valid_rfir, 196
pyhrf.validation.valid_rndm_field, 197
pyhrf.validation.valid_sandbox_parcellation, 197
pyhrf.vbjde, 199
pyhrf.vbjde.vem_asl_models_fast, 199
pyhrf.vbjde.vem_asl_models_fast_ms, 200
pyhrf.vbjde.vem_bold, 201
pyhrf.vbjde.vem_bold_constrained, 203
pyhrf.vbjde.vem_bold_models_fast, 205
pyhrf.vbjde.vem_bold_models_fast_ms, 206
pyhrf.vbjde.vem_tools, 207
pyhrf.version, 258
pyhrf.xmlio, 258
pyhrf.xmliobak, 223
pyhrf.xmliobak.xmlbase, 223
pyhrf.xmliobak.xmlmatlab, 226
pyhrf.xmliobak.xmlnumpy, 227

Index

A

A (class in pyhrf.test.test_xml), 165
A_Entropy() (in module pyhrf.vbjde.vem_tools), 207
abnormal_stop() (pyhrf.grid.OneTaskManager method), 236
abnormal_stop() (pyhrf.grid.RepeatedTasksManager method), 237
abnormal_stop() (pyhrf.grid.TasksManager method), 237
acorr() (in module pyhrf.stats.misc), 145
action() (pyhrf.parcellation.Ant method), 249
add() (pyhrf.ndarray.xndarray method), 241
add_child() (pyhrf.xmlio.UiNode method), 259
add_prefix() (in module pyhrf.tools.misc), 173
add_root() (pyhrf.tools.misc.Pipeline method), 172
add_suffix() (in module pyhrf.tools.misc), 173
addDefaultFunctions() (pyhrf.tools.aexpression.ArithmeticExpression method), 169
addDefaultVariables() (pyhrf.tools.aexpression.ArithmeticExpression method), 169
AgglomerationTransform (class in pyhrf.sandbox.parcellation), 129
align_parcellation() (in module pyhrf.sandbox.parcellation), 131
Alpha_hgroup_Sampler (class in pyhrf.jde.jde_multi_sujets_alpha), 106
AlphaVar_Sampler (class in pyhrf.jde.jde_multi_sujets_alpha), 106
already_done() (pyhrf.ui.treatment.FMRITreatment method), 182
analyse() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
analyse_roi() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
analyse_roi() (pyhrf.ui.glm_analyser.GLMAnalyser method), 179
analyse_roi() (pyhrf.ui.glm_ui.GLMAnalyser method), 179
analyse_roi() (pyhrf.ui.jde.JDEMCMCAnalyser method), 179
analyse_roi() (pyhrf.ui.rfir_ui.RFIRAnalyser method), 180
analyse_roi() (pyhrf.ui.vb_jde_analyser.JDEVEMAnalyser method), 186
analyse_roi() (pyhrf.ui.vb_jde_analyser_asl_fast.JDEVEMAnalyser method), 187
analyse_roi() (pyhrf.ui.vb_jde_analyser_bold_fast.JDEVEMAnalyser method), 188
analyse_roi_wrap() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
analyse_roi_wrap_bak() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
AnsiColorizer (class in pyhrf.tools.misc), 172
Ant (class in pyhrf.parcellation), 249
append() (pyhrf.grid.TaskList method), 237
append_common_treatment_options() (in module pyhrf.ui.treatment), 183
appendParametersToDOMTree() (pyhrf.xmliobak.xmlbase.XMLParamDrivenClass method), 225
apply_to_dict() (in module pyhrf.sandbox.data_parser), 128
apply_to_leaves() (in module pyhrf.tools.misc), 173
ArithmeticExpression (class in pyhrf.tools.aexpression), 169
ArithmeticExpressionNameError, 169
ArithmeticExpressionSyntaxError, 170
ARN_BiG_BOLDSamplerInput (class in pyhrf.jde.models), 115
array_summary() (in module pyhrf.tools.misc), 173
arrayDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 224
arrayDOMWriter() (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler static method), 227
ArrayMappingError, 238
arrayTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 224
arrayTagDOMReader() (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler static method), 227
ASLPhysioHierarchicalTest (class in

pyhrf.validation.valid_jde_asl_physio), 192
ASLPhysioSampler (class in pyhrf.jde.asl_physio), 41
ASLPhysioSampler (class in pyhrf.jde.asl_physio_1step), 47
ASLPhysioSampler (class in pyhrf.jde.asl_physio_1step_params), 53
ASLPhysioSampler (class in pyhrf.jde.asl_physio_det_fwdm), 60
ASLPhysioSampler (class in pyhrf.jde.asl_physio_hierarchical), 68
ASLPhysioSampler (class in pyhrf.jde.asl_physio_joint), 75
ASLPhysioTest (class in pyhrf.test.jdetest), 155
ASLSampler (class in pyhrf.jde.asl), 35
ASLTest (class in pyhrf.test.jdetest), 155
ASLTest (class in pyhrf.validation.valid_jde_asl), 191
ASLTest (class in pyhrf.validation.valid_jde_asl_physio), 192
ASLTest (class in pyhrf.validation.valid_jde_asl_physio_alphm), 193
ASLTest (class in pyhrf.validation.valid_jde_vem_asl), 195
assert_file_exists() (in module pyhrf.tools.misc), 173
assert_file_exists() (pyhrf.test.iotest.RxCopyTest method), 154
assert_html_equal() (pyhrf.test.test_ndarray.TestHtml method), 161
assert_is_initialized() (pyhrf.xmlio.Initable method), 258
assert_parcellation_equal() (in module pyhrf.sandbox.parcellation), 131
assert_path_not_in_src() (in module pyhrf.tools.misc), 173
astype() (pyhrf.ndarray.xndarray method), 241
AttrClass (class in pyhrf.core), 229
ATTRIBUTE_LABEL_META (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
ATTRIBUTE_LABEL_PYTHON_CLASS (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
ATTRIBUTE_LABEL_PYTHON_CLASS_INIT_MODE (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
ATTRIBUTE_LABEL_PYTHON_FUNCTION (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
ATTRIBUTE_LABEL_PYTHON_MODULE (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
ATTRIBUTE_LABEL_TYPE (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
attrs_to_string() (in module pyhrf.tools.misc), 173
autocrop() (in module pyhrf.plot), 253
available_cpu_count() (in module pyhrf.tools.cpus), 170
available_status (pyhrf.grid.HostsManager attribute), 236
average() (pyhrf.core.FmriData method), 231

B

B (class in pyhrf.test.test_xml), 165
b() (in module pyhrf.jde.asl), 39
b() (in module pyhrf.jde.asl_physio), 46
b() (in module pyhrf.jde.asl_physio_1step), 52
b() (in module pyhrf.jde.asl_physio_1step_params), 60
b() (in module pyhrf.jde.asl_physio_det_fwdm), 66
b() (in module pyhrf.jde.asl_physio_hierarchical), 74
b() (in module pyhrf.jde.asl_physio_joint), 80
b() (in module pyhrf.jde.jde_multi_sess), 96
b() (in module pyhrf.jde.jde_multi_sujets), 104
b() (in module pyhrf.jde.jde_multi_sujets_alpha), 114
balanced() (pyhrf.parcellation.World method), 249
BaseEstimator (class in pyhrf.sandbox.parcellation), 129
BaseTest (class in pyhrf.test.test_xml), 165
BEGINC (pyhrf.tools.misc.AnsiColorizer attribute), 172
beta_estim_obs_field() (in module pyhrf.jde.beta), 86
beta_estim_obs_field_mc() (in module pyhrf.validation.valid_beta_estim), 190
beta_gradient() (in module pyhrf.vbjde.vem_tools), 208
beta_maximization() (in module pyhrf.vbjde.vem_tools), 208
BetaEstimESTest (class in pyhrf.test.analysertest), 150
BetaGenerator (class in pyhrf.stats.random), 146
BetaSampler (class in pyhrf.jde.beta), 80
bezierCurve() (in module pyhrf.boldsynth.hrf), 8
bfs_set_label() (in module pyhrf.graph), 234
bfs_sub_graph() (in module pyhrf.graph), 234
BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler (class in pyhrf.jde.jde_multi_sess), 92
BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler (class in pyhrf.jde.nrl.bigaussian_drift), 25
BiGaussMixtureParamsSampler (class in pyhrf.jde.jde_multi_sujets_alpha), 108
BiGaussMixtureParamsSampler (class in pyhrf.jde.nrl.bigaussian), 19
BiGaussMixtureParamsSamplerWithRelVar (class in pyhrf.jde.nrl.bigaussian), 19
BiGaussMixtureParamsSamplerWithRelVar_OLD (class in pyhrf.jde.nrl.bigaussian), 20
BMA_consensus_cluster_parallel() (in module pyhrf.sandbox.func_BMA_consensus_clustering), 128
bold (pyhrf.core.FmriData attribute), 231
BOLDGibbs_Multi_SessSampler (class in pyhrf.jde.jde_multi_sess), 91
BOLDGibbs_Multi_SubjSampler (class in pyhrf.jde.jde_multi_sujets), 99
BOLDGibbs_Multi_SubjSampler (class in pyhrf.jde.jde_multi_sujets_alpha), 106

BOLDGibbsSampler (class in pyhrf.jde.models), 115
BOLDGibbsSampler_AR (class in pyhrf.jde.models), 116
BOLDMixtureSampler (class in pyhrf.jde.asl), 35
BOLDMixtureSampler (class in pyhrf.jde.asl_physio), 42
BOLDMixtureSampler (class in pyhrf.jde.asl_physio_1step), 47
BOLDMixtureSampler (class in pyhrf.jde.asl_physio_1step_params), 53
BOLDMixtureSampler (class in pyhrf.jde.asl_physio_det_fwdm), 61
BOLDMixtureSampler (class in pyhrf.jde.asl_physio_hierarchical), 68
BOLDMixtureSampler (class in pyhrf.jde.asl_physio_joint), 76
BOLDResponseLevelSampler (class in pyhrf.jde.asl), 36
BOLDResponseLevelSampler (class in pyhrf.jde.asl_physio), 42
BOLDResponseLevelSampler (class in pyhrf.jde.asl_physio_1step), 48
BOLDResponseLevelSampler (class in pyhrf.jde.asl_physio_1step_params), 54
BOLDResponseLevelSampler (class in pyhrf.jde.asl_physio_det_fwdm), 61
BOLDResponseLevelSampler (class in pyhrf.jde.asl_physio_hierarchical), 69
BOLDResponseLevelSampler (class in pyhrf.jde.asl_physio_joint), 76
BOLDResponseSampler (class in pyhrf.jde.asl), 36
BOLDResponseVarianceSampler (class in pyhrf.jde.asl), 36
BOLDSampler_Multi_SessInput (class in pyhrf.jde.jde_multi_sess), 92
BOLDSampler_Multi_SessInput (class in pyhrf.jde.models), 118
BOLDSampler_MultiSujInput (class in pyhrf.jde.jde_multi_sujets), 100
BOLDSampler_MultiSujInput (class in pyhrf.jde.jde_multi_sujets_alpha), 108
BOLDSamplerInput (class in pyhrf.jde.models), 117
boolDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
boolTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
breadth_first_search() (in module pyhrf.graph), 234
broken_help() (in module pyhrf.grid), 238
build_ctrl_tag_matrix() (in module pyhrf.boldsynth.scenarios), 9
build_graphs() (pyhrf.core.FmriData method), 231
build_graphs() (pyhrf.core.FmriGroupData method), 233
buildCosMat() (pyhrf.jde.jde_multi_sess.BOLDSampler_Multi_SessInput method), 92
buildCosMat() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSujInput method), 100
buildCosMat() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_MultiSujInput method), 108
buildCosMat() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
buildCosMat() (pyhrf.jde.models.BOLDSamplerInput method), 117
buildCosMat() (pyhrf.rfir.RFIREstim method), 256
buildDiagGaussianMat() (in module pyhrf.jde.hrf), 90
buildFiniteDiffMatrix() (in module pyhrf.boldsynth.hrf), 8
buildFiniteDiffMatrix() (in module pyhrf.vbjde.vem_tools), 209
buildLowFreqMat() (pyhrf.rfir.RFIREstim method), 256
buildNeighboursCoordLists() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 14
buildNeighboursIndexLists() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 14
buildOrder1FiniteDiffMatrix_central() (in module pyhrf.sandbox.physio), 135
buildOrder1FiniteDiffMatrix_central() (in module pyhrf.sandbox.physio_params), 140
buildOtherMatX() (pyhrf.jde.jde_multi_sess.BOLDSampler_Multi_SessInput method), 92
buildOtherMatX() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSujInput method), 100
buildOtherMatX() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_MultiSujInput method), 108
buildOtherMatX() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
buildOtherMatX() (pyhrf.jde.models.BOLDSamplerInput method), 117
buildParadigmConvolMatrix() (pyhrf.jde.jde_multi_sess.BOLDSampler_Multi_SessInput method), 92
buildParadigmConvolMatrix() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSujInput method), 100
buildParadigmConvolMatrix() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_MultiSujInput method), 108
buildParadigmConvolMatrix() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
buildParadigmConvolMatrix() (pyhrf.jde.models.BOLDSamplerInput method), 117
buildParadigmSingleCondMatrix() (pyhrf.jde.models.BOLDSamplerInput method), 117
buildPolyMat() (in module pyhrf.tools.misc), 173
buildPolyMat() (pyhrf.jde.jde_multi_sess.BOLDSampler_Multi_SessInput method), 92

buildPolyMat() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSuj method), 100
buildPolyMat() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_MultiSuj method), 108
buildPolyMat() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
buildPolyMat() (pyhrf.jde.models.BOLDSamplerInput method), 117
buildPolyMat() (pyhrf.rfir.RFIREstim method), 256
buildSharedDataTree() (pyhrf.jde.models.BOLDGibbsSampler method), 116
buildSharedDataTree() (pyhrf.jde.models.BOLDGibbsSampler method), 117
buildXMLString() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225

C

C (class in pyhrf.test.test_xml), 166
c (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping attribute), 14
cache_exists() (in module pyhrf.tools.misc), 173
cache_filename() (in module pyhrf.tools.misc), 173
cached_eval() (in module pyhrf.tools.misc), 173
CachedEvalTest (class in pyhrf.test.toolsTest), 167
calc_asl_shape() (in module pyhrf.boldsynth.scenarios), 9
calc_linear_rfs() (in module pyhrf.sandbox.physio), 135
calc_linear_rfs() (in module pyhrf.sandbox.physio_params), 140
calc_nc2D() (in module pyhrf.tools.misc), 173
calcDt() (pyhrf.jde.jde_multi_sess.BOLDSampler_Multi_SessInput method), 92
calcDt() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSujInput method), 100
calcDt() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_MultiSujInput method), 108
calcDt() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
calcDt() (pyhrf.jde.models.BOLDSamplerInput method), 117
calcEnergy() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
calcFracLambdaTilde() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
calcFracLambdaTildeWithIRRelCond() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
calcFracLambdaTildeWithRelCond() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
calculate_uncertainty() (in module pyhrf.sandbox.parcellation), 131
calcXh() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
calcXh() (pyhrf.jde.hrf.HRFSampler method), 88
calcXh() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
calcXh() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
calcXh() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
calcXResp() (pyhrf.jde.asl.ResponseSampler method), 39
calcXResp() (pyhrf.jde.asl_physio.ResponseSampler method), 45
calcXResp() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 51
calcXResp() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
calcXResp() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
calcXResp() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
calcXResp() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 79
call_if_func() (pyhrf.tools.aexpression.ArithmeticExpression method), 169
callback() (pyhrf.jde.models.CallbackCritDiff method), 118
callback() (pyhrf.jde.samplerbase.GSDefaultCallbackHandler method), 124
callback() (pyhrf.jde.samplerbase.GSPrintCallbackHandler method), 124
CallbackCritDiff (class in pyhrf.jde.models), 118
cartesian() (in module pyhrf.tools.misc), 173
cartesian_apply() (in module pyhrf.tools.misc), 174
cartesian_combine_args() (in module pyhrf.tools.misc), 174
cartesian_eval() (in module pyhrf.tools.misc), 175
cartesian_params() (in module pyhrf.tools.misc), 175
cartesian_test() (in module pyhrf.tools.misc), 175
CartesianTest (class in pyhrf.test.toolsTest), 167
cellDOMWriter() (pyhrf.xmliobak.xmlmatlab.MatlabXMLHandler static method), 226
cellTagDOMReader() (pyhrf.xmliobak.xmlmatlab.MatlabXMLHandler static method), 227
center_mask_at() (in module pyhrf.graph), 234
center_mask_at_v01() (in module pyhrf.graph), 234
center_mask_at_v02() (in module pyhrf.graph), 234
cexpand() (pyhrf.ndarray.xndarray method), 241
cfg_error_report() (in module pyhrf.configuration), 229
cflatten() (pyhrf.ndarray.xndarray method), 241
change_dim() (in module pyhrf.ui.vb_jde_analyser), 186
change_dim() (in module pyhrf.ui.vb_jde_analyser_asl_fast), 187
change_dim() (in module pyhrf.ui.vb_jde_analyser_bold_fast), 189
check() (pyhrf.tools.aexpression.ArithmeticExpression method), 169
check_against_truth() (pyhrf.sandbox.stats.GibbsSampler

method), 144
check_against_truth() (pyhrf.sandbox.stats.GSVariable method), 143
check_files_series() (in module pyhrf.tools.misc), 175
check_final_value() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
check_init_func() (pyhrf.xmliobak.xmlbase.XMLable2 method), 226
check_init_obj() (pyhrf.xmlio.Initable method), 258
check_initialization_arg() (pyhrf.sandbox.stats.GSVariable method), 143
check_stim_durations() (in module pyhrf.paradigm), 247
check_subdirs() (in module pyhrf.sandbox.data_parser), 128
checkAndSetInitHabit() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 32
checkAndSetInitLabels() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
checkAndSetInitNRL() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
checkAndSetInitValue() (pyhrf.jde.asl.DriftCoeffSampler method), 36
checkAndSetInitValue() (pyhrf.jde.asl.DriftVarianceSampler method), 36
checkAndSetInitValue() (pyhrf.jde.asl.LabelSampler method), 37
checkAndSetInitValue() (pyhrf.jde.asl.MixtureParamsSampler method), 37
checkAndSetInitValue() (pyhrf.jde.asl.NoiseVarianceSampler method), 37
checkAndSetInitValue() (pyhrf.jde.asl.PerfBaselineSampler method), 38
checkAndSetInitValue() (pyhrf.jde.asl.PerfBaselineVarianceSampler method), 38
checkAndSetInitValue() (pyhrf.jde.asl.PerfMixtureSampler method), 38
checkAndSetInitValue() (pyhrf.jde.asl.PerfResponseLevelSampler method), 38
checkAndSetInitValue() (pyhrf.jde.asl.ResponseLevelSampler method), 38
checkAndSetInitValue() (pyhrf.jde.asl.ResponseSampler method), 39
checkAndSetInitValue() (pyhrf.jde.asl.ResponseVarianceSampler method), 39
checkAndSetInitValue() (pyhrf.jde.asl_physio.DriftCoeffSampler method), 42
checkAndSetInitValue() (pyhrf.jde.asl_physio.DriftVarianceSampler method), 42
checkAndSetInitValue() (pyhrf.jde.asl_physio.LabelSampler method), 42
checkAndSetInitValue() (pyhrf.jde.asl_physio.MixtureParamsSampler method), 43
checkAndSetInitValue() (pyhrf.jde.asl_physio.NoiseVarianceSampler method), 43
checkAndSetInitValue() (pyhrf.jde.asl_physio.PerfBaselineSampler method), 43
checkAndSetInitValue() (pyhrf.jde.asl_physio.PerfBaselineVarianceSampler method), 44
checkAndSetInitValue() (pyhrf.jde.asl_physio.PerfMixtureSampler method), 44
checkAndSetInitValue() (pyhrf.jde.asl_physio.PerfResponseLevelSampler method), 44
checkAndSetInitValue() (pyhrf.jde.asl_physio.ResponseLevelSampler method), 45
checkAndSetInitValue() (pyhrf.jde.asl_physio.ResponseSampler method), 45
checkAndSetInitValue() (pyhrf.jde.asl_physio.ResponseVarianceSampler method), 46
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.DriftCoeffSampler method), 48
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.DriftVarianceSampler method), 48
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.LabelSampler method), 48
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.MixtureParamsSampler method), 49
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.NoiseVarianceSampler method), 49
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.PerfBaselineSampler method), 49
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.PerfBaselineVarianceS method), 49
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.PerfMixtureSampler method), 50
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.PerfResponseLevelSa method), 50
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.ResponseLevelSample method), 51
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 51
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step.ResponseVarianceSam method), 52
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.DriftCoeffSa method), 54
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.DriftVariance method), 54
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.LabelSampler method), 54
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.MixtureParam method), 55
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.NoiseVarianc method), 55
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.PerfBaseline method), 55
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.PerfBaseline method), 56

checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.PerfMixtureSampler method), 56
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.PerfResponseLevelSampler method), 56
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler method), 59
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
checkAndSetInitValue() (pyhrf.jde.asl_physio_1step_params.ResponseVarianceSampler method), 60
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.DriftCoeffSampler method), 62
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.DriftVarianceSampler method), 62
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.LabelSampler method), 62
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler method), 63
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.NoiseVarianceSampler method), 63
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.PerfBaselineSampler method), 63
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.PerfBaselineVarianceSampler method), 63
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.PerfMixtureSampler method), 64
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.PerfResponseLevelSampler method), 64
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.ResponseLevelSampler method), 65
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
checkAndSetInitValue() (pyhrf.jde.asl_physio_det_fwdm.ResponseVarianceSampler method), 66
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.DriftCoeffSampler method), 69
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.DriftVarianceSampler method), 69
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.LabelSampler method), 69
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler method), 70
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.NoiseVarianceSampler method), 70
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.PerfBaselineSampler method), 70
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.PerfBaselineVarianceSampler method), 71
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.PerfMixtureSampler method), 71
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.PerfResponseLevelSampler method), 71
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseVarianceSampler method), 72
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseSampler method), 72
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.PhysioTrueBOLDResponseSampler method), 73
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.ResponseLevelSampler method), 73
checkAndSetInitValue() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.DriftCoeffSampler method), 76
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.DriftVarianceSampler method), 76
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.LabelSampler method), 76
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.MixtureParamsSampler method), 77
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.NoiseVarianceSampler method), 77
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.PerfBaselineSampler method), 77
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.PerfBaselineVarianceSampler method), 78
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.PerfMixtureSampler method), 78
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.PerfResponseLevelSampler method), 78
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.PhysioJointResponseVarianceSampler method), 78
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.ResponseLevelSampler method), 79
checkAndSetInitValue() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 79
checkAndSetInitValue() (pyhrf.jde.beta.BetaSampler method), 80
checkAndSetInitValue() (pyhrf.jde.drift.DriftARSampler method), 86
checkAndSetInitValue() (pyhrf.jde.drift.DriftSampler method), 86
checkAndSetInitValue() (pyhrf.jde.drift.DriftSamplerWithRelVar method), 87
checkAndSetInitValue() (pyhrf.jde.drift.ETASampler method), 87
checkAndSetInitValue() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
checkAndSetInitValue() (pyhrf.jde.hrf.HRFSampler method), 88
checkAndSetInitValues() (pyhrf.jde.hrf.RHSampler method), 90
checkAndSetInitValue() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Sampler method), 93
checkAndSetInitValue() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93
checkAndSetInitValue() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94

checkAndSetInitValue() (pyhrf.jde.jde_multi_sess.NoiseVarianceSampler_Multi_Sess method), 96
checkAndSetInitValue() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 94
checkAndSetInitValue() (pyhrf.jde.jde_multi_sess.NRLsBar_DriftMultiSess_Sampler method), 95
checkAndSetInitValue() (pyhrf.jde.jde_multi_sess.Variance_GaussianNRL_Multi_Sess method), 96
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.ETASampler_MultiSubj method), 100
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 101
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.HRFVariance_Sampler method), 101
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.LabelSampler method), 103
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler method), 103
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.NoiseVariance_DriftMultiSubj_Sampler method), 104
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.NRLs_SampleAllSubjects method), 104
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.RHGroupSampler method), 104
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets.Variance_GaussianNRL_MultiSubj method), 104
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.AlphaSampler_MultiSubj method), 106
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.AlphaVarianceSampler method), 106
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler method), 108
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler method), 109
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.ETASampler_MultiSubj method), 109
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.HRFVariance_Sampler method), 110
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler method), 112
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler method), 112
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.NoiseVariance_DriftMultiSubj_Sampler method), 113
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.NRLs_SampleAllSubjects method), 113
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.RHGroupSampler method), 113
checkAndSetInitValue() (pyhrf.jde.jde_multi_sujets_alpha.Variance_GaussianNRL_MultiSubj method), 113
checkAndSetInitValue() (pyhrf.jde.noise.NoiseARParamsSampler method), 122
checkAndSetInitValue() (pyhrf.jde.noise.NoiseVariance_Drift_Sampler method), 123
checkAndSetInitValue() (pyhrf.jde.noise.NoiseVarianceARSampler method), 123
checkAndSetInitValue() (pyhrf.jde.noise.NoiseVarianceSampler method), 123
checkAndSetInitValue() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
checkAndSetInitValue() (pyhrf.jde.nrl.bigaussian.MixtureWeightsSampler method), 20
checkAndSetInitValue() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
checkAndSetInitValue() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
checkAndSetInitValue() (pyhrf.jde.nrl.bigaussian.Variance_GaussianNRL_Drift method), 24
checkAndSetInitValue() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParamsSampler method), 25
checkAndSetInitValue() (pyhrf.jde.nrl.bigaussian_drift.NRLsBar_Drift_MultiSess_Sampler method), 29
checkAndSetInitValue() (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler method), 31
checkAndSetInitValue() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRL_Sampler method), 32
checkAndSetInitValue() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 32
checkAndSetInitValue() (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler method), 35
checkAndSetInitValue() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
checkAndSetInitValue() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
checkAndSetInitValue() (pyhrf.jde.wsampler.WSampler method), 126
checkGraph() (pyhrf.tools.misc.Pipeline method), 172
chewUpOnsets() (pyhrf.jde.jde_multi_sess.BOLDSampler_Multi_SessInput method), 92
chewUpOnsets() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSujInput method), 80
chewUpOnsets() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_MultiSujInput method), 108
chewUpOnsets() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
chewUpOnsets() (pyhrf.jde.models.BOLDSamplerInput method), 117
child() (pyhrf.xmlio.UiNode method), 259
child() (subclass in pyhrf.test.test_xml), 166
childCount() (pyhrf.xmlio.UiNode method), 259

children_ (pyhrf.sandbox.parcellation.Ward attribute), 130
children_ (pyhrf.sandbox.parcellation.WardAgglomeration attribute), 131
chooseSampleNext() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
CLASS_NAMES (pyhrf.jde.asl.LabelSampler attribute), 37
CLASS_NAMES (pyhrf.jde.asl_physio.LabelSampler attribute), 42
CLASS_NAMES (pyhrf.jde.asl_physio_1step.LabelSampler attribute), 48
CLASS_NAMES (pyhrf.jde.asl_physio_1step_params.LabelSampler attribute), 54
CLASS_NAMES (pyhrf.jde.asl_physio_det_fwdm.LabelSampler attribute), 62
CLASS_NAMES (pyhrf.jde.asl_physio_hierarchical.LabelSampler attribute), 69
CLASS_NAMES (pyhrf.jde.asl_physio_joint.LabelSampler attribute), 76
CLASS_NAMES (pyhrf.jde.jde_multi_sujets.LabelSampler attribute), 103
CLASS_NAMES (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler attribute), 112
CLASS_NAMES (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler attribute), 112
CLASS_NAMES (pyhrf.jde.nrl.bigaussian.NRLSampler attribute), 21
CLASS_NAMES (pyhrf.jde.nrl.trigaussian.GGGNRLSampler attribute), 34
CLASS_NAMES (pyhrf.jde.wsampler.W_Drift_Sampler attribute), 127
CLASS_NAMES (pyhrf.jde.wsampler.WSampler attribute), 126
CLASSES (pyhrf.jde.asl.LabelSampler attribute), 37
CLASSES (pyhrf.jde.asl_physio.LabelSampler attribute), 42
CLASSES (pyhrf.jde.asl_physio_1step.LabelSampler attribute), 48
CLASSES (pyhrf.jde.asl_physio_1step_params.LabelSampler attribute), 54
CLASSES (pyhrf.jde.asl_physio_det_fwdm.LabelSampler attribute), 62
CLASSES (pyhrf.jde.asl_physio_hierarchical.LabelSampler attribute), 69
CLASSES (pyhrf.jde.asl_physio_joint.LabelSampler attribute), 76
CLASSES (pyhrf.jde.jde_multi_sujets.LabelSampler attribute), 103
CLASSES (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler attribute), 112
CLASSES (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler attribute), 112
CLASSES (pyhrf.jde.nrl.bigaussian.NRLSampler attribute), 21
CLASSES (pyhrf.jde.nrl.trigaussian.GGGNRLSampler attribute), 34
CLASSES (pyhrf.jde.wsampler.W_Drift_Sampler attribute), 127
CLASSES (pyhrf.jde.wsampler.WSampler attribute), 126
clean_cache() (in module pyhrf.validation.config), 190
clean_memory() (pyhrf.rfir.RFIREstim method), 256
clean_output_files() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
clean_output_files() (pyhrf.ui.treatment.FMRITreatment method), 183
cleanMem() (pyhrf.jde.jde_multi_sess.BOLDSampler_Multi_SessInput method), 92
cleanMem() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSujInput method), 100
cleanMem() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_MultiSujInp method), 108
cleanMem() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
cleanMem() (pyhrf.jde.models.BOLDSamplerInput method), 118
cleanMemory() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 94
cleanMemory() (pyhrf.jde.nrl.ar.NRLARSampler method), 18
cleanMemory() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
cleanMemory() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
cleanMemory() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 32
cleanObservables() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSam method), 91
cleanObservables() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSa method), 99
cleanObservables() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_S method), 107
cleanObservables() (pyhrf.jde.models.BOLDGibbsSampler method), 116
cleanObservables() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
cleanObservables() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
cleanObservables() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 32
cleanObservables() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
cleanPrecalculations() (pyhrf.jde.asl.WN_BiG_ASLSamplerInput method), 39
cleanPrecalculations() (pyhrf.jde.asl_physio.WN_BiG_ASLSamplerInput method), 46
cleanPrecalculations() (pyhrf.jde.asl_physio_1step.WN_BiG_ASLSamplerI method), 52

cleanPrecalculations() (pyhrf.jde.asl_physio_1step_params.WN_BiG_ASLSampler method), 60
cleanPrecalculations() (pyhrf.jde.asl_physio_det_fwdm.WN_BiG_ASLSampler method), 66
cleanPrecalculations() (pyhrf.jde.asl_physio_hierarchical.WN_BiG_ASLSampler method), 74
cleanPrecalculations() (pyhrf.jde.asl_physio_joint.WN_BiG_ASLSampler method), 80
cleanPrecalculations() (pyhrf.jde.jde_multi_sess.BOLDSampler_MultiSess method), 92
cleanPrecalculations() (pyhrf.jde.models.ARN_BiG_BOLDSamplerInput method), 115
cleanPrecalculations() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
cleanPrecalculations() (pyhrf.jde.models.BOLDSamplerInput method), 118
cleanPrecalculations() (pyhrf.jde.models.Hab_WN_BiG_BOLDSamplerInput method), 119
cleanPrecalculations() (pyhrf.jde.models.WN_BiG_BOLDSamplerInput method), 119
cleanPrecalculations() (pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerInput method), 119
clear() (pyhrf.tools.backports.OrderedDict method), 170
closestsorted() (in module pyhrf.tools.misc), 175
ClusterMixin (class in pyhrf.sandbox.parcellation), 129
CmdInputTest (class in pyhrf.test.test_treatment), 165
CmdParcellationTest (class in pyhrf.test.test_parcellation), 163
COLORS (pyhrf.tools.misc.AnsiColorizer attribute), 172
compute_aawwXhQXhi() (pyhrf.jde.noise.NoiseVarianceSamplerWithRelVar method), 123
compute_aaXhQXhi() (pyhrf.jde.noise.NoiseVarianceSampler method), 123
compute_average() (pyhrf.core.FmriData method), 231
compute_bRpR() (in module pyhrf.jde.asl_physio), 46
compute_bRpR() (in module pyhrf.jde.asl_physio_1step), 52
compute_bRpR() (in module pyhrf.jde.asl_physio_1step_params), 60
compute_bRpR() (in module pyhrf.jde.asl_physio_det_fwdm), 67
compute_bRpR() (in module pyhrf.jde.asl_physio_hierarchical), 75
compute_bRpR() (in module pyhrf.jde.asl_physio_joint), 80
compute_consensus_clusters_parallel() (in module pyhrf.sandbox.func_BMA_consensus_clustering), 128
compute_contrasts() (in module pyhrf.vbjde.vem_tools), 209
compute_crit_diff() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_Sess method), 91
compute_crit_diff() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_Subj method), 99
compute_crit_diff() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_Subj method), 107
compute_crit_diff() (pyhrf.jde.models.BOLDGibbsSampler method), 116
compute_crit_diff() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
compute_ext_field() (pyhrf.jde.asl.LabelSampler method), 37
compute_ext_field() (pyhrf.jde.asl_physio.LabelSampler method), 42
compute_ext_field() (pyhrf.jde.asl_physio_1step.LabelSampler method), 48
compute_ext_field() (pyhrf.jde.asl_physio_1step_params.LabelSampler method), 54
compute_ext_field() (pyhrf.jde.asl_physio_det_fwdm.LabelSampler method), 62
compute_ext_field() (pyhrf.jde.asl_physio_hierarchical.LabelSampler method), 69
compute_ext_field() (pyhrf.jde.asl_physio_joint.LabelSampler method), 76
compute_ext_field() (pyhrf.jde.jde_multi_sujets.LabelSampler method), 103
compute_ext_field() (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler method), 112
compute_fit() (pyhrf.rfir.RFIREstim method), 256
Compute_FreeEnergy() (in module pyhrf.vbjde.vem_tools), 207
compute_fwhm() (in module pyhrf.sandbox.parcellation), 131
compute_hrf() (in module pyhrf.sandbox.parcellation), 131
Compute_INV_R_and_R_and_DET_R() (pyhrf.rfir.RFIREstim method), 254
compute_mat_X_2() (in module pyhrf.vbjde.vem_tools), 209
compute_mixt_dist() (in module pyhrf.sandbox.parcellation), 131
compute_mixt_dist_skgmm() (in module pyhrf.sandbox.parcellation), 132
Compute_onset_matrix3() (pyhrf.rfir.RFIREstim method), 254
compute_residuals() (pyhrf.jde.asl.PerfBaselineSampler method), 38
compute_residuals() (pyhrf.jde.asl_physio.PerfBaselineSampler method), 43
compute_residuals() (pyhrf.jde.asl_physio_1step.PerfBaselineSampler method), 49
compute_residuals() (pyhrf.jde.asl_physio_1step_params.PerfBaselineSampler method), 55
compute_residuals() (pyhrf.jde.asl_physio_det_fwdm.PerfBaselineSampler method), 63
compute_residuals() (pyhrf.jde.asl_physio_hierarchical.PerfBaselineSampler method), 70

compute_residuals() (pyhrf.jde.asl_physio_joint.PerfBaselineSampler method), 77
compute_roc_labels() (in module pyhrf.stats.misc), 145
compute_roc_labels_scikit() (in module pyhrf.stats.misc), 145
compute_StS_StY() (in module pyhrf.jde.asl), 39
compute_StS_StY() (in module pyhrf.jde.asl_physio), 46
compute_StS_StY() (in module pyhrf.jde.asl_physio_1step), 52
compute_StS_StY() (in module pyhrf.jde.asl_physio_1step_params), 60
compute_StS_StY() (in module pyhrf.jde.asl_physio_det_fwdm), 67
compute_StS_StY() (in module pyhrf.jde.asl_physio_hierarchical), 74
compute_StS_StY() (in module pyhrf.jde.asl_physio_joint), 80
compute_StS_StY_deterministic() (in module pyhrf.jde.asl_physio), 46
compute_StS_StY_deterministic() (in module pyhrf.jde.asl_physio_1step), 52
compute_StS_StY_deterministic() (in module pyhrf.jde.asl_physio_1step_params), 60
compute_StS_StY_deterministic() (in module pyhrf.jde.asl_physio_det_fwdm), 67
compute_StS_StY_deterministic() (in module pyhrf.jde.asl_physio_hierarchical), 74
compute_summary_stats() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
compute_T_Pvalue() (in module pyhrf.stats.misc), 145
compute_uward_dist() (in module pyhrf.sandbox.parcellation), 132
compute_uward_dist2() (in module pyhrf.sandbox.parcellation), 132
compute_wa() (pyhrf.jde.asl.PerfBaselineSampler method), 38
compute_wa() (pyhrf.jde.asl_physio.PerfBaselineSampler method), 43
compute_wa() (pyhrf.jde.asl_physio_1step.PerfBaselineSampler method), 49
compute_wa() (pyhrf.jde.asl_physio_1step_params.PerfBaselineSampler method), 55
compute_wa() (pyhrf.jde.asl_physio_det_fwdm.PerfBaselineSampler method), 63
compute_wa() (pyhrf.jde.asl_physio_hierarchical.PerfBaselineSampler method), 70
compute_wa() (pyhrf.jde.asl_physio_joint.PerfBaselineSampler method), 77
compute_y_tilde() (pyhrf.jde.asl.DriftCoeffSampler method), 36
compute_y_tilde() (pyhrf.jde.asl.NoiseVarianceSampler method), 37
compute_y_tilde() (pyhrf.jde.asl_physio.DriftCoeffSampler method), 42
compute_y_tilde() (pyhrf.jde.asl_physio.NoiseVarianceSampler method), 43
compute_y_tilde() (pyhrf.jde.asl_physio_1step.DriftCoeffSampler method), 48
compute_y_tilde() (pyhrf.jde.asl_physio_1step.NoiseVarianceSampler method), 49
compute_y_tilde() (pyhrf.jde.asl_physio_1step_params.DriftCoeffSampler method), 54
compute_y_tilde() (pyhrf.jde.asl_physio_1step_params.NoiseVarianceSamp method), 55
compute_y_tilde() (pyhrf.jde.asl_physio_det_fwdm.DriftCoeffSampler method), 62
compute_y_tilde() (pyhrf.jde.asl_physio_det_fwdm.NoiseVarianceSampler method), 63
compute_y_tilde() (pyhrf.jde.asl_physio_hierarchical.DriftCoeffSampler method), 69
compute_y_tilde() (pyhrf.jde.asl_physio_hierarchical.NoiseVarianceSample method), 70
compute_y_tilde() (pyhrf.jde.asl_physio_joint.DriftCoeffSampler method), 76
compute_y_tilde() (pyhrf.jde.asl_physio_joint.NoiseVarianceSampler method), 77
computeAA() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 94
computeAA() (pyhrf.jde.jde_multi_sujets.NRLs_Sampler method), 104
computeAA() (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler method), 113
computeAA() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
computeAA() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
computeB() (in module pyhrf.test.toolsTest), 168
computeC() (in module pyhrf.test.toolsTest), 168
computeComponentsApost() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
computeComponentsApost() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
computeComponentsApost() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
computeComponentsApost() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 32
computeComponentsApostWithRelVar() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
computeContrasts() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
computeD() (in module pyhrf.test.toolsTest), 168
computeF() (in module pyhrf.test.toolsTest), 168

computeFit() (in module pyhrf.vbjde.vem_tools), 209
computeFit() (pyhrf.jde.asl.ASLSampler method), 35
computeFit() (pyhrf.jde.asl_physio.ASLPhysioSampler method), 41
computeFit() (pyhrf.jde.asl_physio_1step.ASLPhysioSampler method), 47
computeFit() (pyhrf.jde.asl_physio_1step_params.ASLPhysioSampler method), 53
computeFit() (pyhrf.jde.asl_physio_det_fwdm.ASLPhysioSampler method), 61
computeFit() (pyhrf.jde.asl_physio_hierarchical.ASLPhysioSampler method), 68
computeFit() (pyhrf.jde.asl_physio_joint.ASLPhysioSampler method), 75
computeFit() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSampler method), 91
computeFit() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSampler method), 99
computeFit() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_SubjSampler method), 107
computeFit() (pyhrf.jde.models.BOLDGibbsSampler method), 116
computeFit() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
computeFit() (pyhrf.jde.models.Drift_BOLDGibbsSampler method), 118
computeFit() (pyhrf.jde.samplerbase.GibbsSampler method), 124
computeFit_asl() (in module pyhrf.vbjde.vem_tools), 209
computehXQXh() (in module pyhrf.jde.models), 120
computeInvAutoCorrNoise() (pyhrf.jde.noise.NoiseARParamsSampler method), 122
computeJ() (in module pyhrf.test.toolsTest), 168
computeK() (in module pyhrf.test.toolsTest), 168
computeL() (in module pyhrf.test.toolsTest), 168
computeMean() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
computeMeanClassApost() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
computeMeanVarClassApost() (pyhrf.jde.nrl.ar.NRLARSampler method), 18
computemoyq() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
computemoyq() (pyhrf.jde.wsampler.WSampler method), 126
computemoyqvox() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
computeMXhQXh() (pyhrf.jde.noise.NoiseVarianceSampler method), 123
computePl() (in module pyhrf.jde.models), 120
computePMStimInducedSignal() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSampler method), 91
computePMStimInducedSignal() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSampler method), 99
computePMStimInducedSignal() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_SubjSampler method), 107
computePMStimInducedSignal() (pyhrf.jde.models.BOLDGibbsSampler method), 116
computePMStimInducedSignal() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
computeW1() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
computeW1() (pyhrf.jde.wsampler.WSampler method), 126
computeWRR() (pyhrf.jde.asl.ResponseLevelSampler method), 38
computeRR() (pyhrf.jde.asl_physio.ResponseLevelSampler method), 45
computeRR() (pyhrf.jde.asl_physio_1step.ResponseLevelSampler method), 51
computeRR() (pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler method), 59
computeRR() (pyhrf.jde.asl_physio_det_fwdm.ResponseLevelSampler method), 65
computeRR() (pyhrf.jde.asl_physio_hierarchical.ResponseLevelSampler method), 73
computeRR() (pyhrf.jde.asl_physio_joint.ResponseLevelSampler method), 79
computeStDS_StDY() (pyhrf.jde.hrf.HRF_Drift_Sampler method), 89
computeStDS_StDY() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
computeStDS_StDY() (pyhrf.jde.hrf.HRFARSampler method), 87
computeStDS_StDY() (pyhrf.jde.hrf.HRFSampler method), 88
computeStDS_StDY() (pyhrf.jde.hrf.HRFwithHabSampler method), 90
computeStDS_StDY() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
computeStDS_StDY() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
computeStDS_StDY() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
computeStDS_StDY_from_HRFSampler() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
computeStDS_StDY_one_session() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94

computeStDS_StDY_one_subject() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
computeStDS_StDY_one_subject() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
computeStDS_StDY_WithRelVar() (pyhrf.jde.hrf.HRF_Drift_SamplerWithRelVar method), 89
computeStDS_StDY_WithRelVar() (pyhrf.jde.hrf.HRFSamplerWithRelVar method), 89
computeSumjaXh() (in module pyhrf.jde.models), 120
computeSumWAxh() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
computeVariablesApost() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
computeVarXhtQ() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
computeVarXhtQ() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 32
computeVarXhtQ() (pyhrf.jde.wsampler.WSampler method), 126
computeVarYTilde() (pyhrf.jde.drift.DriftARSampler method), 86
computeVarYTilde() (pyhrf.jde.noise.NoiseVarianceARSampler method), 123
computeVarYTilde() (pyhrf.jde.nrl.ar.NRLARSampler method), 18
computeVarYTilde() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
computeVarYTildeHab() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 32
computeVarYTildeHabOld() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
computeVarYTildeOpt() (pyhrf.jde.asl.BOLDResponseLevelSampler method), 36
computeVarYTildeOpt() (pyhrf.jde.asl.PerfResponseLevelSampler method), 38
computeVarYTildeOpt() (pyhrf.jde.asl.ResponseLevelSampler method), 38
computeVarYTildeOpt() (pyhrf.jde.asl_physio.BOLDResponseLevelSampler method), 42
computeVarYTildeOpt() (pyhrf.jde.asl_physio.PerfResponseLevelSampler method), 44
computeVarYTildeOpt() (pyhrf.jde.asl_physio.ResponseLevelSampler method), 45
computeVarYTildeOpt() (pyhrf.jde.asl_physio_1step.BOLDResponseLevelSampler method), 48
computeVarYTildeOpt() (pyhrf.jde.asl_physio_1step.PerfResponseLevelSampler method), 50
computeVarYTildeOpt() (pyhrf.jde.asl_physio_1step.ResponseLevelSampler method), 51
computeVarYTildeOpt() (pyhrf.jde.asl_physio_1step_params.BOLDResponseLevelSampler method), 54
computeVarYTildeOpt() (pyhrf.jde.asl_physio_1step_params.PerfResponseLevelSampler method), 56
computeVarYTildeOpt() (pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler method), 59
computeVarYTildeOpt() (pyhrf.jde.asl_physio_det_fwdm.BOLDResponseLevelSampler method), 62
computeVarYTildeOpt() (pyhrf.jde.asl_physio_det_fwdm.PerfResponseLevelSampler method), 64
computeVarYTildeOpt() (pyhrf.jde.asl_physio_det_fwdm.ResponseLevelSampler method), 65
computeVarYTildeOpt() (pyhrf.jde.asl_physio_hierarchical.BOLDResponseLevelSampler method), 69
computeVarYTildeOpt() (pyhrf.jde.asl_physio_hierarchical.PerfResponseLevelSampler method), 71
computeVarYTildeOpt() (pyhrf.jde.asl_physio_hierarchical.ResponseLevelSampler method), 73
computeVarYTildeOpt() (pyhrf.jde.asl_physio_joint.BOLDResponseLevelSampler method), 76
computeVarYTildeOpt() (pyhrf.jde.asl_physio_joint.PerfResponseLevelSampler method), 78
computeVarYTildeOpt() (pyhrf.jde.asl_physio_joint.ResponseLevelSampler method), 79
computeVarYTildeOpt() (pyhrf.jde.jde_multi_sujets.NRLs_Sampler method), 104
computeVarYTildeOpt() (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler method), 113
computeVarYTildeOpt() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
computeVarYTildeOpt() (pyhrf.jde.nrl.bigaussian_drift.NRL_Drift_Sampler method), 26
computeVarYTildeOptWithRelVar() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
computeVarYTildeOptWithRelVar() (pyhrf.jde.nrl.bigaussian_drift.NRL_Drift_SamplerWithRelVar method), 27
computeVarYTildeSessionOpt() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
computeVarYTildeSessionOpt() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
computeVarYTildeWithRelVar() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
computeWithJeffreyPriors() (pyhrf.jde.asl.MixtureParamsSampler method), 37
computeWithJeffreyPriors() (pyhrf.jde.asl_physio.MixtureParamsSampler method), 43

computeWithJeffreyPriors() (pyhrf.jde.asl_physio_1step.MixtureParamsSampler method), 49
computeWithJeffreyPriors() (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler method), 55
computeWithJeffreyPriors() (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler method), 63
computeWithJeffreyPriors() (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler method), 70
computeWithJeffreyPriors() (pyhrf.jde.asl_physio_joint.MixtureParamsSampler method), 77
computeWithJeffreyPriors() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler method), 93
computeWithJeffreyPriors() (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler method), 103
computeWithJeffreyPriors() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler method), 108
computeWithJeffreyPriors() (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler method), 112
computeWithJeffreyPriors() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
computeWithJeffreyPriors() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler method), 25
computeWithJeffreyPriors() (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler method), 35
computeWithProperPriors() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler method), 93
computeWithProperPriors() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler method), 108
computeWithProperPriors() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
computeWithProperPriors() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler method), 25
computeWithProperPriorsWithRelVar() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 20
computeWithProperPriorsWithRelVar() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 20
computeWW() (pyhrf.jde.noise.NoiseVarianceSamplerWithRelVar method), 123
computeXh() (in module pyhrf.jde.models), 120
computeYBar() (in module pyhrf.jde.models), 120
computeYTilde() (in module pyhrf.jde.models), 120
computeYTilde() (pyhrf.jde.asl.BOLDResponseSampler method), 36
computeYTilde() (pyhrf.jde.asl.PerfResponseSampler method), 38
computeYTilde() (pyhrf.jde.asl.ResponseSampler method), 39
computeYTilde() (pyhrf.jde.asl_physio.PhysioBOLDResponseSampler method), 44
computeYTilde() (pyhrf.jde.asl_physio.PhysioPerfResponseSampler method), 45
computeYTilde() (pyhrf.jde.asl_physio.ResponseSampler method), 45
computeYTilde() (pyhrf.jde.asl_physio_1step.PhysioBOLDResponseSampler method), 50
computeYTilde() (pyhrf.jde.asl_physio_1step.PhysioPerfResponseSampler method), 50
computeYTilde() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 51
computeYTilde() (pyhrf.jde.asl_physio_1step_params.PhysioBOLDResponseSampler method), 57
computeYTilde() (pyhrf.jde.asl_physio_1step_params.PhysioPerfResponseSampler method), 58
computeYTilde() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
computeYTilde() (pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResponseSampler method), 64
computeYTilde() (pyhrf.jde.asl_physio_det_fwdm.PhysioPerfResponseSampler method), 65
computeYTilde() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
computeYTilde() (pyhrf.jde.asl_physio_hierarchical.PhysioBOLDResponseSampler method), 71
computeYTilde() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseSampler method), 72
computeYTilde() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
computeYTilde() (pyhrf.jde.asl_physio_joint.PhysioBOLDResponseSampler method), 78
computeYTilde() (pyhrf.jde.asl_physio_joint.PhysioPerfResponseSampler method), 79
computeYTilde() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 81
computeYTilde_Pl() (in module pyhrf.jde.models), 120
condense_series() (in module pyhrf.tools.misc), 175
Condition (class in pyhrf.core), 229
ConfigurationError, 229
connected_components() (in module pyhrf.graph), 234
connected_components_dict() (in module pyhrf.graph), 234

constraint_norm1_b() (in module pyhrf.vbjde.vem_tools), 209
contrasts_mean_var_classes() (in module pyhrf.vbjde.vem_tools), 209
contrasts_to_spm_vec() (in module pyhrf.paradigm), 247
convex_hull_mask() (in module pyhrf.tools.misc), 175
copy() (pyhrf.ndarray.xndarray method), 241
copy() (pyhrf.tools.backports.OrderedDict method), 170
cosine_drifts_basis() (in module pyhrf.vbjde.vem_tools), 210
count_homo_cliques() (in module pyhrf.boldsynth.field), 7
countLabels() (pyhrf.jde.asl.LabelSampler method), 37
countLabels() (pyhrf.jde.asl_physio.LabelSampler method), 43
countLabels() (pyhrf.jde.asl_physio_1step.LabelSampler method), 48
countLabels() (pyhrf.jde.asl_physio_1step_params.LabelSampler method), 54
countLabels() (pyhrf.jde.asl_physio_det_fwdm.LabelSampler method), 62
countLabels() (pyhrf.jde.asl_physio_hierarchical.LabelSampler method), 70
countLabels() (pyhrf.jde.asl_physio_joint.LabelSampler method), 77
countLabels() (pyhrf.jde.jde_multi_sujets.LabelSampler method), 103
countLabels() (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler method), 112
countLabels() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
countLabels() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
covariance_matrix() (in module pyhrf.vbjde.vem_tools), 210
Cpt_AcceptNewBeta_Graph() (in module pyhrf.jde.beta), 80
Cpt_Distrib_P_beta_graph() (in module pyhrf.jde.beta), 81
Cpt_Exact_lnZ_graph() (in module pyhrf.jde.beta), 81
Cpt_Expected_U_graph() (in module pyhrf.jde.beta), 82
cpt_ppm_a_apost() (in module pyhrf.stats.misc), 145
cpt_ppm_a_mcmc() (in module pyhrf.stats.misc), 145
cpt_ppm_a_norm() (in module pyhrf.stats.misc), 145
cpt_ppm_g_apost() (in module pyhrf.stats.misc), 146
cpt_ppm_g_mcmc() (in module pyhrf.stats.misc), 146
cpt_ppm_g_norm() (in module pyhrf.stats.misc), 146
Cpt_U_graph() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 4
Cpt_Vec_Estim_lnZ_Graph() (in module pyhrf.jde.beta), 82
Cpt_Vec_Estim_lnZ_Graph_fast() (in module pyhrf.jde.beta), 83
Cpt_Vec_Estim_lnZ_Graph_fast2() (in module pyhrf.jde.beta), 83
Cpt_Vec_Estim_lnZ_Graph_fast3() (in module pyhrf.jde.beta), 84
Cpt_Vec_Estim_lnZ_OLD_Graph() (in module pyhrf.jde.beta), 84
Cpt_Vec_Estim_lnZ_Onsager() (in module pyhrf.jde.beta), 85
Cpt_Vec_U_graph() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 4
cpt_XSigmaX() (pyhrf.rfir.RFIREstim method), 256
CptDefaultGraphLinks() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 3
CptDefaultGraphNodesLabels() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 3
CptDefaultGraphWeight() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 4
CptFctQ() (pyhrf.rfir.RFIREstim method), 255
CptRefGrphNgbhPosi() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 4
CptSigma() (pyhrf.rfir.RFIREstim method), 255
create_3Dlabels_Potts() (in module pyhrf.boldsynth.scenarios), 9
create_alpha_for_hrfgroup() (in module pyhrf.boldsynth.scenarios), 9
create_AR_noise() (in module pyhrf.boldsynth.scenarios), 9
create_asl_from_stim_induced() (in module pyhrf.boldsynth.scenarios), 9
create_asl_from_stim_induced() (in module pyhrf.sandbox.physio), 136
create_bigaussian_nrls() (in module pyhrf.boldsynth.scenarios), 9
create_bold() (in module pyhrf.boldsynth.scenarios), 9
create_bold_controlled_variance() (in module pyhrf.boldsynth.scenarios), 9
create_bold_from_hbr_and_cbv() (in module pyhrf.sandbox.physio), 136
create_bold_from_hbr_and_cbv() (in module pyhrf.sandbox.physio_params), 140
create_bold_from_stim_induced() (in module pyhrf.boldsynth.scenarios), 9
create_bold_from_stim_induced_RealNoise() (in module pyhrf.boldsynth.scenarios), 9
create_bold_stim_induced_signal() (in module pyhrf.boldsynth.scenarios), 10
create_canonical_hrf() (in module pyhrf.boldsynth.scenarios), 10
create_conditions() (in module pyhrf.vbjde.vem_tools),

210
create_connected_label_clusters() (in module pyhrf.boldsynth.scenarios), 10
create_drift_coeffs() (in module pyhrf.boldsynth.scenarios), 10
create_drift_coeffs_asl() (in module pyhrf.boldsynth.scenarios), 10
create_evoked_physio_signals() (in module pyhrf.sandbox.physio), 136
create_evoked_physio_signals() (in module pyhrf.sandbox.physio_params), 141
create_features() (in module pyhrf.validation.valid_sandbox_parcellation), 199
create_gaussian_hrf_subject() (in module pyhrf.boldsynth.scenarios), 10
create_gaussian_hrf_subject_and_group() (in module pyhrf.jde.jde_multi_sujets), 104
create_gaussian_noise() (in module pyhrf.boldsynth.scenarios), 10
create_gaussian_noise_asl() (in module pyhrf.boldsynth.scenarios), 10
create_gaussian_nrls_sessions_and_mean() (in module pyhrf.boldsynth.scenarios), 10
create_gsmooth_hrf() (in module pyhrf.boldsynth.scenarios), 10
create_hrf() (in module pyhrf.boldsynth.scenarios), 10
create_hrf_from_territories() (in module pyhrf.boldsynth.scenarios), 10
create_k_parameters() (in module pyhrf.sandbox.physio_params), 141
create_labels_Potts() (in module pyhrf.boldsynth.scenarios), 10
create_labels_vol() (in module pyhrf.boldsynth.scenarios), 10
create_language_paradigm() (in module pyhrf.boldsynth.scenarios), 10
create_localizer_paradigm() (in module pyhrf.boldsynth.scenarios), 11
create_localizer_paradigm_a() (in module pyhrf.boldsynth.scenarios), 11
create_localizer_paradigm_avd() (in module pyhrf.boldsynth.scenarios), 11
create_multisess_stim_induced_signal() (in module pyhrf.boldsynth.scenarios), 11
create_multisess_stim_induced_signal_asl() (in module pyhrf.boldsynth.scenarios), 11
create_neighbours() (in module pyhrf.vbjde.vem_tools), 211
create_null_drift() (in module pyhrf.boldsynth.scenarios), 11
create_omega_prf() (in module pyhrf.sandbox.physio), 137
create_omega_prf() (in module pyhrf.sandbox.physio_params), 141
create_options() (in module pyhrf.grid), 238
create_outliers() (in module pyhrf.boldsynth.scenarios), 11
create_paradigm_un_evnt() (in module pyhrf.boldsynth.scenarios), 11
create_perf_baseline() (in module pyhrf.boldsynth.scenarios), 11
create_perf_stim_induced_signal() (in module pyhrf.boldsynth.scenarios), 11
create_physio_brf() (in module pyhrf.sandbox.physio), 137
create_physio_brf() (in module pyhrf.sandbox.physio_params), 141
create_physio_prf() (in module pyhrf.sandbox.physio), 137
create_physio_prf() (in module pyhrf.sandbox.physio_params), 142
create_polynomial_drift() (in module pyhrf.boldsynth.scenarios), 11
create_polynomial_drift_from_coeffs() (in module pyhrf.boldsynth.scenarios), 11
create_polynomial_drift_from_coeffs_asl() (in module pyhrf.boldsynth.scenarios), 11
create_prf() (in module pyhrf.boldsynth.scenarios), 12
create_projection_kernels() (in module pyhrf.surface), 257
create_small_bold_simulation() (in module pyhrf.boldsynth.scenarios), 12
create_stim_induced_signal() (in module pyhrf.boldsynth.scenarios), 12
create_stim_induced_signal_Parsi() (in module pyhrf.boldsynth.scenarios), 12
create_t() (in module pyhrf.test.test_xml), 166
create_tbg_neural_efficacies() (in module pyhrf.sandbox.physio), 137
create_tbg_neural_efficacies() (in module pyhrf.sandbox.physio_params), 142
create_time_invariant_gaussian_brls() (in module pyhrf.boldsynth.scenarios), 12
create_time_invariant_gaussian_nrls() (in module pyhrf.boldsynth.scenarios), 12
create_time_invariant_gaussian_prls() (in module pyhrf.boldsynth.scenarios), 12
create_treatment() (in module pyhrf.ui.treatment), 183
create_treatment_surf() (in module pyhrf.ui.treatment), 183
create_unnormed_gaussian_hrf_subject() (in module pyhrf.jde.jde_multi_sujets), 105
create_varying_hrf() (in module pyhrf.boldsynth.scenarios), 12
create_Xh() (in module pyhrf.boldsynth.scenarios), 9
createBiGaussCovarNRL() (in module pyhrf.boldsynth.scenarios), 9

createDocument() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
createExpandedArray() (pyhrf.boldsynth.spatialconfig.MappedIn method), 14
createFromGUI() (pyhrf.boldsynth.spatialconfig.RegularLattice static method), 14
createFromGUI() (pyhrf.boldsynth.spatialconfig.UnboundSpatialM static method), 16
createWAxh() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
crop_array() (in module pyhrf.tools.misc), 175
CropTest (class in pyhrf.test.toolsTest), 167
cuboidPrinter() (in module pyhrf.tools.misc), 175
cumFreq() (in module pyhrf.stats.misc), 146

D

D (class in pyhrf.test.test_xml), 166
DataLoadTest (class in pyhrf.test.iotest), 153
default_nb_its (pyhrf.jde.asl.ASLSampler attribute), 35
default_nb_its (pyhrf.jde.asl_physio.ASLPhysioSampler attribute), 41
default_nb_its (pyhrf.jde.asl_physio_1step.ASLPhysioSampler attribute), 47
default_nb_its (pyhrf.jde.asl_physio_1step_params.ASLPhysioSampler attribute), 53
default_nb_its (pyhrf.jde.asl_physio_det_fwdm.ASLPhysioSampler attribute), 61
default_nb_its (pyhrf.jde.asl_physio_hierarchical.ASLPhysioSampler attribute), 68
default_nb_its (pyhrf.jde.asl_physio_joint.ASLPhysioSampler attribute), 75
default_nb_its (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSampler attribute), 91
default_nb_its (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSampler attribute), 99
default_nb_its (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_SubjSampler attribute), 107
default_nb_its (pyhrf.jde.models.BOLDGibbsSampler attribute), 116
default_nb_its (pyhrf.jde.models.BOLDGibbsSampler_AR attribute), 117
default_nb_its (pyhrf.jde.models.Drift_BOLDGibbsSampler attribute), 118
default_nb_its (pyhrf.jde.models.W_BOLDGibbsSampler attribute), 119
default_nb_its (pyhrf.jde.models.W_Drift_BOLDGibbsSampler attribute), 120
default_nb_its (pyhrf.rfir.RFIREstim attribute), 256
default_stop_crit1 (pyhrf.rfir.RFIREstim attribute), 256
default_stop_crit2 (pyhrf.rfir.RFIREstim attribute), 256
defaultParameters (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
defaultParameters (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler attribute), 109
defaultParameters (pyhrf.jde.noise.NoiseARParamsSampler attribute), 122
defaultParameters (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler attribute), 24
defaultParameters (pyhrf.jde.nrl.bigaussian.Variance_GaussianNRL_Multi_ attribute), 24
defaultParameters (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_M attribute), 25
defaultParameters (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParams attribute), 31
defaultParameters (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSamp attribute), 32
defaultParameters (pyhrf.xmliobak.xmlbase.XMLParamDrivenClass attribute), 225
delete_condition() (pyhrf.paradigm.Paradigm method), 246
deltaWCorr0() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
deltaWCorr1() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
DeprecatedXMLFormatException, 258
describeRois() (in module pyhrf.tools.misc), 175
descrip() (pyhrf.ndarray.xndarray method), 241
descrip_shape() (pyhrf.ndarray.xndarray method), 241
detectCyclity() (pyhrf.tools.misc.Pipeline method), 172
detectShortCircuit() (pyhrf.tools.misc.Pipeline method), 172
detectSignError() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
detectSignError() (pyhrf.jde.hrf.HRFSampler method), 88
diagBlock() (in module pyhrf.tools.misc), 175
DiagBlockTest (class in pyhrf.test.toolsTest), 167
dictDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
dictTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
DictToStringTest (class in pyhrf.test.toolsTest), 167
disable() (pyhrf.tools.misc.AnsiColorizer method), 172
discard_rois() (pyhrf.core.FmriData method), 231
discard_small_rois() (pyhrf.core.FmriData method), 231
DispatchedTasksManager (class in pyhrf.grid), 236
dist() (in module pyhrf.validation.valid_beta_estim), 190
distance() (in module pyhrf.tools.misc), 175
divide() (pyhrf.ndarray.xndarray method), 241
do_if_nonexistent_file() (in module pyhrf.tools.misc), 175
doubleDOMWriter() (pyhrf.xmliobak.xmlmatlab.MatlabXMLHandler static method), 227
doubleTagDOMReader() (pyhrf.xmliobak.xmlmatlab.MatlabXMLHandler static method), 227
DriftBOLDGibbsSampler (class in pyhrf.jde.models), 118

Drift_MultiSess_Sampler (class in pyhrf.jde.jde_multi_sess), 93
Drift_MultiSubj_Sampler (class in pyhrf.jde.jde_multi_sujets), 100
Drift_MultiSubj_Sampler (class in pyhrf.jde.jde_multi_sujets_alpha), 109
DriftARSampler (class in pyhrf.jde.drift), 86
DriftCoeffSampler (class in pyhrf.jde.asl), 36
DriftCoeffSampler (class in pyhrf.jde.asl_physio), 42
DriftCoeffSampler (class in pyhrf.jde.asl_physio_1step), 48
DriftCoeffSampler (class in pyhrf.jde.asl_physio_1step_params), 54
DriftCoeffSampler (class in pyhrf.jde.asl_physio_det_fwdm), 62
DriftCoeffSampler (class in pyhrf.jde.asl_physio_hierarchical), 69
DriftCoeffSampler (class in pyhrf.jde.asl_physio_joint), 76
drifts_coeffs_fit() (in module pyhrf.vbjde.vem_tools), 211
DriftSampler (class in pyhrf.jde.drift), 86
DriftSamplerWithRelVar (class in pyhrf.jde.drift), 86
DriftVarianceSampler (class in pyhrf.jde.asl), 36
DriftVarianceSampler (class in pyhrf.jde.asl_physio), 42
DriftVarianceSampler (class in pyhrf.jde.asl_physio_1step), 48
DriftVarianceSampler (class in pyhrf.jde.asl_physio_1step_params), 54
DriftVarianceSampler (class in pyhrf.jde.asl_physio_det_fwdm), 62
DriftVarianceSampler (class in pyhrf.jde.asl_physio_hierarchical), 69
DriftVarianceSampler (class in pyhrf.jde.asl_physio_joint), 76
dummy_jde() (in module pyhrf.jde.asl_2steps), 40
dump_func() (in module pyhrf.parallel), 247
dump_roi_datasets() (pyhrf.ui.treatment.FMRITreatment method), 183
duplicate_brf() (in module pyhrf.boldsynth.scenarios), 12
duplicate_hrf() (in module pyhrf.boldsynth.scenarios), 12
duplicate_noise_var() (in module pyhrf.boldsynth.scenarios), 12
duplicate_prf() (in module pyhrf.boldsynth.scenarios), 12
DuplicateVariableException, 124

E

EM_solver() (pyhrf.rfir.RFIREstim method), 255
enable() (pyhrf.tools.misc.AnsiColorizer method), 172
enable_draft_testing() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
enable_draft_testing() (pyhrf.ui.jde.JDEMCMCAnalyser method), 180
enable_draft_testing() (pyhrf.ui.treatment.FMRITreatment method), 183
enable_sampling() (pyhrf.sandbox.stats.GSVariable method), 143
ENDC (pyhrf.tools.misc.AnsiColorizer attribute), 172
eps (in module pyhrf.vbjde.vem_bold), 201
error() (pyhrf.tools.message.MessageColor class method), 171
error() (pyhrf.tools.message.MessageNoColor class method), 171
error() (pyhrf.tools.message.NoMessage method), 171
Estim_lnZ_ngbhd_graph() (in module pyhrf.jde.beta), 85
Estim_lnZ_Onsager() (in module pyhrf.jde.beta), 85
ETASampler (class in pyhrf.jde.drift), 87
ETASampler_MultiSess (class in pyhrf.jde.drift), 87
ETASampler_MultiSess (class in pyhrf.jde.jde_multi_sess), 93
ETASampler_MultiSubj (class in pyhrf.jde.jde_multi_sujets), 100
ETASampler_MultiSubj (class in pyhrf.jde.jde_multi_sujets_alpha), 109
evaluate() (pyhrf.tools.aexpression.ArithmeticExpression method), 169
exec_t() (in module pyhrf.ui.treatment), 183
execute() (pyhrf.ui.treatment.FMRITreatment method), 183
expand() (pyhrf.ndarray.xndarray method), 241
expand_array_in_mask() (in module pyhrf.ndarray), 239
expandArray() (pyhrf.boldsynth.spatialconfig.Mapper1D method), 14
expandxndarray() (pyhrf.boldsynth.spatialconfig.xndarrayMapper1D method), 17
expectation_A_asl() (in module pyhrf.vbjde.vem_tools), 211
expectation_A_ms() (in module pyhrf.vbjde.vem_tools), 211
expectation_C_asl() (in module pyhrf.vbjde.vem_tools), 211
expectation_C_ms() (in module pyhrf.vbjde.vem_tools), 212
expectation_G_asl() (in module pyhrf.vbjde.vem_tools), 212
expectation_G_ms() (in module pyhrf.vbjde.vem_tools), 212
expectation_H_asl() (in module pyhrf.vbjde.vem_tools), 212
expectation_H_ms() (in module pyhrf.vbjde.vem_tools), 212
expectation_H_ms_concat() (in module pyhrf.vbjde.vem_tools), 213
expectation_ptilde_hrf() (in module pyhrf.vbjde.vem_tools), 213
expectation_ptilde_labels() (in module pyhrf.vbjde.vem_tools), 213

expectation_Ptilde_Likelihood() (in module pyhrf.vbjde.vem_tools), 213
expectation_ptilde_likelihood() (in module pyhrf.vbjde.vem_tools), 213
expectation_ptilde_nrls() (in module pyhrf.vbjde.vem_tools), 214
expectation_Q_asl() (in module pyhrf.vbjde.vem_tools), 213
expectation_Q_async_asl() (in module pyhrf.vbjde.vem_tools), 213
expectation_Q_ms() (in module pyhrf.vbjde.vem_tools), 213
explode() (pyhrf.ndarray.xndarray method), 242
explode_a() (pyhrf.ndarray.xndarray method), 242
extend_sampled_events() (in module pyhrf.paradigm), 247
extract_file_series() (in module pyhrf.tools.misc), 175
extract_sub_mesh() (in module pyhrf.surface), 257
extract_sub_mesh_with_files() (in module pyhrf.surface), 257
Extract_TTP_whM_from_group() (in module pyhrf.tools.misc), 172
Extract_TTP_whM_hrf() (in module pyhrf.tools.misc), 172
extractRoiMask() (in module pyhrf.tools.misc), 175

F

FALSE_NEG (pyhrf.jde.jde_multi_sujets.LabelSampler attribute), 103
FALSE_NEG (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler attribute), 113
FALSE_NEG (pyhrf.jde.nrl.bigaussian.NRLSampler attribute), 21
FALSE_NEG (pyhrf.jde.nrl.trigaussian.GGGNRLSampler attribute), 34
FALSE_POS (pyhrf.jde.jde_multi_sujets.LabelSampler attribute), 103
FALSE_POS (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler attribute), 113
FALSE_POS (pyhrf.jde.nrl.bigaussian.NRLSampler attribute), 21
FALSE_POS (pyhrf.jde.nrl.trigaussian.GGGNRLSampler attribute), 34
feature_extraction() (in module pyhrf.sandbox.parcellation), 132
FeatureExtractionTest (class in pyhrf.validation.valid_sandbox_parcellation), 197
fetchDefaultParameters() (pyhrf.xmliobak.xmlbase.XMLParamDrivenClass method), 225
field_energy_calculator (class in pyhrf.validation.valid_rndm_field), 197
FieldFuncsTest (class in pyhrf.test.boldsynthTest), 150
figfn() (in module pyhrf.validation.config), 190
FileHandlingTest (class in pyhrf.test.iotest), 153
fill() (pyhrf.ndarray.xndarray method), 242
fillOutputs2() (pyhrf.jde.drift.DriftARSampler method), 86
filter_crashed_results() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
finalizeEstimation() (pyhrf.ui.vb_jde_analyser_asl_fast.JDEVEMAnalyser method), 187
finalizeEstimation() (pyhrf.ui.vb_jde_analyser_bold_fast.JDEVEMAnalyser method), 188
finalizeSampling() (pyhrf.jde.asl.ASLSampler method), 35
finalizeSampling() (pyhrf.jde.asl_physio.ASLPhysioSampler method), 41
finalizeSampling() (pyhrf.jde.asl_physio_1step.ASLPhysioSampler method), 47
finalizeSampling() (pyhrf.jde.asl_physio_1step_params.ASLPhysioSampler method), 53
finalizeSampling() (pyhrf.jde.asl_physio_det_fwdm.ASLPhysioSampler method), 61
finalizeSampling() (pyhrf.jde.asl_physio_hierarchical.ASLPhysioSampler method), 68
finalizeSampling() (pyhrf.jde.asl_physio_joint.ASLPhysioSampler method), 75
finalizeSampling() (pyhrf.jde.drift.DriftARSampler method), 86
finalizeSampling() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
finalizeSampling() (pyhrf.jde.hrf.HRFARSampler method), 87
finalizeSampling() (pyhrf.jde.hrf.HRFSampler method), 88
finalizeSampling() (pyhrf.jde.hrf.HRFSamplerWithRelVar method), 89
finalizeSampling() (pyhrf.jde.hrf.HRFwithHabSampler method), 90
finalizeSampling() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi method), 93
finalizeSampling() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSamp method), 91
finalizeSampling() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
finalizeSampling() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
finalizeSampling() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSa method), 99
finalizeSampling() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 101
finalizeSampling() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
finalizeSampling() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixturePara method), 108
finalizeSampling() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_S

method), 107
finalizeSampling() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
finalizeSampling() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
finalizeSampling() (pyhrf.jde.noise.NoiseARParamsSampler method), 122
finalizeSampling() (pyhrf.jde.noise.NoiseVarianceARSampler method), 123
finalizeSampling() (pyhrf.jde.noise.NoiseVarianceSampler method), 123
finalizeSampling() (pyhrf.jde.noise.NoiseVarianceSamplerWithRelVar method), 123
finalizeSampling() (pyhrf.jde.noise.NoiseVariancewithHabSampler method), 123
finalizeSampling() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
finalizeSampling() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
finalizeSampling() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
finalizeSampling() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler method), 25
finalizeSampling() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
finalizeSampling() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
finalizeSampling() (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler method), 35
finalizeSampling() (pyhrf.jde.samplerbase.GibbsSampler method), 124
finalizeSampling() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
finalizeSampling() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
finalizeSampling() (pyhrf.jde.wsampler.WSampler method), 126
fit() (pyhrf.sandbox.parcellation.Ward method), 130
fit() (pyhrf.sandbox.parcellation.WardAgglomeration method), 131
fit_hrf_two_gammas() (in module pyhrf.vbjde.vem_tools), 214
fix_explosion() (pyhrf.parcellation.Ant method), 249
flatten() (pyhrf.ndarray.xndarray method), 242
flatten_and_graph() (in module pyhrf.graph), 234
flatten_labels_vol() (in module pyhrf.boldsynth.scenarios), 12
flattenArray() (pyhrf.boldsynth.spatialconfig.Mapper1D method), 14
flattenData() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping2D method), 15
flattenElements() (in module pyhrf.boldsynth.spatialconfig), 17
flattenxndarray() (pyhrf.boldsynth.spatialconfig.xndarrayMapper1D method), 17
flip() (in module pyhrf.plot), 253
floatDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
floatTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
FMRIAnalyser (class in pyhrf.ui.analyser_ui), 178
FmriData (class in pyhrf.core), 231
FMRIDataTest (class in pyhrf.test.core_test), 153
FmriGroupData (class in pyhrf.core), 233
FMRISessionSimulationData (class in pyhrf.core), 229
FMRISessionSurfacicData (class in pyhrf.core), 230
FMRISessionVolumicData (class in pyhrf.core), 230
FMRITreatment (class in pyhrf.ui.treatment), 182
foo() (in module pyhrf.test.seppotest), 156
foo() (in module pyhrf.test.test_parallel), 162
foo() (in module pyhrf.test.toolsTest), 168
foo() (in module pyhrf.tools.misc), 175
foo_a() (in module pyhrf.test.toolsTest), 168
foo_default_arg() (in module pyhrf.test.toolsTest), 169
foo_func() (in module pyhrf.test.toolsTest), 169
foo_multiple_returns() (in module pyhrf.test.toolsTest), 169
foo_raise() (in module pyhrf.test.test_parallel), 162
force_end() (pyhrf.parcellation.World method), 249
forge_nrl_files() (in module pyhrf.sandbox.data_parser), 128
formatDuration() (in module pyhrf.tools.misc), 175
format_serie() (in module pyhrf.tools.misc), 175
free_energy_computation() (in module pyhrf.vbjde.vem_tools), 215
from_csv() (pyhrf.paradigm.Paradigm class method), 246
from_param_c() (pyhrf.test.test_xml.T class method), 166
from_py_object (pyhrf.xmlio.UiNode attribute), 259
from_session_dict() (pyhrf.paradigm.Paradigm class method), 246
from_simu_ui() (pyhrf.core.FmriData class method), 231
from_simulation_dict() (pyhrf.core.FmriData class method), 231
from_spm_mat() (pyhrf.paradigm.Paradigm class method), 246
from_stuff() (pyhrf.test.test_xml.B class method), 165
from_surf_files() (pyhrf.core.FmriData class method), 231
from_surf_ui() (pyhrf.core.FmriData class method), 231
from_ui_node (pyhrf.xmlio.Initable attribute), 258
from_vol_files (pyhrf.core.FmriData attribute), 232
from_vol_files_rel (pyhrf.core.FmriData attribute), 232
from_vol_ui (pyhrf.core.FmriData attribute), 232
from_xml (pyhrf.xmlio.UiNode attribute), 259
from_xml() (in module pyhrf.xmlio), 259
from_xml() (in module pyhrf.xmliobak.xmlbase), 226

fromkeys() (pyhrf.tools.backports.OrderedDict class method), 170
fromLattice() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem static method), 14
fromLattice() (pyhrf.boldsynth.spatialconfig.SpatialMapping2 static method), 16
fromMesh() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem static method), 14
fromMesh() (pyhrf.boldsynth.spatialconfig.SpatialMapping2 static method), 16
fun() (in module pyhrf.vbjde.vem_tools), 215
functions (pyhrf.tools.aexpression.ArithmeticExpression attribute), 169
FuncWrapper (class in pyhrf.xmliobak.xmlbase), 223
FWHM() (in module pyhrf.sandbox.parcellation), 129

G

GamGaussMixtureParamsSampler (class in pyhrf.jde.nrl.gammagaussian), 31
GammaGenerator (class in pyhrf.stats.random), 146
gaussian_blur() (in module pyhrf.tools.misc), 175
gaussian_kernel() (in module pyhrf.tools.misc), 175
GaussianGenerator (class in pyhrf.stats.random), 146
genBezierHRF() (in module pyhrf.boldsynth.hrf), 8
genCanoBezierHRF() (in module pyhrf.boldsynth.hrf), 8
generate() (pyhrf.boldsynth.spatialconfig.StateField method), 16
generate() (pyhrf.stats.random.BetaGenerator method), 146
generate() (pyhrf.stats.random.GammaGenerator method), 146
generate() (pyhrf.stats.random.GaussianGenerator method), 147
generate() (pyhrf.stats.random.IndependentMixtureLaw method), 147
generate() (pyhrf.stats.random.LogNormalGenerator method), 147
generate() (pyhrf.stats.random.RandomGenerator method), 147
generate() (pyhrf.stats.random.UniformGenerator method), 147
generate() (pyhrf.stats.random.ZeroGenerator method), 147
generate_features() (in module pyhrf.sandbox.parcellation), 133
genExpHRF() (in module pyhrf.boldsynth.hrf), 8
genGaussianSmoothHRF() (in module pyhrf.boldsynth.hrf), 8
genPepperSaltField() (in module pyhrf.boldsynth.field), 7
genPotts() (in module pyhrf.boldsynth.field), 8
genPottsMap() (in module pyhrf.boldsynth.field), 8
genPriorCov() (in module pyhrf.boldsynth.hrf), 8
GeometryTest (class in pyhrf.test.toolsTest), 167
get() (pyhrf.grid.Task method), 237
get_2Dtable_string() (in module pyhrf.tools.misc), 175
get_accuracy() (pyhrf.jde.asl.DriftCoeffSampler method), 36
get_accuracy() (pyhrf.jde.hrf.HRFSampler method), 88
get_accuracy() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93
get_accuracy() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
get_accuracy() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
get_accuracy() (pyhrf.jde.jde_multi_sess.NRLsBar_Drift_Multi_Sess_Samp method), 95
get_accuracy() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
get_accuracy() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
get_accuracy() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
get_accuracy() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler method), 109
get_accuracy() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
get_accuracy() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
get_accuracy() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
get_accuracy_against_truth() (pyhrf.sandbox.stats.GSVariable method), 143
get_arg_for_ui() (pyhrf.xmlio.Initable method), 258
get_arg_from_ui() (pyhrf.xmlio.Initable method), 258
get_attribute() (pyhrf.xmlio.UiNode method), 259
get_axes_domains() (pyhrf.ndarray.xndarray method), 242
get_axes_ids() (pyhrf.ndarray.xndarray method), 242
get_axis_id() (pyhrf.ndarray.xndarray method), 242
get_axis_name() (pyhrf.ndarray.xndarray method), 242
get_bold_shape() (in module pyhrf.boldsynth.scenarios), 12
get_cache_filename() (in module pyhrf.tools.misc), 176
get_children() (pyhrf.xmlio.UiNode method), 259
get_condition_names() (pyhrf.core.FmriData method), 232
get_current_means() (pyhrf.jde.asl.MixtureParamsSampler method), 37
get_current_means() (pyhrf.jde.asl_physio.MixtureParamsSampler method), 43
get_current_means() (pyhrf.jde.asl_physio_1step.MixtureParamsSampler method), 49
get_current_means() (pyhrf.jde.asl_physio_1step_params.MixtureParamsSa method), 55
get_current_means() (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSamp method), 63
get_current_means() (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSam

method), 70
get_current_means() (pyhrf.jde.asl_physio_joint.MixtureParamsSampler method), 77
get_current_means() (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler method), 103
get_current_means() (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler method), 112
get_current_vars() (pyhrf.jde.asl.MixtureParamsSampler method), 37
get_current_vars() (pyhrf.jde.asl_physio.MixtureParamsSampler method), 43
get_current_vars() (pyhrf.jde.asl_physio_1step.MixtureParamsSampler method), 49
get_current_vars() (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler method), 55
get_current_vars() (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler method), 63
get_current_vars() (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler method), 70
get_current_vars() (pyhrf.jde.asl_physio_joint.MixtureParamsSampler method), 77
get_current_vars() (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler method), 103
get_current_vars() (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler method), 112
get_custom_init() (pyhrf.sandbox.stats.GSVariable method), 143
get_data_file_name() (in module pyhrf.core), 233
get_data_files() (pyhrf.core.FmriData method), 232
get_data_files() (pyhrf.ui.treatment.FMRITreatment method), 183
get_domain() (pyhrf.ndarray.xndarray method), 242
get_domain_idx() (pyhrf.ndarray.xndarray method), 243
get_dshape() (pyhrf.ndarray.xndarray method), 243
get_estim_value_for_check() (pyhrf.sandbox.stats.GSVariable method), 143
get_extra_data() (pyhrf.core.FmriData method), 232
get_extra_info() (pyhrf.ndarray.xndarray method), 243
get_file_blocs() (pyhrf.sandbox.data_parser.StructuredDataParser method), 127
get_files() (pyhrf.sandbox.data_parser.StructuredDataParser method), 128
get_final_labels() (pyhrf.parcellation.World method), 250
get_final_summary() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
get_final_summary() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
get_final_value() (pyhrf.jde.asl.DriftCoeffSampler method), 36
get_final_value() (pyhrf.jde.hrf.HRFSampler method), 88
get_final_value() (pyhrf.jde.hrf.RHSampler method), 90
get_final_value() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93
get_final_value() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
get_final_value() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler method), 109
get_final_value() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
get_func() (pyhrf.tools.misc.Pipeline method), 172
get_graph() (pyhrf.core.FmriData method), 232
get_info() (pyhrf.paradigm.Paradigm method), 246
get_init_func() (pyhrf.xmlio.Initable method), 258
get_joined_and_rastered() (pyhrf.paradigm.Paradigm method), 246
get_joined_durations() (pyhrf.core.FmriData method), 232
get_joined_durations() (pyhrf.paradigm.Paradigm method), 246
get_joined_durations_dim() (pyhrf.paradigm.Paradigm method), 246
get_joined_onsets() (pyhrf.core.FmriData method), 232
get_joined_onsets() (pyhrf.paradigm.Paradigm method), 246
get_joined_onsets_dim() (pyhrf.paradigm.Paradigm method), 246
get_labels() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
get_label() (pyhrf.ui.glm_analyser.GLMAnalyser method), 179
get_label() (pyhrf.ui.glm_ui.GLMAnalyser method), 179
get_label() (pyhrf.ui.jde.JDEAnalyser method), 179
get_last() (pyhrf.jde.samplerbase.Trajectory method), 126
get_last() (pyhrf.sandbox.stats.Trajectory method), 145
get_leaf() (in module pyhrf.tools.misc), 176
get_MAP_labels() (pyhrf.jde.asl.LabelSampler method), 37
get_mat_X() (pyhrf.jde.asl.BOLDResponseSampler method), 36
get_mat_X() (pyhrf.jde.asl.PerfResponseSampler method), 38
get_mat_X() (pyhrf.jde.asl.ResponseSampler method), 39
get_mat_X() (pyhrf.jde.asl_physio.PhysioBOLDResponseSampler method), 44
get_mat_X() (pyhrf.jde.asl_physio.PhysioPerfResponseSampler method), 45
get_mat_X() (pyhrf.jde.asl_physio.ResponseSampler method), 45
get_mat_X() (pyhrf.jde.asl_physio_1step.PhysioBOLDResponseSampler method), 50
get_mat_X() (pyhrf.jde.asl_physio_1step.PhysioPerfResponseSampler method), 51
get_mat_X() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 51
get_mat_X() (pyhrf.jde.asl_physio_1step_params.PhysioBOLDResponseSampler method), 57

get_mat_X() (pyhrf.jde.asl_physio_1step_params.PhysioPerfResponseSampler method), 58
get_mat_X() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
get_mat_X() (pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResponseSampler method), 64
get_mat_X() (pyhrf.jde.asl_physio_det_fwdm.PhysioPerfResponseSampler method), 65
get_mat_X() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
get_mat_X() (pyhrf.jde.asl_physio_hierarchical.PhysioBOLDResponseSampler method), 71
get_mat_X() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseSampler method), 72
get_mat_X() (pyhrf.jde.asl_physio_hierarchical.PhysioTrueBOLDResponseSampler method), 73
get_mat_X() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
get_mat_X() (pyhrf.jde.asl_physio_joint.PhysioBOLDResponseSampler method), 78
get_mat_X() (pyhrf.jde.asl_physio_joint.PhysioPerfResponseSampler method), 79
get_mat_X() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 79
get_mat_XtWX() (pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResponseSampler method), 64
get_mat_XtX() (pyhrf.jde.asl.BOLDResponseSampler method), 36
get_mat_XtX() (pyhrf.jde.asl.PerfResponseSampler method), 38
get_mat_XtX() (pyhrf.jde.asl.ResponseSampler method), 39
get_mat_XtX() (pyhrf.jde.asl_physio.PhysioBOLDResponseSampler method), 44
get_mat_XtX() (pyhrf.jde.asl_physio.PhysioPerfResponseSampler method), 45
get_mat_XtX() (pyhrf.jde.asl_physio.ResponseSampler method), 46
get_mat_XtX() (pyhrf.jde.asl_physio_1step.PhysioBOLDResponseSampler method), 50
get_mat_XtX() (pyhrf.jde.asl_physio_1step.PhysioPerfResponseSampler method), 51
get_mat_XtX() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 51
get_mat_XtX() (pyhrf.jde.asl_physio_1step_params.PhysioBOLDResponseSampler method), 57
get_mat_XtX() (pyhrf.jde.asl_physio_1step_params.PhysioPerfResponseSampler method), 58
get_mat_XtX() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
get_mat_XtX() (pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResponseSampler method), 64
get_mat_XtX() (pyhrf.jde.asl_physio_det_fwdm.PhysioPerfResponseSampler method), 65
get_mat_XtX() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
get_mat_XtX() (pyhrf.jde.asl_physio_hierarchical.PhysioBOLDResponseSampler method), 71
get_mat_XtX() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseSampler method), 72
get_mat_XtX() (pyhrf.jde.asl_physio_hierarchical.PhysioTrueBOLDResponseSampler method), 73
get_mat_XtX() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
get_mat_XtX() (pyhrf.jde.asl_physio_joint.PhysioBOLDResponseSampler method), 78
get_mat_XtX() (pyhrf.jde.asl_physio_joint.PhysioPerfResponseSampler method), 79
get_mat_XtX() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
get_nb_rois() (pyhrf.core.FmriData method), 232
get_nb_trials() (pyhrf.paradigm.Paradigm method), 246
get_nb_voxel_in_mask() (pyhrf.core.FmriData method), 232
get_nipalts() (pyhrf.ndarray.xndarray method), 243
get_new_axis_name() (pyhrf.ndarray.xndarray method), 243
get_orientation() (pyhrf.ndarray.xndarray method), 243
get_output() (pyhrf.sandbox.stats.GibbsSampler method), 144
get_parameters_comments() (pyhrf.xmlio.Initable method), 258
get_parameters_comments() (pyhrf.xmliobak.xmlbase.XMLable method), 226
get_parameters_comments() (pyhrf.xmliobak.xmlbase.XMLable2 method), 226
get_parameters_meta() (pyhrf.xmlio.Initable method), 258
get_parameters_meta() (pyhrf.xmliobak.xmlbase.XMLable method), 226
get_parameters_meta() (pyhrf.xmliobak.xmlbase.XMLable2 method), 226
get_parameters_to_show() (pyhrf.xmlio.Initable method), 258
get_parameters_to_show() (pyhrf.xmliobak.xmlbase.XMLable method), 226
get_parameters_to_show() (pyhrf.xmliobak.xmlbase.XMLable2 method), 226
get_random_init() (pyhrf.sandbox.stats.GSVariable method), 143
get_rois() (pyhrf.paradigm.Paradigm method), 246
get_rlrl() (pyhrf.jde.asl.ResponseSampler method), 39
get_rois() (pyhrf.jde.asl_physio.ResponseSampler method), 46

get_rlrl() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 51
get_rlrl() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
get_rlrl() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
get_rlrl() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
get_rlrl() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
get_roi_id() (pyhrf.core.FmriData method), 232
get_roi_id() (pyhrf.core.FmriGroupData method), 233
get_roi_mask() (pyhrf.core.FmriData method), 232
get_roi_simulation() (in module pyhrf.core), 233
get_src_doc_path() (in module pyhrf.core), 233
get_src_path() (in module pyhrf.core), 233
get_stackX() (pyhrf.jde.asl.BOLDResponseSampler method), 36
get_stackX() (pyhrf.jde.asl.PerfResponseSampler method), 38
get_stackX() (pyhrf.jde.asl.ResponseSampler method), 39
get_stackX() (pyhrf.jde.asl_physio.PhysioBOLDResponseSampler method), 44
get_stackX() (pyhrf.jde.asl_physio.PhysioPerfResponseSampler method), 45
get_stackX() (pyhrf.jde.asl_physio.ResponseSampler method), 46
get_stackX() (pyhrf.jde.asl_physio_1step.PhysioBOLDResponseSampler method), 50
get_stackX() (pyhrf.jde.asl_physio_1step.PhysioPerfResponseSampler method), 51
get_stackX() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 51
get_stackX() (pyhrf.jde.asl_physio_1step_params.PhysioBOLDResponseSampler method), 57
get_stackX() (pyhrf.jde.asl_physio_1step_params.PhysioPerfResponseSampler method), 58
get_stackX() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
get_stackX() (pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResponseSampler method), 64
get_stackX() (pyhrf.jde.asl_physio_det_fwdm.PhysioPerfResponseSampler method), 65
get_stackX() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
get_stackX() (pyhrf.jde.asl_physio_hierarchical.PhysioBOLDResponseSampler method), 71
get_stackX() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseSampler method), 72
get_stackX() (pyhrf.jde.asl_physio_hierarchical.PhysioTrueBOLDResponseSampler method), 73
get_stackX() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
get_stackX() (pyhrf.jde.asl_physio_joint.PhysioBOLDResponseSampler method), 78
get_stackX() (pyhrf.jde.asl_physio_joint.PhysioPerfResponseSampler method), 79
get_stackX() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
get_stimulus_names() (pyhrf.paradigm.Paradigm method), 247
get_string_value() (pyhrf.jde.beta.BetaSampler method), 80
get_string_value() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi method), 93
get_string_value() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParam method), 109
get_string_value() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
get_string_value() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_M method), 26
get_string_value() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
get_summary() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
get_tmax() (pyhrf.paradigm.Paradigm method), 247
get_tmp_path() (in module pyhrf.core), 233
get_true_value() (pyhrf.jde.asl.DriftCoeffSampler method), 36
get_true_value() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93
get_true_value() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
get_true_value() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
get_true_value() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
get_true_value() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sample method), 109
get_true_value() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
get_true_value_for_check() (pyhrf.sandbox.stats.GSVariable method), 148
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl.BOLDMixtureSampler method), 36
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl.PerfMixtureSampler method), 38
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio.BOLDMixtureSampler method), 42
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio.PerfMixtureSampler method), 44
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_1step.BOLDMixtureSampler

method), 47
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_1step.PerfMixtureSampler method), 50
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_1step_params.BOLDMixtureSampler method), 54
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_1step_params.PerfMixtureSampler method), 56
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_det_fwdm.BOLDMixtureSampler method), 61
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_det_fwdm.PerfMixtureSampler method), 64
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_hierarchical.BOLDMixtureSampler method), 69
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_hierarchical.PerfMixtureSampler method), 71
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_joint.BOLDMixtureSampler method), 76
get_true_values_from_simulation_cdefs() (pyhrf.jde.asl_physio_joint.PerfMixtureSampler method), 78
get_true_values_from_simulation_cdefs() (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler method), 104
get_true_values_from_simulation_cdefs() (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler method), 112
get_true_values_from_simulation_dict() (pyhrf.jde.asl.MixtureParamsSampler method), 37
get_true_values_from_simulation_dict() (pyhrf.jde.asl_physio.MixtureParamsSampler method), 43
get_true_values_from_simulation_dict() (pyhrf.jde.asl_physio_1step.MixtureParamsSampler method), 49
get_true_values_from_simulation_dict() (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler method), 55
get_true_values_from_simulation_dict() (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler method), 63
get_true_values_from_simulation_dict() (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler method), 70
get_true_values_from_simulation_dict() (pyhrf.jde.asl_physio_joint.MixtureParamsSampler method), 77
get_value() (pyhrf.tools.misc.Pipeline method), 172
get_values() (pyhrf.tools.misc.Pipeline method), 172
get_variable() (pyhrf.jde.samplerbase.GibbsSampler method), 124
get_variable() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
get_variable() (pyhrf.sandbox.stats.GibbsSampler method), 144
get_variable() (pyhrf.sandbox.stats.GSVariable method), 143
get_variable_value() (pyhrf.sandbox.stats.GibbsSampler method), 144
get_variable_value() (pyhrf.sandbox.stats.GSVariable method), 143
get_voxel_size() (pyhrf.ndarray.xndarray method), 243
get_ybar() (pyhrf.jde.asl.ResponseSampler method), 39
get_ybar() (pyhrf.jde.asl_physio.ResponseSampler method), 46
get_ybar() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 52
get_ybar() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
get_ybar() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
get_ybar() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
get_ybar() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
getargspec() (in module pyhrf.xmliobak.xmlbase), 226
getCanoHRF() (in module pyhrf.boldsynth.hrf), 8
getCanoHRF_tderivative() (in module pyhrf.boldsynth.hrf), 9
getClassId() (pyhrf.boldsynth.spatialconfig.StateField method), 16
getClassifRate() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
getClassName() (pyhrf.boldsynth.spatialconfig.StateField method), 16
getClassNames() (pyhrf.boldsynth.spatialconfig.StateField method), 16
getClosestNeighboursIndexes() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 14
getCoord() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 14
getCoord() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 15
getCoord() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getCurrentMeans() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Mult method), 93
getCurrentMeans() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixturePara method), 109

getCurrentMeans() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
getCurrentMeans() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParamsDrift_Sampler method), 25
getCurrentMeans() (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler method), 35
getCurrentVar() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
getCurrentVar() (pyhrf.jde.hrf.HRFSampler method), 88
getCurrentVar() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
getCurrentVar() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 101
getCurrentVar() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
getCurrentVar() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
getCurrentVar() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
getCurrentVars() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_MultiSess_Sampler method), 93
getCurrentVars() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler method), 109
getCurrentVars() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
getCurrentVars() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParamsDrift_Sampler method), 25
getCurrentVars() (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler method), 35
getFieldValues() (pyhrf.boldsynth.spatialconfig.StateField method), 16
getFinalLabels() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
getFinalVar() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
getFinalVar() (pyhrf.jde.hrf.HRFSampler method), 88
getFinalVar() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
getFinalVar() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
getFinalVar() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
getFinalVar() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
getFinalVar() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
getFitAxes() (pyhrf.jde.samplerbase.GibbsSampler method), 124
getFunctionNames() (pyhrf.tools.aexpression.ArithmeticExpression method), 169
getGlobalOutputs() (pyhrf.jde.asl.ASLSampler method), 35
getGlobalOutputs() (pyhrf.jde.asl_physio.ASLPhysioSampler method), 41
getGlobalOutputs() (pyhrf.jde.asl_physio_1step.ASLPhysioSampler method), 47
getGlobalOutputs() (pyhrf.jde.asl_physio_1step_params.ASLPhysioSampler method), 53
getGlobalOutputs() (pyhrf.jde.asl_physio_det_fwdm.ASLPhysioSampler method), 61
getGlobalOutputs() (pyhrf.jde.asl_physio_hierarchical.ASLPhysioSampler method), 68
getGlobalOutputs() (pyhrf.jde.asl_physio_joint.ASLPhysioSampler method), 75
getGlobalOutputs() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSam method), 92
getGlobalOutputs() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSa method), 99
getGlobalOutputs() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_S method), 107
getGlobalOutputs() (pyhrf.jde.models.BOLDGibbsSampler method), 116
getGlobalOutputs() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
getGlobalOutputs() (pyhrf.jde.samplerbase.GibbsSampler method), 124
getIndex() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 14
getIndex() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 15
getIndex() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getMappedFieldValues() (pyhrf.boldsynth.spatialconfig.StateField method), 16
getMapping() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getMapping() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 15
getMapping() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getMaxIndex() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem method), 14
getMaxNeighbours() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem method), 14
getMean() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
getSmplHistory() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
getNbClasses() (pyhrf.boldsynth.spatialconfig.StateField method), 16
getNbCliques() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getNbVoxels() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getNbVoxels() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 15
getNbVoxels() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17

getNdArrayMask() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getNdArrayMask() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 15
getNdArrayMask() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getNeighbours() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem method), 14
getNeighboursArrays() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem method), 14
getNeighboursCoordLists() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getNeighboursCoordLists() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 15
getNeighboursCoordLists() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getNeighboursCoords() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getNeighboursCoords() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 15
getNeighboursCoords() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getNeighboursIndexes() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getNeighboursIndexes() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 16
getNeighboursIndexes() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getNeighboursIndexLists() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getNeighboursIndexLists() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 16
getNeighboursIndexLists() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getNeighboursLists() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem method), 14
getNeighboursSets() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem method), 14
getOutputs() (pyhrf.jde.asl.BOLDResponseLevelSampler method), 36
getOutputs() (pyhrf.jde.asl.ResponseLevelSampler method), 39
getOutputs() (pyhrf.jde.asl_physio.BOLDResponseLevelSampler method), 42
getOutputs() (pyhrf.jde.asl_physio.DriftCoeffSampler method), 42
getOutputs() (pyhrf.jde.asl_physio.ResponseSampler method), 45
getOutputs() (pyhrf.jde.asl_physio_1step.BOLDResponseLevelSampler method), 48
getOutputs() (pyhrf.jde.asl_physio_1step.DriftCoeffSampler method), 48
getOutputs() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 51
getOutputs() (pyhrf.jde.asl_physio_1step_params.BOLDResponseLevelSam method), 54
getOutputs() (pyhrf.jde.asl_physio_1step_params.DriftCoeffSampler method), 54
getOutputs() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
getOutputs() (pyhrf.jde.asl_physio_det_fwdm.BOLDResponseLevelSample method), 62
getOutputs() (pyhrf.jde.asl_physio_det_fwdm.DriftCoeffSampler method), 62
getOutputs() (pyhrf.jde.asl_physio_hierarchical.BOLDResponseLevelSamp method), 69
getOutputs() (pyhrf.jde.asl_physio_hierarchical.DriftCoeffSampler method), 69
getOutputs() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
getOutputs() (pyhrf.jde.asl_physio_joint.BOLDResponseLevelSampler method), 76
getOutputs() (pyhrf.jde.asl_physio_joint.DriftCoeffSampler method), 76
getOutputs() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 79
getOutputs() (pyhrf.jde.beta.BetaSampler method), 80
getOutputs() (pyhrf.jde.drift.DriftSampler method), 86
getOutputs() (pyhrf.jde.drift.DriftSamplerWithRelVar method), 87
getOutputs() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
getOutputs() (pyhrf.jde.hrf.HRFSampler method), 88
getOutputs() (pyhrf.jde.hrf.RHSampler method), 90
getOutputs() (pyhrf.jde.hrf.ScaleSampler method), 90
getOutputs() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess method), 93
getOutputs() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93
getOutputs() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
getOutputs() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
getOutputs() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
getOutputs() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
getOutputs() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
getOutputs() (pyhrf.jde.jde_multi_sujets.HRFVarianceSubjectSampler method), 101
getOutputs() (pyhrf.jde.jde_multi_sujets.RHGroupSampler

method), 104
getOutputs() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler method), 109
getOutputs() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler method), 109
getOutputs() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
getOutputs() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
getOutputs() (pyhrf.jde.jde_multi_sujets_alpha.HRFVariancePerSubjectSampler method), 110
getOutputs() (pyhrf.jde.jde_multi_sujets_alpha.RHGroupSampler method), 113
getOutputs() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
getOutputs() (pyhrf.jde.nrl.bigaussian.MixtureWeightsSampler method), 20
getOutputs() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
getOutputs() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
getOutputs() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParamsSampler method), 25
getOutputs() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
getOutputs() (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler method), 35
getOutputs() (pyhrf.jde.samplerbase.GibbsSampler method), 124
getOutputs() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
getOutputs() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
getOutputs() (pyhrf.jde.wsampler.WSampler method), 126
getOutputs() (pyhrf.rfir.RFIREstim method), 256
getPositions() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 16
getRocData() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
getRoiMask() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getRoiMask() (pyhrf.boldsynth.spatialconfig.SpatialMapping method), 16
getRoiMask() (pyhrf.boldsynth.spatialconfig.UnboundSpatialMapping method), 17
getRotationMatrix() (in module pyhrf.boldsynth.spatialconfig), 17
getScaleFactor() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
getScaleFactor() (pyhrf.jde.hrf.HRFSampler method), 88
getScaleFactor() (pyhrf.jde.hrf.HRFwithHabSampler method), 90
getScaleFactor() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
getScaleFactor() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
getScaleFactor() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
getScaleFactor() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
getScaleFactor() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
getSubjectSampler() (pyhrf.jde.samplerbase.GibbsSampler method), 124
getSize() (pyhrf.boldsynth.spatialconfig.StateField method), 16
getSummary() (pyhrf.core.FmriData method), 232
getSummary() (pyhrf.core.FmriGroupData method), 233
getTargetAxesNames() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
getTrueMyProfile() (pyhrf.jde.samplerbase.GibbsSampler method), 124
getVariableNames() (pyhrf.tools.aexpression.ArithmeticExpression method), 169
GGGNRMultiSess_cNRLinPyhrf.jde.nrl.trigaussian), 34
GibbsSampler (class in pyhrf.jde.samplerbase), 124
GibbsSampler (class in pyhrf.sandbox.stats), 144
GibbsSamplerVariable (class in pyhrf.jde.samplerbase), 125
GibbsTest (class in pyhrf.test.test_sampler), 164
GiftiTest (class in pyhrf.test.iotest), 154
GLM_method() (in module pyhrf.sandbox.parcellation), 129
glm_nipy() (in module pyhrf.glm), 234
glm_nipy_from_files() (in module pyhrf.glm), 234
GLMAnalyser (class in pyhrf.ui.glm_analyser), 179
GLMAnalyser (class in pyhrf.ui.glm_ui), 179
gm_cdf() (in module pyhrf.stats.misc), 146
gm_mean() (in module pyhrf.stats.misc), 146
gm_sample() (in module pyhrf.stats.random), 147
gm_var() (in module pyhrf.stats.misc), 146
grad_fun() (in module pyhrf.vbjde.vem_tools), 215
graph_from_lattice() (in module pyhrf.graph), 235
graph_from_lattice3D() (in module pyhrf.graph), 235
graph_from_mesh() (in module pyhrf.graph), 235
graph_is_sane() (in module pyhrf.graph), 235
graph_nb_cliques() (in module pyhrf.graph), 235
graph_pool_indexes() (in module pyhrf.graph), 235
graph_pygraph() (in module pyhrf.graph), 235
graph_to_sparse_matrix() (in module pyhrf.graph), 235
GraphBetaMix() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 5
GraphTest (class in pyhrf.test.graphtest), 153
GraphToImage() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 5

group_file_series() (in module pyhrf.tools.misc), 176
GSDefaultCallbackHandler (class in pyhrf.jde.samplerbase), 124
GSPrintCallbackHandler (class in pyhrf.jde.samplerbase), 124
GSVariable (class in pyhrf.sandbox.stats), 143

H

H_Entropy() (in module pyhrf.vbjde.vem_tools), 207
Hab_WN_BiG_BOLDSamplerInput (class in pyhrf.jde.models), 119
habitCondSampler() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
habitCondSamplerParallel() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
habitCondSamplerSerial() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
has_attribute() (pyhrf.xmlio.UiNode method), 259
has_axes() (pyhrf.ndarray.xndarray method), 243
has_axis() (pyhrf.ndarray.xndarray method), 243
has_ext() (in module pyhrf.tools.misc), 176
hash_func_input() (in module pyhrf.tools.misc), 176
hashMask() (in module pyhrf.boldsynth.spatialconfig), 17
haveColor() (pyhrf.tools.message.MessageColor method), 171
haveColor() (pyhrf.tools.message.MessageNoColor method), 171
haveColor() (pyhrf.tools.message.NoMessage method), 171
hc_get_heads() (in module pyhrf.sandbox.parcellation), 133
HierarchicalTasksManager (class in pyhrf.grid), 236
Host (class in pyhrf.grid), 236
hosts_help() (in module pyhrf.grid), 238
HostsManager (class in pyhrf.grid), 236
hrf_canonical_derivatives() (in module pyhrf.sandbox.parcellation), 133
HRF_Drift_Sampler (class in pyhrf.jde.hrf), 89
HRF_Drift_SamplerWithRelVar (class in pyhrf.jde.hrf), 89
hrf_entropy() (in module pyhrf.vbjde.vem_tools), 215
hrf_expectation() (in module pyhrf.vbjde.vem_tools), 215
HRF_Group_Sampler (class in pyhrf.jde.jde_multi_sujets), 101
HRF_Group_Sampler (class in pyhrf.jde.jde_multi_sujets_alpha), 110
HRF_MultiSess_Sampler (class in pyhrf.jde.jde_multi_sess), 93
HRF_Sampler (class in pyhrf.jde.jde_multi_sujets), 102
HRF_Sampler (class in pyhrf.jde.jde_multi_sujets_alpha), 111
HRF_two_parts_Sampler (class in pyhrf.jde.hrf), 89
HRFARSampler (class in pyhrf.jde.hrf), 87
HRFSampler (class in pyhrf.jde.hrf), 87
HRFSamplerWithRelVar (class in pyhrf.jde.hrf), 88
HRFVarianceSubjectSampler (class in pyhrf.jde.jde_multi_sujets), 101
HRFVarianceSubjectSampler (class in pyhrf.jde.jde_multi_sujets_alpha), 109
HRFwithHabSampler (class in pyhrf.jde.hrf), 90
html_body() (in module pyhrf.tools.misc), 176
html_cell() (in module pyhrf.tools.misc), 176
html_div() (in module pyhrf.tools.misc), 176
html_doc() (in module pyhrf.tools.misc), 176
html_head() (in module pyhrf.tools.misc), 176
html_img() (in module pyhrf.tools.misc), 176
html_list_to_row() (in module pyhrf.tools.misc), 176
html_row() (in module pyhrf.tools.misc), 176
html_style() (in module pyhrf.tools.misc), 176
html_table() (in module pyhrf.tools.misc), 176

I

I_MEAN_CA (pyhrf.jde.asl.MixtureParamsSampler attribute), 37
I_MEAN_CA (pyhrf.jde.asl_physio.MixtureParamsSampler attribute), 43
I_MEAN_CA (pyhrf.jde.asl_physio_1step.MixtureParamsSampler attribute), 49
I_MEAN_CA (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler attribute), 55
I_MEAN_CA (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler attribute), 63
I_MEAN_CA (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler attribute), 70
I_MEAN_CA (pyhrf.jde.asl_physio_joint.MixtureParamsSampler attribute), 77
I_MEAN_CA (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Ses attribute), 92
I_MEAN_CA (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler attribute), 103
I_MEAN_CA (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSa attribute), 108
I_MEAN_CA (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler attribute), 112
I_MEAN_CA (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
I_MEAN_CA (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi attribute), 25
I_MEAN_CA (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSam attribute), 31
I_MEAN_CD (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler attribute), 34
I_VAR_CA (pyhrf.jde.asl.MixtureParamsSampler attribute), 37
I_VAR_CA (pyhrf.jde.asl_physio.MixtureParamsSampler attribute), 43

I_VAR_CA (pyhrf.jde.asl_physio_1step.MixtureParamsSampler attribute), 49
I_VAR_CA (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler attribute), 55
I_VAR_CA (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler attribute), 63
I_VAR_CA (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler attribute), 70
I_VAR_CA (pyhrf.jde.asl_physio_joint.MixtureParamsSampler attribute), 77
I_VAR_CA (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_MultiSess attribute), 92
I_VAR_CA (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler attribute), 103
I_VAR_CA (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParams attribute), 108
I_VAR_CA (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler attribute), 112
I_VAR_CA (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
I_VAR_CA (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_MultiSubj attribute), 25
I_VAR_CA (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler attribute), 31
I_VAR_CD (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler attribute), 34
I_VAR_CI (pyhrf.jde.asl.MixtureParamsSampler attribute), 37
I_VAR_CI (pyhrf.jde.asl_physio.MixtureParamsSampler attribute), 43
I_VAR_CI (pyhrf.jde.asl_physio_1step.MixtureParamsSampler attribute), 49
I_VAR_CI (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler attribute), 55
I_VAR_CI (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler attribute), 63
I_VAR_CI (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler attribute), 70
I_VAR_CI (pyhrf.jde.asl_physio_joint.MixtureParamsSampler attribute), 77
I_VAR_CI (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_MultiSess attribute), 92
I_VAR_CI (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler attribute), 103
I_VAR_CI (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParams attribute), 108
I_VAR_CI (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler attribute), 112
I_VAR_CI (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
I_VAR_CI (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_MultiSubj attribute), 25
I_VAR_CI (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler attribute), 31
iparcesian_combine_args() (in module pyhrf.tools.misc), 176
ImageGraph() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 6
includeTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
IndependentMixtureLaw (class in pyhrf.stats.random), 147
info() (pyhrf.tools.message.MessageColor class method), 171
info() (pyhrf.tools.message.MessageNoColor class method), 171
info() (pyhrf.tools.message.NoMessage method), 171
informedGMM() (in module pyhrf.sandbox.parcellation), 133
informedGMM_MV() (in module pyhrf.sandbox.parcellation), 133
InhomogeneousNRLSampler (class in pyhrf.jde.nrl.bigaussian), 31
init() (pyhrf.grid.TaskHierarchical method), 237
init_constants() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
init_dependencies() (pyhrf.tools.misc.Pipeline method), 172
init_dict() (in module pyhrf.rfir), 257
init_edge_data() (in module pyhrf.parcellation), 250
init_new_obj() (pyhrf.xmlio.Initable method), 258
init_observables() (pyhrf.sandbox.stats.GSVariable method), 144
init_sampling() (pyhrf.sandbox.stats.GSVariable method), 144
Initable (class in pyhrf.xmlio), 258
InitableTest (class in pyhrf.test.test_xml), 166
initGlobalObservables() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_Sess method), 92
initGlobalObservables() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_Su method), 99
initGlobalObservables() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Mu
initGlobalObservables() (pyhrf.jde.models.BOLDGibbsSampler method), 116
initGlobalObservables() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
initGlobalObservables() (pyhrf.jde.samplerbase.GibbsSampler method), 124
InitMatrixAndVectors() (pyhrf.rfir.RFIREstim method), 255
initObservables() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
initObservables() (pyhrf.jde.hrf.HRFSampler method), 88
initObservables() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler

method), 94
initObservables() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
initObservables() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
initObservables() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
initObservables() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
initObservables() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
initObservables() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
initObservables() (pyhrf.jde.wsampler.WSampler method), 126
initOutputs2() (pyhrf.jde.drift.DriftARSampler method), 86
InitStorageMat() (pyhrf.rfir.RFIREstim method), 255
inputClass (pyhrf.jde.asl.ASLSampler attribute), 35
inputClass (pyhrf.jde.asl_physio.ASLPhysioSampler attribute), 42
inputClass (pyhrf.jde.asl_physio_1step.ASLPhysioSampler attribute), 47
inputClass (pyhrf.jde.asl_physio_1step_params.ASLPhysioSampler attribute), 53
inputClass (pyhrf.jde.asl_physio_det_fwdm.ASLPhysioSampler attribute), 61
inputClass (pyhrf.jde.asl_physio_hierarchical.ASLPhysioSampler attribute), 68
inputClass (pyhrf.jde.asl_physio_joint.ASLPhysioSampler attribute), 75
inputClass (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_Sess_Sampler attribute), 92
inputClass (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSampler attribute), 99
inputClass (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_SubjSampler attribute), 107
inputClass (pyhrf.jde.models.BOLDGibbsSampler attribute), 116
inputClass (pyhrf.jde.models.BOLDGibbsSampler_AR attribute), 117
inputClass (pyhrf.jde.models.Drift_BOLDGibbsSampler attribute), 119
inputClass (pyhrf.jde.models.W_BOLDGibbsSampler attribute), 119
inputClass (pyhrf.jde.models.W_Drift_BOLDGibbsSampler attribute), 120
inspect_and_append_to_DOM_tree() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
inspect_default_args() (in module pyhrf.tools.misc), 176
inspectable() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
int16DOMWriter() (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler static method), 227
int16TagDOMReader() (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler static method), 227
intDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
intTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
is_accurate() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
is_accurate() (pyhrf.jde.jde_multi_sess.NRLsBar_Drift_Multi_Sess_Sampler method), 95
is_importable() (in module pyhrf.tools.misc), 176
is_inside() (pyhrf.grid.TimeSlot method), 237
is_inside() (pyhrf.grid.TimeSlotList method), 238
is_inside_now() (pyhrf.grid.TimeSlot method), 237
is_leaf_node() (pyhrf.xmlio.UiNode method), 259
is_tmp_file() (in module pyhrf.validation.config), 190
isExpendable() (pyhrf.boldsynth.spatialconfig.xndarrayMapper1D method), 17
isFlattenable() (pyhrf.boldsynth.spatialconfig.xndarrayMapper1D method), 17
isup() (pyhrf.grid.HostsManager method), 236
items() (pyhrf.tools.backports.OrderedDict method), 170
iter_sampling() (pyhrf.jde.samplerbase.GibbsSampler method), 124
iter_sampling() (pyhrf.sandbox.stats.GibbsSampler method), 144
iteritems() (pyhrf.tools.backports.OrderedDict method), 170
iterkeys() (pyhrf.tools.backports.OrderedDict method), 170
itervalues() (pyhrf.tools.backports.OrderedDict method), 170

J

jde_analyse() (in module pyhrf.ui.jde), 180
jde_analyse_2steps_v1() (in module pyhrf.jde.asl_2steps), 40
jde_surf_from_files() (in module pyhrf.ui.treatment), 183
jde_vem_bold() (in module pyhrf.vbjde.vem_bold), 201
jde_vol_from_files() (in module pyhrf.ui.treatment), 184
JDEAnalyser (class in pyhrf.ui.jde), 179
JDEMCMCAnalyser (class in pyhrf.ui.jde), 179
JDETest (class in pyhrf.test.jdetest), 155
JDETest (class in pyhrf.validation.valid_jde_bold_mono_subj_sess), 195
JDEVEMAnalyser (class in pyhrf.ui.vb_jde_analyser), 185
JDEVEMAnalyser (class in pyhrf.ui.vb_jde_analyser_asl_fast), 187
JDEVEMAnalyser (class in pyhrf.ui.vb_jde_analyser_bold_fast), 188
join_sessions() (pyhrf.paradigm.Paradigm method), 247

joinOutputs() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178

K

keep_only_rois() (pyhrf.core.FmriData method), 232
kerMask2D_4n (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem attribute), 14
kerMask3D_6n (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem attribute), 14
key() (pyhrf.grid.User method), 238
keys() (pyhrf.tools.backports.OrderedDict method), 170
kill() (pyhrf.grid.TasksStarter method), 237
kill_threads() (in module pyhrf.grid), 238

L

L_CA (pyhrf.jde.asl.LabelSampler attribute), 37
L_CA (pyhrf.jde.asl.MixtureParamsSampler attribute), 37
L_CA (pyhrf.jde.asl_physio.LabelSampler attribute), 42
L_CA (pyhrf.jde.asl_physio.MixtureParamsSampler attribute), 43
L_CA (pyhrf.jde.asl_physio_1step.LabelSampler attribute), 48
L_CA (pyhrf.jde.asl_physio_1step.MixtureParamsSampler attribute), 49
L_CA (pyhrf.jde.asl_physio_1step_params.LabelSampler attribute), 54
L_CA (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler attribute), 55
L_CA (pyhrf.jde.asl_physio_det_fwdm.LabelSampler attribute), 62
L_CA (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler attribute), 63
L_CA (pyhrf.jde.asl_physio_hierarchical.LabelSampler attribute), 69
L_CA (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler attribute), 70
L_CA (pyhrf.jde.asl_physio_joint.LabelSampler attribute), 76
L_CA (pyhrf.jde.asl_physio_joint.MixtureParamsSampler attribute), 77
L_CA (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler attribute), 93
L_CA (pyhrf.jde.jde_multi_sujets.LabelSampler attribute), 103
L_CA (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler attribute), 103
L_CA (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler attribute), 108
L_CA (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler attribute), 112
L_CA (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler attribute), 112
L_CA (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler attribute), 113
L_CA (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
L_CA (pyhrf.jde.nrl.bigaussian.NRLSampler attribute), 21
L_CA (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Sess_N attribute), 25
L_CA (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler attribute), 31
L_CA (pyhrf.jde.nrl.trigaussian.GGGNRLSampler attribute), 34
L_CA (pyhrf.jde.wsampler.W_Drift_Sampler attribute), 127
L_CA (pyhrf.jde.wsampler.WSampler attribute), 126
L_CD (pyhrf.jde.nrl.trigaussian.GGGNRLSampler attribute), 34
L_CD (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler attribute), 34
L_CI (pyhrf.jde.asl.LabelSampler attribute), 37
L_CI (pyhrf.jde.asl.MixtureParamsSampler attribute), 37
L_CI (pyhrf.jde.asl_physio.LabelSampler attribute), 42
L_CI (pyhrf.jde.asl_physio.MixtureParamsSampler attribute), 43
L_CI (pyhrf.jde.asl_physio_1step.LabelSampler attribute), 48
L_CI (pyhrf.jde.asl_physio_1step.MixtureParamsSampler attribute), 49
L_CI (pyhrf.jde.asl_physio_1step_params.LabelSampler attribute), 54
L_CI (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler attribute), 55
L_CI (pyhrf.jde.asl_physio_det_fwdm.LabelSampler attribute), 62
L_CI (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler attribute), 63
L_CI (pyhrf.jde.asl_physio_hierarchical.LabelSampler attribute), 69
L_CI (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler attribute), 70
L_CI (pyhrf.jde.asl_physio_joint.LabelSampler attribute), 76
L_CI (pyhrf.jde.asl_physio_joint.MixtureParamsSampler attribute), 77
L_CI (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess_NRLs attribute), 93
L_CI (pyhrf.jde.jde_multi_sujets.LabelSampler attribute), 103
L_CI (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler attribute), 103
L_CI (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler attribute), 108
L_CI (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler attribute), 112

L_CI (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler attribute), 112
L_CI (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler attribute), 113
L_CI (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
L_CI (pyhrf.jde.nrl.bigaussian.NRLSampler attribute), 21
L_CI (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParamsSampler attribute), 25
L_CI (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler attribute), 31
L_CI (pyhrf.jde.nrl.trigaussian.GGGNRLSampler attribute), 34
L_CI (pyhrf.jde.wsampler.W_Drift_Sampler attribute), 127
L_CI (pyhrf.jde.wsampler.WSampler attribute), 126
label() (pyhrf.xmlio.UiNode method), 259
labels_ (pyhrf.sandbox.parcellation.Ward attribute), 130
labels_ (pyhrf.sandbox.parcellation.WardAgglomeration attribute), 131
labels_entropy() (in module pyhrf.vbjde.vem_tools), 216
labels_expectation() (in module pyhrf.vbjde.vem_tools), 217
LabelSampler (class in pyhrf.jde.asl), 36
LabelSampler (class in pyhrf.jde.asl_physio), 42
LabelSampler (class in pyhrf.jde.asl_physio_1step), 48
LabelSampler (class in pyhrf.jde.asl_physio_1step_params), 54
LabelSampler (class in pyhrf.jde.asl_physio_det_fwdm), 62
LabelSampler (class in pyhrf.jde.asl_physio_hierarchical), 69
LabelSampler (class in pyhrf.jde.asl_physio_joint), 76
LabelSampler (class in pyhrf.jde.jde_multi_sujets), 103
LabelSampler (class in pyhrf.jde.jde_multi_sujets_alpha), 112
LaplacianPdf() (in module pyhrf.jde.nrl.habituation), 32
lattice_indexes() (in module pyhrf.boldsynth.spatialconfig), 17
len() (pyhrf.ndarray.xndarray method), 243
linear_rf_operator() (in module pyhrf.sandbox.physio), 138
linear_rf_operator() (in module pyhrf.sandbox.physio_params), 142
linkNodes() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 7
linkNodesSets() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 7
linkToData() (pyhrf.jde.asl.DriftCoeffSampler method), 36
linkToData() (pyhrf.jde.asl.DriftVarianceSampler method), 36
linkToData() (pyhrf.jde.asl.LabelSampler method), 37
linkToData() (pyhrf.jde.asl.MixtureParamsSampler method), 37
linkToData() (pyhrf.jde.asl.NoiseVarianceSampler method), 37
linkToData() (pyhrf.jde.asl.PerfBaselineSampler method), 38
linkToData() (pyhrf.jde.asl.PerfBaselineVarianceSampler method), 38
linkToData() (pyhrf.jde.asl.ResponseLevelSampler method), 39
linkToData() (pyhrf.jde.asl.ResponseSampler method), 39
linkToData() (pyhrf.jde.asl.ResponseVarianceSampler method), 39
linkToData() (pyhrf.jde.asl_physio.DriftCoeffSampler method), 42
linkToData() (pyhrf.jde.asl_physio.DriftVarianceSampler method), 42
linkToData() (pyhrf.jde.asl_physio.LabelSampler method), 43
linkToData() (pyhrf.jde.asl_physio.MixtureParamsSampler method), 43
linkToData() (pyhrf.jde.asl_physio.NoiseVarianceSampler method), 43
linkToData() (pyhrf.jde.asl_physio.PerfBaselineSampler method), 43
linkToData() (pyhrf.jde.asl_physio.PerfBaselineVarianceSampler method), 44
linkToData() (pyhrf.jde.asl_physio.ResponseLevelSampler method), 45
linkToData() (pyhrf.jde.asl_physio.ResponseSampler method), 46
linkToData() (pyhrf.jde.asl_physio.ResponseVarianceSampler method), 46
linkToData() (pyhrf.jde.asl_physio_1step.DriftCoeffSampler method), 48
linkToData() (pyhrf.jde.asl_physio_1step.DriftVarianceSampler method), 48
linkToData() (pyhrf.jde.asl_physio_1step.LabelSampler method), 48
linkToData() (pyhrf.jde.asl_physio_1step.MixtureParamsSampler method), 49
linkToData() (pyhrf.jde.asl_physio_1step.NoiseVarianceSampler method), 49
linkToData() (pyhrf.jde.asl_physio_1step.PerfBaselineSampler method), 49
linkToData() (pyhrf.jde.asl_physio_1step.PerfBaselineVarianceSampler method), 49
linkToData() (pyhrf.jde.asl_physio_1step.ResponseLevelSampler method), 51
linkToData() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 52
linkToData() (pyhrf.jde.asl_physio_1step.ResponseVarianceSampler

method), 52
linkToData() (pyhrf.jde.asl_physio_1step_params.DriftCoeffSampler method), 54
linkToData() (pyhrf.jde.asl_physio_1step_params.DriftVarianceSampler method), 54
linkToData() (pyhrf.jde.asl_physio_1step_params.LabelSampler method), 55
linkToData() (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler method), 55
linkToData() (pyhrf.jde.asl_physio_1step_params.NoiseVarianceSampler method), 55
linkToData() (pyhrf.jde.asl_physio_1step_params.PerfBaselineSampler method), 55
linkToData() (pyhrf.jde.asl_physio_1step_params.PerfBaselineVarianceSampler method), 56
linkToData() (pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler method), 59
linkToData() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
linkToData() (pyhrf.jde.asl_physio_1step_params.ResponseVarianceSampler method), 60
linkToData() (pyhrf.jde.asl_physio_det_fwdm.DriftCoeffSampler method), 62
linkToData() (pyhrf.jde.asl_physio_det_fwdm.DriftVarianceSampler method), 62
linkToData() (pyhrf.jde.asl_physio_det_fwdm.LabelSampler method), 62
linkToData() (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler method), 63
linkToData() (pyhrf.jde.asl_physio_det_fwdm.NoiseVarianceSampler method), 63
linkToData() (pyhrf.jde.asl_physio_det_fwdm.PerfBaselineSampler method), 63
linkToData() (pyhrf.jde.asl_physio_det_fwdm.PerfBaselineVarianceSampler method), 64
linkToData() (pyhrf.jde.asl_physio_det_fwdm.ResponseLevelSampler method), 65
linkToData() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
linkToData() (pyhrf.jde.asl_physio_det_fwdm.ResponseVarianceSampler method), 66
linkToData() (pyhrf.jde.asl_physio_hierarchical.DriftCoeffSampler method), 69
linkToData() (pyhrf.jde.asl_physio_hierarchical.DriftVarianceSampler method), 69
linkToData() (pyhrf.jde.asl_physio_hierarchical.LabelSampler method), 70
linkToData() (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler method), 70
linkToData() (pyhrf.jde.asl_physio_hierarchical.NoiseVarianceSampler method), 70
linkToData() (pyhrf.jde.asl_physio_hierarchical.PerfBaselineSampler method), 70
linkToData() (pyhrf.jde.asl_physio_hierarchical.PerfBaselineVarianceSampler method), 71
linkToData() (pyhrf.jde.asl_physio_hierarchical.PhysioBOLDResponseVariSampler method), 72
linkToData() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseVarianSampler method), 72
linkToData() (pyhrf.jde.asl_physio_hierarchical.PhysioTrueBOLDResponseSampler method), 73
linkToData() (pyhrf.jde.asl_physio_hierarchical.ResponseLevelSampler method), 73
linkToData() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
linkToData() (pyhrf.jde.asl_physio_joint.DriftCoeffSampler method), 76
linkToData() (pyhrf.jde.asl_physio_joint.DriftVarianceSampler method), 76
linkToData() (pyhrf.jde.asl_physio_joint.LabelSampler method), 77
linkToData() (pyhrf.jde.asl_physio_joint.MixtureParamsSampler method), 77
linkToData() (pyhrf.jde.asl_physio_joint.NoiseVarianceSampler method), 77
linkToData() (pyhrf.jde.asl_physio_joint.PerfBaselineSampler method), 77
linkToData() (pyhrf.jde.asl_physio_joint.PerfBaselineVarianceSampler method), 78
linkToData() (pyhrf.jde.asl_physio_joint.PhysioJointResponseVarianceSampler method), 78
linkToData() (pyhrf.jde.asl_physio_joint.ResponseLevelSampler method), 79
linkToData() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
linkToData() (pyhrf.jde.beta.BetaSampler method), 80
linkToData() (pyhrf.jde.drift.DriftARSampler method), 86
linkToData() (pyhrf.jde.drift.DriftSampler method), 86
linkToData() (pyhrf.jde.drift.DriftSamplerWithRelVar method), 87
linkToData() (pyhrf.jde.drift.ETASampler method), 87
linkToData() (pyhrf.jde.drift.ETASampler_MultiSess method), 87
linkToData() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 89
linkToData() (pyhrf.jde.hrf.HRFARSampler method), 87
linkToData() (pyhrf.jde.hrf.HRFSampler method), 88
linkToData() (pyhrf.jde.hrf.HRFSamplerWithRelVar method), 89
linkToData() (pyhrf.jde.hrf.HRFwithHabSampler method), 90
linkToData() (pyhrf.jde.hrf.RHSampler method), 90
linkToData() (pyhrf.jde.hrf.ScaleSampler method), 90
linkToData() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess method), 93
linkToData() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93

linkToData() (pyhrf.jde.jde_multi_sess.ETASampler_MultiSess method), 93
linkToData() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
linkToData() (pyhrf.jde.jde_multi_sess.NoiseVariance_Drift_MultiSess method), 96
linkToData() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
linkToData() (pyhrf.jde.jde_multi_sess.NRLsBar_Drift_Multi_Sess method), 95
linkToData() (pyhrf.jde.jde_multi_sess.Variance_GaussianNRL_Multi_Sess method), 96
linkToData() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
linkToData() (pyhrf.jde.jde_multi_sujets.ETASampler_MultiSubj method), 100
linkToData() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
linkToData() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 102
linkToData() (pyhrf.jde.jde_multi_sujets.HRFVarianceSubjectSampler method), 101
linkToData() (pyhrf.jde.jde_multi_sujets.LabelSampler method), 103
linkToData() (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler method), 104
linkToData() (pyhrf.jde.jde_multi_sujets.NoiseVariance_Drift_MultiSubj_Sampler method), 104
linkToData() (pyhrf.jde.jde_multi_sujets.NRLs_Sampler method), 104
linkToData() (pyhrf.jde.jde_multi_sujets.RHGroupSampler method), 104
linkToData() (pyhrf.jde.jde_multi_sujets.Variance_GaussianNRL_MultiSubj_Sampler method), 104
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.Alpha_hgroup_Sampler method), 106
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.AlphaVar_Sampler method), 106
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler method), 109
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler method), 109
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.ETASampler_MultiSubj method), 109
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.HRFVarianceSubjectSampler method), 110
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler method), 112
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler method), 112
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.NoiseVariance_Drift_MultiSubj_Sampler method), 113
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler method), 113
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.RHGroupSampler method), 113
linkToData() (pyhrf.jde.jde_multi_sujets_alpha.Variance_GaussianNRL_MultiSubj_Sampler method), 113
linkToData() (pyhrf.jde.noise.NoiseARParamsSampler method), 122
linkToData() (pyhrf.jde.noise.NoiseVariance_Drift_Sampler method), 123
linkToData() (pyhrf.jde.noise.NoiseVarianceSampler method), 123
linkToData() (pyhrf.jde.nrl.ar.NRLARSampler method), 18
linkToData() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
linkToData() (pyhrf.jde.nrl.bigaussian.MixtureWeightsSampler method), 20
linkToData() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
linkToData() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
linkToData() (pyhrf.jde.nrl.bigaussian.Variance_GaussianNRL_Multi_Sess method), 24
linkToData() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Sess method), 26
linkToData() (pyhrf.jde.nrl.bigaussian_drift.NRLsBar_Drift_Multi_Sess_Sampler method), 29
linkToData() (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler method), 31
linkToData() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
linkToData() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
linkToData() (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler method), 35
linkToData() (pyhrf.jde.samplerbase.GibbsSampler method), 124
linkToData() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
linkToData() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
linkToData() (pyhrf.jde.wsampler.WSampler method), 126
linkToData() (pyhrf.rfir.RFIREstim method), 256
list_data_file_names() (in module pyhrf.core), 233
listDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
listTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
load() (pyhrf.ndarray.xndarray static method), 243
load_configuration() (in module pyhrf.configuration), 229

load_drawn_labels() (in module pyhrf.boldsynth.scenarios), 12
load_hrf_territories() (in module pyhrf.boldsynth.scenarios), 12
load_many_hrf_territories() (in module pyhrf.boldsynth.scenarios), 12
load_surf_bold_mask() (in module pyhrf.core), 233
load_vol_bold_and_mask() (in module pyhrf.core), 233
LoadBaseLogPartFctRef() (in module pyhrf.jde.beta), 85
loadBetaGrid() (pyhrf.jde.beta.BetaSampler method), 80
log() (pyhrf.xmlio.UiNode method), 259
log_help() (in module pyhrf.grid), 238
logger (in module pyhrf.vbjde.vem_bold), 201
loglikelihood_computation() (in module pyhrf.sandbox.parcellation), 133
LogNormalGenerator (class in pyhrf.stats.random), 147
logpf_ising_onsager() (in module pyhrf.jde.beta), 86

M

main() (in module pyhrf.grid), 238
main_safe() (in module pyhrf.grid), 238
Main_vbjde_Extension_constrained() (in module pyhrf.vbjde.vem_bold_constrained), 204
Main_vbjde_Extension_constrained_stable() (in module pyhrf.vbjde.vem_bold_constrained), 204
Main_vbjde_physio() (in module pyhrf.vbjde.vem_asl_models_fast), 199
Main_vbjde_physio() (in module pyhrf.vbjde.vem_asl_models_fast_ms), 200
Main_vbjde_physio() (in module pyhrf.vbjde.vem_bold_models_fast), 205
Main_vbjde_physio() (in module pyhrf.vbjde.vem_bold_models_fast_ms), 206
Main_vbjde_Python_constrained() (in module pyhrf.vbjde.vem_bold_constrained), 205
make_mask() (in module pyhrf.sandbox.make_parcellation), 129
make_outfile() (in module pyhrf.ui.treatment), 184
make_outputs_multi_subjects() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
make_outputs_single_subject() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
make_parcellation() (in module pyhrf.sandbox.make_parcellation), 129
make_parcellation_cubed_blobs_from_file() (in module pyhrf.parcellation), 250
make_parcellation_from_files() (in module pyhrf.parcellation), 250
make_parcellation_surf_from_files() (in module pyhrf.parcellation), 250
makePrecalculations() (pyhrf.jde.asl.WN_BiG_ASLSamplerInput method), 39
makePrecalculations() (pyhrf.jde.asl_physio.WN_BiG_ASLSamplerInput method), 46
makePrecalculations() (pyhrf.jde.asl_physio_1step.WN_BiG_ASLSamplerI method), 52
makePrecalculations() (pyhrf.jde.asl_physio_1step_params.WN_BiG_ASLS method), 60
makePrecalculations() (pyhrf.jde.asl_physio_det_fwdm.WN_BiG_ASLSam method), 66
makePrecalculations() (pyhrf.jde.asl_physio_hierarchical.WN_BiG_ASLSa method), 74
makePrecalculations() (pyhrf.jde.asl_physio_joint.WN_BiG_ASLSamplerI method), 80
makePrecalculations() (pyhrf.jde.jde_multi_sess.BOLDSampler_Multi_Ses method), 92
makePrecalculations() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSuj method), 100
makePrecalculations() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_M method), 108
makePrecalculations() (pyhrf.jde.models.ARN_BiG_BOLDSamplerInput method), 115
makePrecalculations() (pyhrf.jde.models.BOLDSampler_Multi_SessInput method), 118
makePrecalculations() (pyhrf.jde.models.BOLDSamplerInput method), 118
makePrecalculations() (pyhrf.jde.models.Hab_WN_BiG_BOLDSamplerInp method), 119
makePrecalculations() (pyhrf.jde.models.WN_BiG_BOLDSamplerInput method), 119
makePrecalculations() (pyhrf.jde.models.WN_BiG_Drift_BOLDSamplerIn method), 119
makeQuietOutputs() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
makeQuietOutputs() (pyhrf.test.test_glm.NipyGLMTest method), 158
manageMapping() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
manageMappingInit() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
map_dict() (in module pyhrf.tools.misc), 176
map_onto() (pyhrf.ndarray.xndarray method), 243
mapData() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping2 method), 15
Mapper1D (class in pyhrf.boldsynth.spatialconfig), 14
Mapper1DTest (class in pyhrf.test.boldsynthTest), 150
mapVoxData() (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping method), 15
mark_wrong_labels() (in module pyhrf.stats.misc), 146
markWrongLabels() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
mask_to_coords() (in module pyhrf.boldsynth.spatialconfig), 17
MaskToGraph() (in module

pyhrf.boldsynth.pottsfield.swendsenwang), 6
maskToMapping() (in module pyhrf.boldsynth.spatialconfig), 17
match_init_args() (in module pyhrf.xmliobak.xmlbase), 226
MatlabXMLHandler (class in pyhrf.xmliobak.xmlmatlab), 226
max() (pyhrf.ndarray.xndarray method), 244
maximization_beta_m2_asl() (in module pyhrf.vbjde.vem_tools), 218
maximization_beta_m2_scipy_asl() (in module pyhrf.vbjde.vem_tools), 218
maximization_beta_m4_asl() (in module pyhrf.vbjde.vem_tools), 218
maximization_class_proba() (in module pyhrf.vbjde.vem_tools), 218
maximization_drift_coeffs() (in module pyhrf.vbjde.vem_tools), 218
maximization_LA_asl() (in module pyhrf.vbjde.vem_tools), 218
maximization_Mu_asl() (in module pyhrf.vbjde.vem_tools), 218
maximization_mu_sigma_asl() (in module pyhrf.vbjde.vem_tools), 218
maximization_mu_sigma_ms() (in module pyhrf.vbjde.vem_tools), 218
maximization_noise_var() (in module pyhrf.vbjde.vem_tools), 218
maximization_sigma_asl() (in module pyhrf.vbjde.vem_tools), 219
maximization_sigma_noise_asl() (in module pyhrf.vbjde.vem_tools), 219
maximization_sigmaH() (in module pyhrf.vbjde.vem_tools), 219
maximization_sigmaH_prior() (in module pyhrf.vbjde.vem_tools), 219
maximum() (in module pyhrf.vbjde.vem_tools), 219
MC_comp_pfmethods_ML() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
MC_comp_pfmethods_ML_3C() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
mean() (pyhrf.ndarray.xndarray method), 244
MeasureTest (class in pyhrf.test.test_parcellation), 163
Memory() (in module pyhrf.sandbox.parcellation), 129
merge() (in module pyhrf.ndarray), 239
merge_default_kwargs() (in module pyhrf.parallel), 247
merge_fmri_sessions() (in module pyhrf.core), 233
merge_fmri_subjects() (in module pyhrf.core), 233
merge_onsets() (in module pyhrf.paradigm), 247
mesh_contour() (in module pyhrf.surface), 257
mesh_contour_with_files() (in module pyhrf.surface), 257
Message (class in pyhrf.tools.message), 171
MessageColor (class in pyhrf.tools.message), 171
MessageNoColor (class in pyhrf.tools.message), 171
meta_obj (pyhrf.core.FmriData attribute), 231
MH_ARsampling_gauss_proposal() (pyhrf.jde.noise.NoiseARParamsSampler method), 122
MH_ARsampling_optim() (pyhrf.jde.noise.NoiseARParamsSampler method), 122
min() (pyhrf.ndarray.xndarray method), 244
MiscCommandTest (class in pyhrf.test.commandTest), 152
MiscTest (class in pyhrf.test.toolsTest), 167
mix_cmap() (in module pyhrf.plot), 253
mixtp_to_str() (in module pyhrf.sandbox.parcellation), 133
MixtureParamsSampler (class in pyhrf.jde.asl), 37
MixtureParamsSampler (class in pyhrf.jde.asl_physio), 43
MixtureParamsSampler (class in pyhrf.jde.asl_physio_1step), 48
MixtureParamsSampler (class in pyhrf.jde.asl_physio_1step_params), 55
MixtureParamsSampler (class in pyhrf.jde.asl_physio_det_fwdm), 62
MixtureParamsSampler (class in pyhrf.jde.asl_physio_hierarchical), 70
MixtureParamsSampler (class in pyhrf.jde.asl_physio_joint), 77
MixtureParamsSampler (class in pyhrf.jde.jde_multi_sujets), 103
MixtureParamsSampler (class in pyhrf.jde.jde_multi_sujets_alpha), 112
MixtureWeightsSampler (class in pyhrf.jde.nrl.bigaussian), 20
mode_help() (in module pyhrf.grid), 238
montecarlo() (in module pyhrf.tools.misc), 176
mountDefaultHandlers() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
msqrt() (in module pyhrf.jde.hrf), 90
mult() (in module pyhrf.vbjde.vem_tools), 219
multiply() (pyhrf.ndarray.xndarray method), 244
MultiSessTest (class in pyhrf.test.jdetest), 155
MultiSessTest (class in pyhrf.validation.valid_jde_bold_mono_subj_multi_sess), 194
MultiSubjTest (class in pyhrf.test.test_jde_multi_subj), 158
my_func() (in module pyhrf.tools.misc), 176

N

n_components_ (pyhrf.sandbox.parcellation.Ward at-

tribute), 130
n_leaves_ (pyhrf.sandbox.parcellation.Ward attribute), 130
n_leaves_ (pyhrf.sandbox.parcellation.WardAgglomeration attribute), 131
NB_PARAMS (pyhrf.jde.asl.MixtureParamsSampler attribute), 37
NB_PARAMS (pyhrf.jde.asl_physio.MixtureParamsSampler attribute), 43
NB_PARAMS (pyhrf.jde.asl_physio_1step.MixtureParamsSampler attribute), 49
NB_PARAMS (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler attribute), 55
NB_PARAMS (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler attribute), 63
NB_PARAMS (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler attribute), 70
NB_PARAMS (pyhrf.jde.asl_physio_joint.MixtureParamsSampler attribute), 77
NB_PARAMS (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler attribute), 93
NB_PARAMS (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler attribute), 103
NB_PARAMS (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler attribute), 108
NB_PARAMS (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler attribute), 112
NB_PARAMS (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
NB_PARAMS (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParamsSampler attribute), 25
NB_PARAMS (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler attribute), 31
NB_PARAMS (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler attribute), 34
nbNeighboursOrder1 (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping attribute), 15
nbNeighboursOrder2 (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping attribute), 15
nc2DGrid() (in module pyhrf.tools.misc), 176
NeighbourhoodSystem (class in pyhrf.boldsynth.spatialconfig), 14
next() (pyhrf.grid.TaskHierarchical method), 237
next() (pyhrf.grid.TaskList method), 237
NiftiTest (class in pyhrf.test.iotest), 154
NipyGLMTest (class in pyhrf.test.test_glm), 158
no_tty_check() (pyhrf.tools.misc.AnsiColorizer method), 172
NoiseARParamsSampler (class in pyhrf.jde.noise), 122
NoiseVariance_Drift_Multi_Sess_Sampler (class in pyhrf.jde.jde_multi_sess), 96
NoiseVariance_Drift_MultiSubj_Sampler (class in pyhrf.jde.jde_multi_sujets), 104
NoiseVariance_Drift_MultiSubj_Sampler (class in pyhrf.jde.jde_multi_sujets_alpha), 113
NoiseVariance_Drift_Sampler (class in pyhrf.jde.noise), 123
NoiseVarianceARSampler (class in pyhrf.jde.noise), 123
NoiseVarianceSampler (class in pyhrf.jde.asl), 37
NoiseVarianceSampler (class in pyhrf.jde.asl_physio), 43
NoiseVarianceSampler (class in pyhrf.jde.asl_physio_1step), 49
NoiseVarianceSampler (class in pyhrf.jde.asl_physio_1step_params), 55
NoiseVarianceSampler (class in pyhrf.jde.asl_physio_det_fwdm), 63
NoiseVarianceSampler (class in pyhrf.jde.asl_physio_hierarchical), 70
NoiseVarianceSampler (class in pyhrf.jde.asl_physio_joint), 77
NoiseVarianceSampler (class in pyhrf.jde.noise), 123
NoiseVarianceSamplerWithRelVar (class in pyhrf.jde.noise), 123
NoiseVariancesHRFSampler (class in pyhrf.jde.noise), 123
NoMessage (class in pyhrf.tools.message), 171
non_existent_canditate() (in module pyhrf.tools.misc), 176
non_existent_file() (in module pyhrf.tools.misc), 176
nonsDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
nonsTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
norm1_constraint() (in module pyhrf.vbjde.vem_tools), 219
norm2_bc() (in module pyhrf.sandbox.parcellation), 133
normpdf() (in module pyhrf.vbjde.vem_tools), 220
not_available_status (pyhrf.grid.HostsManager attribute), 236
Nothing() (in module pyhrf.tools.misc), 176
NRL_Drift_Sampler (class in pyhrf.jde.nrl.bigaussian_drift), 26
NRL_Drift_SamplerWithRelVar (class in pyhrf.jde.nrl.bigaussian_drift), 26
NRL_Multi_Sess_Sampler (class in pyhrf.jde.jde_multi_sess), 94
NRL_Multi_Sess_Sampler (class in pyhrf.jde.nrl.bigaussian), 23
NRLARSampler (class in pyhrf.jde.nrl.ar), 18
nrls_entropy() (in module pyhrf.vbjde.vem_tools), 220
nrls_expectation() (in module pyhrf.vbjde.vem_tools), 220
NRLs_Sampler (class in pyhrf.jde.jde_multi_sujets), 104
NRLs_Sampler (class in pyhrf.jde.jde_multi_sujets_alpha), 112
NRLSampler (class in pyhrf.jde.nrl.bigaussian), 20
NRLSamplerWithRelVar (class in pyhrf.jde.nrl.bigaussian), 22

NRLsBar_Drift_Multi_Sess_Sampler (class in pyhrf.jde.jde_multi_sess), 95
NRLsBar_Drift_Multi_Sess_Sampler (class in pyhrf.jde.nrl.bigaussian_drift), 27
NRLwithHabSampler (class in pyhrf.jde.nrl.habituation), 32
numpy_array_from_string() (in module pyhrf.xmlio), 259
NUMPY_ARRAY_TAG_NAME (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler attribute), 227
NUMPY_INT16_TAG_NAME (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler attribute), 227
NUMPY_INT32_TAG_NAME (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler attribute), 227
numpyObjectTagDOMReader() (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler static method), 227
numpyObjectTagDOMWriter() (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler static method), 227
NumpyXMLHandler (class in pyhrf.xmliobak.xmlnumpy), 227

O

Object (class in pyhrf.core), 233
ObsField2DTest (class in pyhrf.validation.valid_beta_estim), 190
odictDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
odictTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
old_to_new_log_dict (pyhrf.Verbose attribute), 1
OneTaskManager (class in pyhrf.grid), 236
onsets (pyhrf.core.FmriData attribute), 231
order1Mask (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping attribute), 15
order2Mask (pyhrf.boldsynth.spatialconfig.RegularLatticeMapping attribute), 15
OrderedDict (class in pyhrf.tools.backports), 170
output() (pyhrf.ui.treatment.FMRITreatment method), 183
outputResults() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
outputResults_back_compat() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
override_init() (pyhrf.xmliobak.xmlbase.XMLable method), 226
override_param_init() (pyhrf.xmliobak.xmlbase.XMLable2 method), 226
override_value() (pyhrf.xmliobak.xmlbase.XMLable method), 226

P

P_ACTIV_THRESH (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams attribute), 25
P_BETA (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler attribute), 31
P_COMPUTE_AH_ONLINE (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_COVAR_HACK (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_DERIV_ORDER (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_DRIFT_LFD_PARAM (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 179
P_DRIFT_LFD_TYPE (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 179
P_DT (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 179
P_DTMIN (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 179
P_DURATION (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_HAB_ALGO_PARAM (pyhrf.jde.nrl.habituation.NRLwithHabSampler attribute), 32
P_HABITS_INI (pyhrf.jde.nrl.habituation.NRLwithHabSampler attribute), 32
P_HYPER_PRIOR (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_ attribute), 25
P_LABELS_COLORS (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLS attribute), 31
P_LABELS_INI (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSample attribute), 31
P_MEAN_CA_PR_MEAN (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Ses attribute), 25
P_MEAN_CA_PR_VAR (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Ses attribute), 25
P_MEAN_CD_PR_MEAN (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler attribute), 34
P_MEAN_CD_PR_VAR (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler attribute), 35
P_NORMALISE (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_OSFMAX (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 179
P_OUTPUT_NRL (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler attribute), 23
P_OUTPUT_PMHRF (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_OUTPUT_PREFIX (pyhrf.ui.analyser_ui.FMRIAnalyser

attribute), 178
P_OUTPUT_RATIO (pyhrf.jde.nrl.habituation.NRLwithHabSampler attribute), 32
P_PRIOR_TYPE (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_RANDOM_SEED (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 179
P_REGULARIZE (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_ROI_AVERAGE (pyhrf.ui.analyser_ui.FMRIAnalyser attribute), 178
P_SAMPLE_FLAG (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_SAMPLE_FLAG (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler attribute), 109
P_SAMPLE_FLAG (pyhrf.jde.noise.NoiseARParamsSampler attribute), 122
P_SAMPLE_FLAG (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler attribute), 23
P_SAMPLE_FLAG (pyhrf.jde.nrl.bigaussian.Variance_GaussianNRL_Multi_Sess attribute), 24
P_SAMPLE_FLAG (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Sess attribute), 25
P_SAMPLE_FLAG (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler attribute), 31
P_SAMPLE_FLAG (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler attribute), 31
P_SAMPLE_HABITS (pyhrf.jde.nrl.habituation.NRLwithHabSampler attribute), 32
P_SAMPLE_LABELS (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler attribute), 32
P_SAMPLER (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 179
P_SCALE_CA_PR_ALPHA (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler attribute), 31
P_SCALE_CA_PR_BETA (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler attribute), 31
P_SHAPE_CA_PR_MEAN (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler attribute), 31
P_TRUE_HABITS (pyhrf.jde.nrl.habituation.NRLwithHabSampler attribute), 32
P_TRUE_LABELS (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler attribute), 32
P_TrueNrlFilename (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler attribute), 23
P_USE_TRUE_NRLS (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler attribute), 23
P_USE_TRUE_VALUE (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_USE_TRUE_VALUE (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler attribute), 109
P_USE_TRUE_VALUE (pyhrf.jde.noise.NoiseARParamsSampler attribute), 122
P_USE_TRUE_VALUE (pyhrf.jde.nrl.bigaussian.Variance_GaussianNRL_ attribute), 24
P_USE_TRUE_VALUE (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixturePar attribute), 25
P_VAL_INI (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_VAL_INI (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler attribute), 109
P_VAL_INI (pyhrf.jde.noise.NoiseARParamsSampler attribute), 122
P_VAL_INI (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler attribute), 24
P_VAL_INI (pyhrf.jde.nrl.bigaussian.Variance_GaussianNRL_Multi_Sess attribute), 24
P_VAL_INI (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_ attribute), 25
P_VAL_INI (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampl attribute), 31
P_VAL_INI (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler attribute), 32
P_VAR_CA_PR_ALPHA (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Ses attribute), 25
P_VAR_CA_PR_BETA (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixturePara attribute), 25
P_VAR_CD_PR_ALPHA (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSampler attribute), 35
P_VAR_CD_PR_BETA (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsS attribute), 35
P_VAR_CI_PR_ALPHA (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Ses attribute), 25
P_VAR_CI_PR_ALPHA (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParamsSampler attribute), 31
P_VAR_CI_PR_BETA (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixturePara attribute), 25
P_VAR_CI_PR_BETA (pyhrf.jde.nrl.gammagaussian.GamGaussMixturePa attribute), 31
P_VOXELWISE_OUTPUTS (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
P_ZERO_CONSTR (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 101
packHandlers() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
packHandlers() (pyhrf.xmliobak.xmlmatlab.MatlabXMLHandler method), 227
packHandlers() (pyhrf.xmliobak.xmlnumpy.NumpyXMLHandler method), 227
packSamplerInput() (pyhrf.ui.jde.JDEMCMCAnalyser

method), 180
Paradigm (class in pyhrf.paradigm), 246
ParadigmTest (class in pyhrf.test.test_paradigm), 162
ParallelTest (class in pyhrf.test.test_parallel), 162
parametersComments (pyhrf.core.FmriData attribute), 232
parametersComments (pyhrf.core.FMRISessionVolumicData attribute), 230
parametersComments (pyhrf.jde.beta.BetaSampler attribute), 80
parametersComments (pyhrf.jde.hrf.HRFSampler attribute), 88
parametersComments (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler attribute), 93
parametersComments (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSampler attribute), 92
parametersComments (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler attribute), 94
parametersComments (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSampler attribute), 99
parametersComments (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 102
parametersComments (pyhrf.jde.jde_multi_sujets.HRF_Sampler attribute), 102
parametersComments (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParams_Sampler attribute), 109
parametersComments (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler attribute), 110
parametersComments (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler attribute), 111
parametersComments (pyhrf.jde.models.BOLDGibbsSampler attribute), 116
parametersComments (pyhrf.jde.models.BOLDGibbsSampler_AR attribute), 117
parametersComments (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
parametersComments (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler attribute), 24
parametersComments (pyhrf.jde.nrl.bigaussian.NRLSampler attribute), 22
parametersComments (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParamsDrift_Sampler attribute), 26
parametersComments (pyhrf.jde.nrl.habituation.NRLwithHabSampler attribute), 33
parametersComments (pyhrf.rfir.RFIREstim attribute), 256
parametersComments (pyhrf.ui.analyser_ui.FMRIAnalyser attribute), 178
parametersComments (pyhrf.ui.glm_ui.GLMAnalyser attribute), 179
parametersComments (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 180
parametersComments (pyhrf.ui.treatment.FMRITreatment attribute), 183
parametersComments (pyhrf.ui.vb_jde_analyser.JDEVEMAnalyser attribute), 186
parametersComments (pyhrf.ui.vb_jde_analyser_asl_fast.JDEVEMAnalyse attribute), 187
parametersComments (pyhrf.ui.vb_jde_analyser_bold_fast.JDEVEMAnalys attribute), 189
parametersComments (pyhrf.xmliobak.xmlbase.XMLParamDrivenClass attribute), 226
parametersMeta (pyhrf.xmliobak.xmlbase.XMLParamDrivenClass attribute), 226
parametersToShow (pyhrf.core.FmriData attribute), 232
parametersToShow (pyhrf.jde.asl.ASLSampler attribute), 35
parametersToShow (pyhrf.jde.asl_physio.ASLPhysioSampler attribute), 42
parametersToShow (pyhrf.jde.asl_physio_1step.ASLPhysioSampler attribute), 47
parametersToShow (pyhrf.jde.asl_physio_1step_params.ASLPhysioSample attribute), 53
parametersToShow (pyhrf.jde.asl_physio_det_fwdm.ASLPhysioSampler attribute), 61
parametersToShow (pyhrf.jde.asl_physio_hierarchical.ASLPhysioSampler attribute), 68
parametersToShow (pyhrf.jde.asl_physio_joint.ASLPhysioSampler attribute), 76
parametersToShow (pyhrf.jde.beta.BetaSampler attribute), 80
parametersToShow (pyhrf.jde.hrf.HRFSampler attribute), 88
parametersToShow (pyhrf.jde.hrf.RHSampler attribute), 90
parametersToShow (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Mult attribute), 93
parametersToShow (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSam attribute), 92
parametersToShow (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler attribute), 94
parametersToShow (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSa attribute), 99
parametersToShow (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler attribute), 102
parametersToShow (pyhrf.jde.jde_multi_sujets_alpha.Alpha_hgroup_Samp attribute), 106
parametersToShow (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler attribute), 110
parametersToShow (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler attribute), 111
parametersToShow (pyhrf.jde.jde_multi_sujets_alpha.NoiseVariance_Drift_ attribute), 113
parametersToShow (pyhrf.jde.models.BOLDGibbsSampler attribute), 116
parametersToShow (pyhrf.jde.models.BOLDGibbsSampler_AR attribute), 117
parametersToShow (pyhrf.jde.models.Drift_BOLDGibbsSampler

attribute), 119
parametersToShow (pyhrf.jde.models.W_BOLDGibbsSampler attribute), 119
parametersToShow (pyhrf.jde.models.W_Drift_BOLDGibbsSampler attribute), 120
parametersToShow (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
parametersToShow (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler attribute), 24
parametersToShow (pyhrf.jde.nrl.bigaussian.NRLSampler attribute), 22
parametersToShow (pyhrf.jde.nrl.bigaussian.Variance_GaussianNRL_Multi_Sess attribute), 24
parametersToShow (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler attribute), 26
parametersToShow (pyhrf.rfir.RFIREstim attribute), 256
parametersToShow (pyhrf.ui.analyser_ui.FMRIAnalyser attribute), 178
parametersToShow (pyhrf.ui.glm_ui.GLMAnalyser attribute), 179
parametersToShow (pyhrf.ui.jde.JDEMCMCAnalyser attribute), 180
parametersToShow (pyhrf.ui.rfir_ui.RFIRAnalyser attribute), 180
parametersToShow (pyhrf.ui.treatment.FMRITreatment attribute), 183
parametersToShow (pyhrf.ui.vb_jde_analyser.JDEVEMAnalyser attribute), 186
parametersToShow (pyhrf.ui.vb_jde_analyser_asl_fast.JDEVEMAnalyser attribute), 187
parametersToShow (pyhrf.ui.vb_jde_analyser_bold_fast.JDEVEMAnalyser attribute), 189
parametersToXml() (pyhrf.xmliobak.xmlbase.XMLParamDrivenClass method), 226
PARAMS_NAMES (pyhrf.jde.asl.MixtureParamsSampler attribute), 37
PARAMS_NAMES (pyhrf.jde.asl_physio.MixtureParamsSampler attribute), 43
PARAMS_NAMES (pyhrf.jde.asl_physio_1step.MixtureParamsSampler attribute), 49
PARAMS_NAMES (pyhrf.jde.asl_physio_1step_params.MixtureParamsSampler attribute), 55
PARAMS_NAMES (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler attribute), 63
PARAMS_NAMES (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler attribute), 70
PARAMS_NAMES (pyhrf.jde.asl_physio_joint.MixtureParamsSampler attribute), 77
PARAMS_NAMES (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Multi_Sess_NRLsBar_Sampler attribute), 93
PARAMS_NAMES (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler attribute), 103
PARAMS_NAMES (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler attribute), 108
PARAMS_NAMES (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler attribute), 112
PARAMS_NAMES (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler attribute), 19
PARAMS_NAMES (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams_ attribute), 25
PARAMS_NAMES (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParam attribute), 31
PARAMS_NAMES (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsSamp attribute), 34
parcellate_balanced_vol() (in module pyhrf.parcellation), 250
parcellate_voronoi_vol() (in module pyhrf.parcellation), 250
parcellation_dist() (in module pyhrf.parcellation), 250
parcellation_for_jde() (in module pyhrf.parcellation), 250
parcellation_hemodynamics() (in module pyhrf.sandbox.parcellation), 133
parcellation_report() (in module pyhrf.parcellation), 251
parcellation_ward_spatial() (in module pyhrf.parcellation), 251
ParcellationMethodTest (class in pyhrf.test.test_parcellation), 163
ParcellationTest (class in pyhrf.validation.valid_sandbox_parcellation), 197
parcels_to_graphs() (in module pyhrf.graph), 235
parse_data_options() (in module pyhrf.ui.treatment), 184
parse_options() (in module pyhrf.grid), 238
parseXMLString() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
PartitionFunctionTest (class in pyhrf.test.jdetest), 156
PartitionFunctionTest (class in pyhrf.validation.valid_rndm_field), 197
peelVolume3D() (in module pyhrf.tools.misc), 176
PeelVolumeTest (class in pyhrf.test.toolsTest), 168
PerfBaselineSampler (class in pyhrf.jde.asl), 37
PerfBaselineSampler (class in pyhrf.jde.asl_physio), 43
PerfBaselineSampler (class in pyhrf.jde.asl_physio_1step), 49
PerfBaselineSampler (class in pyhrf.jde.asl_physio_1step_params), 55
PerfBaselineSampler (class in pyhrf.jde.asl_physio_det_fwdm), 63
PerfBaselineSampler (class in pyhrf.jde.asl_physio_hierarchical), 70
PerfBaselineSampler (class in pyhrf.jde.asl_physio_joint), 77
PerfBaseline_Multi_Sess_NRLsBar_Sampler (class in pyhrf.jde.asl), 38
PerfBaselineVarianceSampler (class in pyhrf.jde.asl_physio), 43
PerfBaselineVarianceSampler (class in pyhrf.jde.asl_physio_1step), 49
PerfBaselineVarianceSampler (class in

pyhrf.jde.asl_physio_1step_params), 56
PerfBaselineVarianceSampler (class in pyhrf.jde.asl_physio_det_fwdm), 63
PerfBaselineVarianceSampler (class in pyhrf.jde.asl_physio_hierarchical), 71
PerfBaselineVarianceSampler (class in pyhrf.jde.asl_physio_joint), 78
PerfMixtureSampler (class in pyhrf.jde.asl), 38
PerfMixtureSampler (class in pyhrf.jde.asl_physio), 44
PerfMixtureSampler (class in pyhrf.jde.asl_physio_1step), 50
PerfMixtureSampler (class in pyhrf.jde.asl_physio_1step_params), 56
PerfMixtureSampler (class in pyhrf.jde.asl_physio_det_fwdm), 64
PerfMixtureSampler (class in pyhrf.jde.asl_physio_hierarchical), 71
PerfMixtureSampler (class in pyhrf.jde.asl_physio_joint), 78
PerfResponseLevelSampler (class in pyhrf.jde.asl), 38
PerfResponseLevelSampler (class in pyhrf.jde.asl_physio), 44
PerfResponseLevelSampler (class in pyhrf.jde.asl_physio_1step), 50
PerfResponseLevelSampler (class in pyhrf.jde.asl_physio_1step_params), 56
PerfResponseLevelSampler (class in pyhrf.jde.asl_physio_det_fwdm), 64
PerfResponseLevelSampler (class in pyhrf.jde.asl_physio_hierarchical), 71
PerfResponseLevelSampler (class in pyhrf.jde.asl_physio_joint), 78
PerfResponseSampler (class in pyhrf.jde.asl), 38
PerfResponseVarianceSampler (class in pyhrf.jde.asl), 38
permutation() (in module pyhrf.jde.jde_multi_sess), 96
permutation() (in module pyhrf.jde.models), 120
permutation() (in module pyhrf.jde.nrl.bigaussian_drift), 29
permutation() (in module pyhrf.parcellation), 251
phy_integrate_euler() (in module pyhrf.sandbox.physio), 138
phy_integrate_euler() (in module pyhrf.sandbox.physio_params), 143
physio_build_jde_mcmc_sampler() (in module pyhrf.jde.asl_2steps), 40
PhysioBOLDResponseSampler (class in pyhrf.jde.asl_physio), 44
PhysioBOLDResponseSampler (class in pyhrf.jde.asl_physio_1step), 50
PhysioBOLDResponseSampler (class in pyhrf.jde.asl_physio_1step_params), 56
PhysioBOLDResponseSampler (class in pyhrf.jde.asl_physio_det_fwdm), 64
PhysioBOLDResponseSampler (class in pyhrf.jde.asl_physio_hierarchical), 71
PhysioBOLDResponseSampler (class in pyhrf.jde.asl_physio_joint), 78
PhysioBOLDResponseVarianceSampler (class in pyhrf.jde.asl_physio), 44
PhysioBOLDResponseVarianceSampler (class in pyhrf.jde.asl_physio_1step), 50
PhysioBOLDResponseVarianceSampler (class in pyhrf.jde.asl_physio_1step_params), 57
PhysioBOLDResponseVarianceSampler (class in pyhrf.jde.asl_physio_det_fwdm), 64
PhysioBOLDResponseVarianceSampler (class in pyhrf.jde.asl_physio_hierarchical), 71
PhysioJointResponseVarianceSampler (class in pyhrf.jde.asl_physio_joint), 78
PhysioPerfResponseSampler (class in pyhrf.jde.asl_physio), 44
PhysioPerfResponseSampler (class in pyhrf.jde.asl_physio_1step), 50
PhysioPerfResponseSampler (class in pyhrf.jde.asl_physio_1step_params), 58
PhysioPerfResponseSampler (class in pyhrf.jde.asl_physio_det_fwdm), 65
PhysioPerfResponseSampler (class in pyhrf.jde.asl_physio_hierarchical), 72
PhysioPerfResponseSampler (class in pyhrf.jde.asl_physio_joint), 79
PhysioPerfResponseVarianceSampler (class in pyhrf.jde.asl_physio), 45
PhysioPerfResponseVarianceSampler (class in pyhrf.jde.asl_physio_1step), 51
PhysioPerfResponseVarianceSampler (class in pyhrf.jde.asl_physio_1step_params), 58
PhysioPerfResponseVarianceSampler (class in pyhrf.jde.asl_physio_det_fwdm), 65
PhysioPerfResponseVarianceSampler (class in pyhrf.jde.asl_physio_hierarchical), 72
PhysioTrueBOLDResponseSampler (class in pyhrf.jde.asl_physio_hierarchical), 72
PhysioTrueBOLDResponseVarianceSampler (class in pyhrf.jde.asl_physio_hierarchical), 73
pickLabels() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 7
pickle_result() (pyhrf.ui.treatment.FMRITreatment method), 183
PickleableStaticMethod (class in pyhrf.tools.misc), 172
Pipeline (class in pyhrf.tools.misc), 172
PipelineTest (class in pyhrf.test.toolsTest), 168
plot_anat_parcel_func_fusion() (in module pyhrf.plot), 253
plot_calc_hrf() (in module pyhrf.sandbox.physio), 138
plot_convergence() (in module pyhrf.vbjde.vem_tools), 221

plot_cub_as_curve() (in module pyhrf.plot), 253
plot_cub_as_image() (in module pyhrf.plot), 254
plot_func_slice() (in module pyhrf.plot), 254
plot_gaussian_mixture() (in module pyhrf.plot), 254
plot_gaussian_pdf() (in module pyhrf.plot), 254
plot_palette() (in module pyhrf.plot), 254
plot_response_functions_it() (in module pyhrf.vbjde.vem_tools), 221
plot_spm_mip() (in module pyhrf.plot), 254
PlotCommandTest (class in pyhrf.test.test_plot), 163
PlotFunctionsTest (class in pyhrf.test.test_plot), 163
poly_drifts_basis() (in module pyhrf.vbjde.vem_tools), 221
polyError, 176
polyFit() (in module pyhrf.tools.misc), 176
polyFit() (in module pyhrf.vbjde.vem_tools), 221
PolyMat() (in module pyhrf.vbjde.vem_tools), 207
pop() (pyhrf.tools.backports.OrderedDict method), 170
popitem() (pyhrf.tools.backports.OrderedDict method), 170
potts_generator() (in module pyhrf.boldsynth.field), 8
PottsField (class in pyhrf.boldsynth.spatialconfig), 14
PottsTest (class in pyhrf.validation.valid_rndm_field), 197
ppm_contrasts() (in module pyhrf.vbjde.vem_tools), 221
PPMcalculus() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
PPMcalculus_jde() (in module pyhrf.tools.misc), 172
ppms_computation() (in module pyhrf.vbjde.vem_tools), 222
PPMTest (class in pyhrf.test.statsTest), 156
PR_MEAN (pyhrf.jde.jde_multi_sujets_alpha.HRFVarianceSubjectSampler attribute), 110
PR_MEAN (pyhrf.jde.jde_multi_sujets_alpha.RHGroupSampler attribute), 113
PR_VAR (pyhrf.jde.jde_multi_sujets_alpha.HRFVarianceSubjectSampler attribute), 110
PR_VAR (pyhrf.jde.jde_multi_sujets_alpha.RHGroupSampler attribute), 113
prepare_treatment_jobs() (in module pyhrf.parallel), 247
print_status() (pyhrf.grid.TasksManager method), 237
printState() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
probe() (pyhrf.grid.HostsManager method), 236
ProbeHost (class in pyhrf.grid), 236
project_fmri() (in module pyhrf.surface), 257
project_fmri_from_kernels() (in module pyhrf.surface), 257
protect_xml_attr() (in module pyhrf.xmlio), 259
ptp() (pyhrf.ndarray.xndarray method), 244
pyhrf (module), 1
pyhrf.boldsynth (module), 3
pyhrf.boldsynth.field (module), 7
pyhrf.boldsynth.hrf (module), 8
pyhrf.boldsynth.pottsfield (module), 3
pyhrf.boldsynth.pottsfield.swendsenwang (module), 3
pyhrf.boldsynth.scenarios (module), 9
pyhrf.boldsynth.spatialconfig (module), 14
pyhrf.configuration (module), 229
pyhrf.core (module), 229
pyhrf.glm (module), 234
pyhrf.graph (module), 234
pyhrf.grid (module), 236
pyhrf.jde (module), 18
pyhrf.jde.asl (module), 35
pyhrf.jde.asl_2steps (module), 40
pyhrf.jde.asl_physio (module), 41
pyhrf.jde.asl_physio_1step (module), 47
pyhrf.jde.asl_physio_1step_params (module), 53
pyhrf.jde.asl_physio_det_fwdm (module), 60
pyhrf.jde.asl_physio_hierarchical (module), 68
pyhrf.jde.asl_physio_joint (module), 75
pyhrf.jde.beta (module), 80
pyhrf.jde.drift (module), 86
pyhrf.jde.hrf (module), 87
pyhrf.jde.jde_multi_sess (module), 91
pyhrf.jde.jde_multi_sujets (module), 99
pyhrf.jde.jde_multi_sujets_alpha (module), 106
pyhrf.jde.models (module), 115
pyhrf.jde.noise (module), 122
pyhrf.jde.nrl (module), 18
pyhrf.jde.nrl.ar (module), 18
pyhrf.jde.nrl.base (module), 19
pyhrf.jde.nrl.bigaussian (module), 19
pyhrf.jde.nrl.bigaussian_drift (module), 25
pyhrf.jde.nrl.gammagaussian (module), 31
pyhrf.jde.nrl.habituation (module), 32
pyhrf.jde.nrl.trigaussian (module), 34
pyhrf.jde.samplerbase (module), 124
pyhrf.jde.sampler (module), 126
pyhrf.ndarray (module), 238
pyhrf.paradigm (module), 246
pyhrf.parallel (module), 247
pyhrf.parcellation (module), 249
pyhrf.plot (module), 253
pyhrf.rfir (module), 254
pyhrf.sandbox (module), 127
pyhrf.sandbox.data_parser (module), 127
pyhrf.sandbox.func_BMA_consensus_clustering (module), 128
pyhrf.sandbox.make_parcellation (module), 129
pyhrf.sandbox.parcellation (module), 129
pyhrf.sandbox.physio (module), 135
pyhrf.sandbox.physio_params (module), 140
pyhrf.sandbox.stats (module), 143
pyhrf.stats (module), 145
pyhrf.stats.misc (module), 145
pyhrf.stats.random (module), 146

pyhrf.surface (module), 257
pyhrf.test (module), 149
pyhrf.test.analysertest (module), 150
pyhrf.test.boldsynthTest (module), 150
pyhrf.test.commandTest (module), 152
pyhrf.test.core_test (module), 153
pyhrf.test.graphtest (module), 153
pyhrf.test.iotest (module), 153
pyhrf.test.jdetest (module), 155
pyhrf.test.rfir_test (module), 156
pyhrf.test.seppotest (module), 156
pyhrf.test.statsTest (module), 156
pyhrf.test.test (module), 156
pyhrf.test.test_glm (module), 158
pyhrf.test.test_jde_multi_subj (module), 158
pyhrf.test.test_jde_vem_asl (module), 158
pyhrf.test.test_jde_vem_bold (module), 159
pyhrf.test.test_jde_vem_tools (module), 159
pyhrf.test.test_jde_vem_tools_asl (module), 160
pyhrf.test.test_jde_vem_tools_UtilsC (module), 160
pyhrf.test.test_ndarray (module), 161
pyhrf.test.test_paradigm (module), 162
pyhrf.test.test_parallel (module), 162
pyhrf.test.test_parcellation (module), 163
pyhrf.test.test_plot (module), 163
pyhrf.test.test_rfir (module), 164
pyhrf.test.test_sampler (module), 164
pyhrf.test.test_sandbox_physio (module), 164
pyhrf.test.test_treatment (module), 165
pyhrf.test.test_xml (module), 165
pyhrf.test.toolsTest (module), 167
pyhrf.tools (module), 169
pyhrf.tools.aexpression (module), 169
pyhrf.tools.backports (module), 170
pyhrf.tools.cpus (module), 170
pyhrf.tools.message (module), 171
pyhrf.tools.misc (module), 172
pyhrf.ui (module), 178
pyhrf.ui.analyser_ui (module), 178
pyhrf.ui.glm_analyser (module), 179
pyhrf.ui.glm_ui (module), 179
pyhrf.ui.jde (module), 179
pyhrf.ui.rfir_ui (module), 180
pyhrf.ui.treatment (module), 182
pyhrf.ui.vb_jde_analyser (module), 185
pyhrf.ui.vb_jde_analyser_asl_fast (module), 187
pyhrf.ui.vb_jde_analyser_bold_fast (module), 188
pyhrf.usemode (module), 258
pyhrf.validation (module), 189
pyhrf.validation.config (module), 190
pyhrf.validation.valid_beta_estim (module), 190
pyhrf.validation.valid_jde_asl (module), 191
pyhrf.validation.valid_jde_asl_physio (module), 192
pyhrf.validation.valid_jde_asl_physio_alpha (module), 193
pyhrf.validation.valid_jde_bold_mono_subj_multi_sess (module), 194
pyhrf.validation.valid_jde_bold_mono_subj_sess (module), 195
pyhrf.validation.valid_jde_vem_asl (module), 195
pyhrf.validation.valid_rfir (module), 196
pyhrf.validation.valid_rndm_field (module), 197
pyhrf.validation.valid_sandbox_parcellation (module), 197
pyhrf.vbjde (module), 199
pyhrf.vbjde.vem_asl_models_fast (module), 199
pyhrf.vbjde.vem_asl_models_fast_ms (module), 200
pyhrf.vbjde.vem_bold (module), 201
pyhrf.vbjde.vem_bold_constrained (module), 203
pyhrf.vbjde.vem_bold_models_fast (module), 205
pyhrf.vbjde.vem_bold_models_fast_ms (module), 206
pyhrf.vbjde.vem_tools (module), 207
pyhrf.version (module), 258
pyhrf.xmlio (module), 258
pyhrf.xmliobak (module), 223
pyhrf.xmliobak.xmlbase (module), 223
pyhrf.xmliobak.xmlmatlab (module), 226
pyhrf.xmliobak.xmlnumpy (module), 227

Q

Q_Entropy() (in module pyhrf.vbjde.vem_tools), 207
Q_expectation_Ptilde() (in module pyhrf.vbjde.vem_tools), 207
quit() (in module pyhrf.grid), 238

R

rand() (in module pyhrf.jde.jde_multi_sess), 97
rand() (in module pyhrf.jde.models), 121
rand() (in module pyhrf.jde.nrl.bigaussian_drift), 29
rand() (in module pyhrf.parcellation), 251
rand() (in module pyhrf.stats.random), 147
rand() (in module pyhrf.test), 149
rand() (in module pyhrf.test.boldsynthTest), 151
rand() (in module pyhrf.test.test), 156
randint() (in module pyhrf.parcellation), 252
randn() (in module pyhrf.boldsynth.scenarios), 12
randn() (in module pyhrf.jde.asl), 39
randn() (in module pyhrf.jde.jde_multi_sess), 97
randn() (in module pyhrf.jde.jde_multi_sujets), 105
randn() (in module pyhrf.jde.jde_multi_sujets_alpha), 114
randn() (in module pyhrf.jde.models), 121
randn() (in module pyhrf.jde.nrl.bigaussian_drift), 30
randn() (in module pyhrf.stats.random), 148
randn() (in module pyhrf.test), 149
randn() (in module pyhrf.test.boldsynthTest), 151
randn() (in module pyhrf.test.test), 157

randn() (in module pyhrf.validation), 189
randn() (in module pyhrf.validation.valid_beta_estim), 190
random_field_generator (class in pyhrf.boldsynth.field), 8
random_pick() (in module pyhrf.parcellation), 253
RandomGenerator (class in pyhrf.stats.random), 147
randomize() (pyhrf.boldsynth.spatialconfig.StateField method), 16
rasterize_paradigm() (in module pyhrf.boldsynth.scenarios), 13
read_hierarchic_tasks() (in module pyhrf.grid), 238
read_hosts() (in module pyhrf.grid), 238
read_tasks() (in module pyhrf.grid), 238
read_timeslot() (in module pyhrf.grid), 238
read_xml() (in module pyhrf.xmlio), 259
read_xml() (in module pyhrf.xmliobak.xmlbase), 226
readDOMData() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
ReadPointOfInterestData() (pyhrf.rfir.RFIREstim method), 255
rebin() (in module pyhrf.tools.misc), 177
record() (pyhrf.jde.samplerbase.Trajectory method), 126
record_trajectories() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
registerNbIterations() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
RegularLatticeMapping (class in pyhrf.boldsynth.spatialconfig), 14
RegularLatticeMapping2 (class in pyhrf.boldsynth.spatialconfig), 15
regVarsInPipeline() (pyhrf.jde.samplerbase.GibbsSampler method), 124
remote_dir_is_writable() (in module pyhrf.grid), 238
remote_map() (in module pyhrf.parallel), 248
remote_map_marshal() (in module pyhrf.parallel), 249
RemoteException, 247
removeShortCircuits() (pyhrf.tools.misc.Pipeline method), 172
render_ward_tree() (in module pyhrf.sandbox.parcellation), 134
reorient() (pyhrf.ndarray.xndarray method), 244
repeat() (pyhrf.ndarray.xndarray method), 244
RepeatedTasksManager (class in pyhrf.grid), 237
replace_data_dir() (pyhrf.ui.treatment.FMRITreatment method), 183
replace_ext() (in module pyhrf.tools.misc), 177
report_arrays_in_obj() (in module pyhrf.tools.misc), 177
reportChange() (pyhrf.tools.misc.Pipeline method), 173
reportCurrentVal() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 90
reportCurrentVal() (pyhrf.jde.hrf.HRFSampler method), 88
reportCurrentVal() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
reportCurrentVal() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
reportCurrentVal() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 103
reportCurrentVal() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
reportCurrentVal() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
reportDetection() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
reprAllDeps() (pyhrf.tools.misc.Pipeline method), 173
reprDep() (pyhrf.tools.misc.Pipeline method), 173
represent_features() (in module pyhrf.sandbox.parcellation), 134
resampleSignal() (in module pyhrf.tools.misc), 177
ResampleTest (class in pyhrf.test.toolsTest), 168
resampleToGrid() (in module pyhrf.tools.misc), 177
rescale_bold_over_perf() (in module pyhrf.sandbox.physio), 139
rescale_hrf_group() (in module pyhrf.jde.jde_multi_sujets), 105
rescale_hrf_subj() (in module pyhrf.jde.jde_multi_sujets), 105
rescale_hrf_subj_var() (in module pyhrf.jde.jde_multi_sujets), 105
rescale_values() (in module pyhrf.tools.misc), 177
rescale_values() (pyhrf.ndarray.xndarray method), 244
reset() (pyhrf.sandbox.stats.GibbsSampler method), 144
reset() (pyhrf.sandbox.stats.GSVariable method), 144
resolve() (pyhrf.parcellation.World method), 250
resolve() (pyhrf.tools.misc.Pipeline method), 173
ResponseLevelSampler (class in pyhrf.jde.asl), 38
ResponseLevelSampler (class in pyhrf.jde.asl_physio), 45
ResponseLevelSampler (class in pyhrf.jde.asl_physio_1step), 51
ResponseLevelSampler (class in pyhrf.jde.asl_physio_1step_params), 59
ResponseLevelSampler (class in pyhrf.jde.asl_physio_det_fwdm), 65
ResponseLevelSampler (class in pyhrf.jde.asl_physio_hierarchical), 73
ResponseLevelSampler (class in pyhrf.jde.asl_physio_joint), 79
ResponseSampler (class in pyhrf.jde.asl), 39
ResponseSampler (class in pyhrf.jde.asl_physio), 45
ResponseSampler (class in pyhrf.jde.asl_physio_1step), 51
ResponseSampler (class in pyhrf.jde.asl_physio_1step_params), 59
ResponseSampler (class in pyhrf.jde.asl_physio_det_fwdm), 66
ResponseSampler (class in pyhrf.jde.asl_physio_hierarchical), 73
ResponseSampler (class in pyhrf.jde.asl_physio_joint),

79
ResponseVarianceSampler (class in pyhrf.jde.asl), 39
ResponseVarianceSampler (class in pyhrf.jde.asl_physio), 46
ResponseVarianceSampler (class in pyhrf.jde.asl_physio_1step), 52
ResponseVarianceSampler (class in pyhrf.jde.asl_physio_1step_params), 60
ResponseVarianceSampler (class in pyhrf.jde.asl_physio_det_fwdm), 66
restarize_events() (in module pyhrf.paradigm), 247
RF_Entropy() (in module pyhrf.vbjde.vem_tools), 208
RF_expectation_Ptilde() (in module pyhrf.vbjde.vem_tools), 208
rfir() (in module pyhrf.rfir), 257
RFIRAnalyser (class in pyhrf.ui.rfir_ui), 180
RFIREstim (class in pyhrf.rfir), 254
RFIRTest (class in pyhrf.test.test_rfir), 164
RFIRTest (class in pyhrf.validation.valid_rfir), 196
RHGroupSampler (class in pyhrf.jde.jde_multi_sujets), 104
RHGroupSampler (class in pyhrf.jde.jde_multi_sujets_alpha), 113
RHSampler (class in pyhrf.jde.hrf), 90
RL_Entropy() (in module pyhrf.vbjde.vem_tools), 208
RL_expectation_Ptilde() (in module pyhrf.vbjde.vem_tools), 208
roc_curve() (in module pyhrf.vbjde.vem_tools), 223
roi_split() (pyhrf.core.FmriData method), 232
roi_split() (pyhrf.core.FmriGroupData method), 233
roiMapped() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
roiMask (pyhrf.core.FmriData attribute), 231, 232
roll() (pyhrf.ndarray.xndarray method), 244
root3() (in module pyhrf.tools.misc), 177
rootTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
rotate() (in module pyhrf.plot), 254
round_nb_parcels() (in module pyhrf.parcellation), 253
rpnorm() (in module pyhrf.stats.random), 148
RPNormTest (class in pyhrf.test.statsTest), 156
run() (pyhrf.grid.ProbeHost method), 237
run() (pyhrf.grid.TasksStarter method), 237
run() (pyhrf.rfir.RFIREstim method), 257
run() (pyhrf.sandbox.stats.GibbsSampler method), 144
run() (pyhrf.ui.treatment.FMRITreatment method), 183
run_analysis() (in module pyhrf.ui.vb_jde_analyser), 186
run_analysis() (in module pyhrf.ui.vb_jde_analyser_asl_fast), 188
run_analysis() (in module pyhrf.ui.vb_jde_analyser_bold_fast), 189
run_calc_linear_rfs() (in module pyhrf.sandbox.physio), 139
run_grid() (in module pyhrf.grid), 238
run_pyhrf_cmd_treatment() (in module pyhrf.ui.treatment), 184
run_soma_workflow() (in module pyhrf.parallel), 249
runEstimationBetaEstim() (in module pyhrf.ui.jde), 180
runEstimationSupervised() (in module pyhrf.ui.jde), 180
runSampling() (pyhrf.jde.samplerbase.GibbsSampler method), 124
RxCopyTest (class in pyhrf.test.iotest), 154

S

safe_init() (in module pyhrf.sandbox.data_parser), 128
safe_list() (in module pyhrf.sandbox.data_parser), 128
same() (in module pyhrf.sandbox.data_parser), 128
sample() (pyhrf.sandbox.stats.GSVariable method), 144
sampleDrift() (in module pyhrf.jde.drift), 87
sampleHRF_single_hrf() (in module pyhrf.jde.hrf), 91
sampleHRF_single_hrf() (in module pyhrf.jde.jde_multi_sess), 98
sampleHRF_single_hrf() (in module pyhrf.jde.jde_multi_sujets), 106
sampleHRF_single_hrf() (in module pyhrf.jde.jde_multi_sujets_alpha), 114
sampleHRF_single_hrf_hack() (in module pyhrf.jde.hrf), 91
sampleHRF_single_hrf_hack() (in module pyhrf.jde.jde_multi_sess), 98
sampleHRF_single_hrf_hack() (in module pyhrf.jde.jde_multi_sujets), 106
sampleHRF_single_hrf_hack() (in module pyhrf.jde.jde_multi_sujets_alpha), 114
sampleHRF_voxelwise_iid() (in module pyhrf.jde.hrf), 91
sampleHRF_voxelwise_iid() (in module pyhrf.jde.jde_multi_sess), 98
sampleHRF_voxelwise_iid() (in module pyhrf.jde.jde_multi_sujets), 106
sampleHRF_voxelwise_iid() (in module pyhrf.jde.jde_multi_sujets_alpha), 114
sampleLabels() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
sampleLabels() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
sampleLabels() (pyhrf.jde.nrl.trigaussian.GGGNRLSampler method), 34
sampleLabelsWithRelVar() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
sampleNextAlt() (pyhrf.jde.drift.DriftARSampler method), 86
sampleNextAlt() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 90
sampleNextAlt() (pyhrf.jde.hrf.HRFSampler method), 88
sampleNextAlt() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93

sampleNextAlt() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
sampleNextAlt() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
sampleNextAlt() (pyhrf.jde.jde_multi_sess.NRLsBar_Drift_MultiSess_Sampler method), 96
sampleNextAlt() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
sampleNextAlt() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
sampleNextAlt() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 103
sampleNextAlt() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler method), 109
sampleNextAlt() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
sampleNextAlt() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
sampleNextAlt() (pyhrf.jde.nrl.ar.NRLARSampler method), 18
sampleNextAlt() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
sampleNextAlt() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
sampleNextAlt() (pyhrf.jde.nrl.bigaussian_drift.NRLsBar_Drift_MultiSess_Sampler method), 29
sampleNextAlt() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
sampleNextAlt() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
sampleNextAlt() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
sampleNextInternal() (pyhrf.jde.asl.DriftCoeffSampler method), 36
sampleNextInternal() (pyhrf.jde.asl.DriftVarianceSampler method), 36
sampleNextInternal() (pyhrf.jde.asl.LabelSampler method), 37
sampleNextInternal() (pyhrf.jde.asl.MixtureParamsSampler method), 37
sampleNextInternal() (pyhrf.jde.asl.NoiseVarianceSampler method), 37
sampleNextInternal() (pyhrf.jde.asl.PerfBaselineSampler method), 38
sampleNextInternal() (pyhrf.jde.asl.PerfBaselineVarianceSampler method), 38
sampleNextInternal() (pyhrf.jde.asl.ResponseLevelSampler method), 39
sampleNextInternal() (pyhrf.jde.asl.ResponseSampler method), 39
sampleNextInternal() (pyhrf.jde.asl.ResponseVarianceSampler method), 39
sampleNextInternal() (pyhrf.jde.asl_physio.DriftCoeffSampler method), 42
sampleNextInternal() (pyhrf.jde.asl_physio.DriftVarianceSampler method), 42
sampleNextInternal() (pyhrf.jde.asl_physio.LabelSampler method), 43
sampleNextInternal() (pyhrf.jde.asl_physio.MixtureParamsSampler method), 43
sampleNextInternal() (pyhrf.jde.asl_physio.NoiseVarianceSampler method), 43
sampleNextInternal() (pyhrf.jde.asl_physio.PerfBaselineSampler method), 43
sampleNextInternal() (pyhrf.jde.asl_physio.PerfBaselineVarianceSampler method), 44
sampleNextInternal() (pyhrf.jde.asl_physio.PhysioBOLDResponseSampler method), 44
sampleNextInternal() (pyhrf.jde.asl_physio.PhysioPerfResponseSampler method), 45
sampleNextInternal() (pyhrf.jde.asl_physio.ResponseLevelSampler method), 45
sampleNextInternal() (pyhrf.jde.asl_physio.ResponseSampler method), 46
sampleNextInternal() (pyhrf.jde.asl_physio.ResponseVarianceSampler method), 46
sampleNextInternal() (pyhrf.jde.asl_physio_1step.DriftCoeffSampler method), 48
sampleNextInternal() (pyhrf.jde.asl_physio_1step.DriftVarianceSampler method), 48
sampleNextInternal() (pyhrf.jde.asl_physio_1step.LabelSampler method), 48
sampleNextInternal() (pyhrf.jde.asl_physio_1step.MixtureParamsSampler method), 49
sampleNextInternal() (pyhrf.jde.asl_physio_1step.NoiseVarianceSampler method), 49
sampleNextInternal() (pyhrf.jde.asl_physio_1step.PerfBaselineSampler method), 49
sampleNextInternal() (pyhrf.jde.asl_physio_1step.PerfBaselineVarianceSam method), 50
sampleNextInternal() (pyhrf.jde.asl_physio_1step.PhysioBOLDResponseSa method), 50
sampleNextInternal() (pyhrf.jde.asl_physio_1step.PhysioPerfResponseSamp method), 51
sampleNextInternal() (pyhrf.jde.asl_physio_1step.ResponseLevelSampler method), 51
sampleNextInternal() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 52
sampleNextInternal() (pyhrf.jde.asl_physio_1step.ResponseVarianceSample method), 52
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.DriftCoeffSampl method), 54
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.DriftVarianceSar method), 54
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.LabelSampler method), 55
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.MixtureParamsS method), 55

sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.NoiseVarianceSampler method), 55
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.PerfBaselineSampler method), 55
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.PerfBaselineVarianceSampler method), 56
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.PhysioBOLDResponseSampler method), 57
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.PhysioPerfResponseSampler method), 58
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler method), 59
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
sampleNextInternal() (pyhrf.jde.asl_physio_1step_params.ResponseVarianceSampler method), 60
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.DriftCoeffSampler method), 62
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.DriftVarianceSampler method), 62
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.LabelSampler method), 62
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.MixtureParamsSampler method), 63
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.NoiseVarianceSampler method), 63
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.PerfBaselineSampler method), 63
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.PerfBaselineVarianceSampler method), 64
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResponseSampler method), 64
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResponseVarianceSampler method), 64
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.PhysioPerfResponseSampler method), 65
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.PhysioPerfResponseVarianceSampler method), 65
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.ResponseLevelSampler method), 66
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
sampleNextInternal() (pyhrf.jde.asl_physio_det_fwdm.ResponseVarianceSampler method), 66
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.DriftCoeffSampler method), 69
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.DriftVarianceSampler method), 69
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.LabelSampler method), 70
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.MixtureParamsSampler method), 70
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.NoiseVarianceSampler method), 70
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.PerfBaselineSampler method), 71
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.PerfBaselineVarianceSampler method), 71
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.PhysioBOLDResponseSampler method), 71
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.PhysioBOLDResponseVarianceSampler method), 72
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseSampler method), 72
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseVarianceSampler method), 72
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.PhysioTrueBOLDResponseSampler method), 73
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.PhysioTrueBOLDResponseVarianceSampler method), 73
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.ResponseLevelSampler method), 73
sampleNextInternal() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
sampleNextInternal() (pyhrf.jde.asl_physio_joint.DriftCoeffSampler method), 76
sampleNextInternal() (pyhrf.jde.asl_physio_joint.DriftVarianceSampler method), 76
sampleNextInternal() (pyhrf.jde.asl_physio_joint.LabelSampler method), 77
sampleNextInternal() (pyhrf.jde.asl_physio_joint.MixtureParamsSampler method), 77
sampleNextInternal() (pyhrf.jde.asl_physio_joint.NoiseVarianceSampler method), 77
sampleNextInternal() (pyhrf.jde.asl_physio_joint.PerfBaselineSampler method), 77
sampleNextInternal() (pyhrf.jde.asl_physio_joint.PerfBaselineVarianceSampler method), 78
sampleNextInternal() (pyhrf.jde.asl_physio_joint.PhysioBOLDResponseSampler method), 78
sampleNextInternal() (pyhrf.jde.asl_physio_joint.PhysioJointResponseVarianceSampler method), 78
sampleNextInternal() (pyhrf.jde.asl_physio_joint.PhysioPerfResponseSampler method), 79
sampleNextInternal() (pyhrf.jde.asl_physio_joint.ResponseLevelSampler method), 79
sampleNextInternal() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
sampleNextInternal() (pyhrf.jde.beta.BetaSampler method), 80
sampleNextInternal() (pyhrf.jde.drift.DriftARSampler method), 86
sampleNextInternal() (pyhrf.jde.drift.DriftSampler method), 86
sampleNextInternal() (pyhrf.jde.drift.DriftSamplerWithRelVar method), 87
sampleNextInternal() (pyhrf.jde.drift.ETASampler method), 87

sampleNextInternal() (pyhrf.jde.drift.ETASampler_MultiSess method), 87
sampleNextInternal() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 90
sampleNextInternal() (pyhrf.jde.hrf.HRFARSampler method), 87
sampleNextInternal() (pyhrf.jde.hrf.HRFSampler method), 88
sampleNextInternal() (pyhrf.jde.hrf.HRFSamplerWithRelVar method), 89
sampleNextInternal() (pyhrf.jde.hrf.HRFwithHabSampler method), 90
sampleNextInternal() (pyhrf.jde.hrf.RHSampler method), 90
sampleNextInternal() (pyhrf.jde.hrf.ScaleSampler method), 90
sampleNextInternal() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParamsSampler_MultiSess_NRLsBar_Sampler method), 93
sampleNextInternal() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93
sampleNextInternal() (pyhrf.jde.jde_multi_sess.ETASampler_MultiSess method), 93
sampleNextInternal() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
sampleNextInternal() (pyhrf.jde.jde_multi_sess.NoiseVariance_Drift_MultiSess_Sampler method), 96
sampleNextInternal() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
sampleNextInternal() (pyhrf.jde.jde_multi_sess.NRLsBar_Drift_MultiSess_Sampler method), 96
sampleNextInternal() (pyhrf.jde.jde_multi_sess.Variance_GaussianNRL_MultiSess_Sampler method), 96
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.ETASampler_MultiSubj method), 101
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 103
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.HRFVarianceSubjectSampler method), 101
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.LabelSampler method), 103
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.MixtureParamsSampler method), 104
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.NoiseVariance_Drift_MultiSubj_Sampler method), 104
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.NRLs_Sampler method), 104
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.RHGroupSampler method), 104
sampleNextInternal() (pyhrf.jde.jde_multi_sujets.Variance_GaussianNRL_MultiSubj_Sampler method), 104
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.Alpha_hgroup_Sampler method), 106
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.AlphaVar_Sampler method), 106
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixtureParamsSampler_MultiSess_NRLsBar_Sampler method), 109
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler method), 109
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.ETASampler_MultiSubj method), 109
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 110
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.HRFVarianceSubjectSampler method), 110
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler method), 112
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.MixtureParamsSampler method), 112
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.NoiseVariance_Drift_MultiSubj_Sampler method), 113
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler method), 113
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.RHGroupSampler method), 113
sampleNextInternal() (pyhrf.jde.jde_multi_sujets_alpha.Variance_GaussianNRL_MultiSubj_Sampler method), 113
sampleNextInternal() (pyhrf.jde.noise.NoiseARParamsSampler method), 123
sampleNextInternal() (pyhrf.jde.noise.NoiseVariance_Drift_Sampler method), 123
sampleNextInternal() (pyhrf.jde.noise.NoiseVarianceARSampler method), 123
sampleNextInternal() (pyhrf.jde.noise.NoiseVarianceSampler method), 123
sampleNextInternal() (pyhrf.jde.noise.NoiseVarianceSamplerWithRelVar method), 123
sampleNextInternal() (pyhrf.jde.noise.NoiseVariancewithHabSampler method), 123
sampleNextInternal() (pyhrf.jde.nrl.ar.NRLARSampler method), 18
sampleNextInternal() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 19
sampleNextInternal() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 20
sampleNextInternal() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSampler method), 20
sampleNextInternal() (pyhrf.jde.nrl.bigaussian.MixtureWeightsSampler method), 20
sampleNextInternal() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
sampleNextInternal() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22

sampleNextInternal() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
sampleNextInternal() (pyhrf.jde.nrl.bigaussian.Variance_GaussianNRLW_Multi_Sess method), 24
sampleNextInternal() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParamsMultiSessNRLRespLevel method), 26
sampleNextInternal() (pyhrf.jde.nrl.bigaussian_drift.NRL_Drift_Sampler method), 26
sampleNextInternal() (pyhrf.jde.nrl.bigaussian_drift.NRL_Drift_SamplerWithRelVar method), 27
sampleNextInternal() (pyhrf.jde.nrl.bigaussian_drift.NRLsBar_Drift_Multi_Sess method), 29
sampleNextInternal() (pyhrf.jde.nrl.gammagaussian.GamGaussMixtureParams method), 31
sampleNextInternal() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSampler method), 32
sampleNextInternal() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
sampleNextInternal() (pyhrf.jde.nrl.trigaussian.TriGaussMixtureParamsMultiSess method), 35
sampleNextInternal() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
sampleNextInternal() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
sampleNextInternal() (pyhrf.jde.wsampler.WSampler method), 127
sampleNextInternal_bak() (pyhrf.jde.noise.NoiseVarianceSampler method), 123
sampleNrlsParallel() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
sampleNrlsParallel() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
sampleNrlsParallelWithRelVar() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
sampleNrlsSerial() (pyhrf.jde.jde_multi_sess.NRLsBar_Drift_Multi_Sess method), 96
sampleNrlsSerial() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
sampleNrlsSerial() (pyhrf.jde.nrl.bigaussian_drift.NRL_Drift_Sampler method), 26
sampleNrlsSerial() (pyhrf.jde.nrl.bigaussian_drift.NRLsBar_Drift_Multi_Sess method), 29
sampleNrlsSerial() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
sampleNrlsSerial_bak() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
sampleNrlsSerialWithRelVar() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
sampleNrlsSerialWithRelVar() (pyhrf.jde.nrl.bigaussian_drift.NRL_Drift_SamplerWithRelVar method), 27
samplingWarmUp() (pyhrf.jde.asl.BOLDResponseLevelSampler method), 36
samplingWarmUp() (pyhrf.jde.asl.LabelSampler method), 37
samplingWarmUp() (pyhrf.jde.asl.PhysioBOLDResponseSampler method), 39
samplingWarmUp() (pyhrf.jde.asl_physio.BOLDResponseLevelSampler method), 42
samplingWarmUp() (pyhrf.jde.asl_physio.DriftCoeffSampler method), 42
samplingWarmUp() (pyhrf.jde.asl_physio.LabelSampler method), 43
samplingWarmUp() (pyhrf.jde.asl_physio.PhysioBOLDResponseSampler method), 44
samplingWarmUp() (pyhrf.jde.asl_physio.PhysioPerfResponseSampler method), 45
samplingWarmUp() (pyhrf.jde.asl_physio.ResponseLevelSampler method), 45
samplingWarmUp() (pyhrf.jde.asl_physio_1step.BOLDResponseLevelSampler method), 48
samplingWarmUp() (pyhrf.jde.asl_physio_1step.DriftCoeffSampler method), 48
samplingWarmUp() (pyhrf.jde.asl_physio_1step.LabelSampler method), 48
samplingWarmUp() (pyhrf.jde.asl_physio_1step.PhysioBOLDResponseSampler method), 50
samplingWarmUp() (pyhrf.jde.asl_physio_1step.PhysioPerfResponseSampler method), 51
samplingWarmUp() (pyhrf.jde.asl_physio_1step.ResponseLevelSampler method), 51
samplingWarmUp() (pyhrf.jde.asl_physio_1step_params.BOLDResponseLevelSampler method), 54
samplingWarmUp() (pyhrf.jde.asl_physio_1step_params.DriftCoeffSampler method), 54
samplingWarmUp() (pyhrf.jde.asl_physio_1step_params.LabelSampler method), 55
samplingWarmUp() (pyhrf.jde.asl_physio_1step_params.PhysioBOLDResponseSampler method), 57
samplingWarmUp() (pyhrf.jde.asl_physio_1step_params.PhysioPerfResponseSampler method), 58
samplingWarmUp() (pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler method), 59
samplingWarmUp() (pyhrf.jde.asl_physio_det_fwdm.BOLDResponseLevelSampler method), 62
samplingWarmUp() (pyhrf.jde.asl_physio_det_fwdm.LabelSampler method), 62
samplingWarmUp() (pyhrf.jde.asl_physio_det_fwdm.PhysioBOLDResponseSampler method), 64
samplingWarmUp() (pyhrf.jde.asl_physio_det_fwdm.PhysioPerfResponseSampler method), 65
samplingWarmUp() (pyhrf.jde.asl_physio_det_fwdm.PhysioPerfResponseVarSampler method), 65
samplingWarmUp() (pyhrf.jde.asl_physio_det_fwdm.ResponseLevelSampler method), 66

samplingWarmUp() (pyhrf.jde.asl_physio_hierarchical.BOLDResponseLevelSampler method), 69
samplingWarmUp() (pyhrf.jde.asl_physio_hierarchical.DriftCoeffSampler method), 69
samplingWarmUp() (pyhrf.jde.asl_physio_hierarchical.LabelSampler method), 70
samplingWarmUp() (pyhrf.jde.asl_physio_hierarchical.PhysioBOLDResponseSampler method), 71
samplingWarmUp() (pyhrf.jde.asl_physio_hierarchical.PhysioPerfResponseSampler method), 72
samplingWarmUp() (pyhrf.jde.asl_physio_hierarchical.ResponseLevelSampler method), 73
samplingWarmUp() (pyhrf.jde.asl_physio_joint.BOLDResponseLevelSampler method), 76
samplingWarmUp() (pyhrf.jde.asl_physio_joint.DriftCoeffSampler method), 76
samplingWarmUp() (pyhrf.jde.asl_physio_joint.LabelSampler method), 77
samplingWarmUp() (pyhrf.jde.asl_physio_joint.PhysioBOLDResponseSampler method), 78
samplingWarmUp() (pyhrf.jde.asl_physio_joint.PhysioPerfResponseSampler method), 79
samplingWarmUp() (pyhrf.jde.asl_physio_joint.ResponseLevelSampler method), 79
samplingWarmUp() (pyhrf.jde.beta.BetaSampler method), 80
samplingWarmUp() (pyhrf.jde.drift.DriftARSampler method), 86
samplingWarmUp() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 90
samplingWarmUp() (pyhrf.jde.hrf.HRFSampler method), 88
samplingWarmUp() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
samplingWarmUp() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
samplingWarmUp() (pyhrf.jde.jde_multi_sess.NRLsBar_Drift_Multi_Sess_Sampler method), 96
samplingWarmUp() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
samplingWarmUp() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 103
samplingWarmUp() (pyhrf.jde.jde_multi_sujets.LabelSampler method), 103
samplingWarmUp() (pyhrf.jde.jde_multi_sujets.NRLs_Sampler method), 104
samplingWarmUp() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 111
samplingWarmUp() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
samplingWarmUp() (pyhrf.jde.jde_multi_sujets_alpha.LabelSampler method), 112
samplingWarmUp() (pyhrf.jde.jde_multi_sujets_alpha.NRLs_Sampler method), 113
samplingWarmUp() (pyhrf.jde.nrl.ar.NRLARSampler method), 18
samplingWarmUp() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
samplingWarmUp() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
samplingWarmUp() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
samplingWarmUp() (pyhrf.jde.nrl.bigaussian_drift.NRLsBar_Drift_Multi_S method), 29
samplingWarmUp() (pyhrf.jde.nrl.gammagaussian.InhomogeneousNRLSan method), 32
samplingWarmUp() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
samplingWarmUp() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 125
save() (pyhrf.core.FmriData method), 232
save() (pyhrf.ndarray.xndarray method), 244
save() (pyhrf.paradigm.Paradigm method), 247
save_graph_plot() (pyhrf.tools.misc.Pipeline method), 178
save_parcellation_outputs() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 197
save_simulation() (in module pyhrf.boldsynth.scenarios), 13
save_spm_mat_for_1st_level_glm() (pyhrf.paradigm.Paradigm method), 247
save_treatment() (in module pyhrf.parallel), 249
saveCurrentValue() (pyhrf.jde.beta.BetaSampler method), 80
saveCurrentValue() (pyhrf.jde.jde_multi_sess.NRL_Multi_Sess_Sampler method), 95
saveCurrentValue() (pyhrf.jde.nrl.bigaussian.NRL_Multi_Sess_Sampler method), 24
saveCurrentValue() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
saveCurrentValue() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
saveCurrentValue() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 126
saveCurrentValue() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
saveCurrentValue() (pyhrf.jde.wsampler.WSampler method), 127
saveGlobalObservables() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSampler method), 92
saveGlobalObservables() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSampler method), 100
saveGlobalObservables() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_SubjSamp method), 108

saveGlobalObservables() (pyhrf.jde.models.BOLDGibbsSampler method), 116
saveGlobalObservables() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
saveGlobalObservables() (pyhrf.jde.samplerbase.GibbsSampler method), 124
saveObservables() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
saveObservables() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 126
saveObservables() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
saveObservables() (pyhrf.jde.wsampler.WSampler method), 127
ScaleSampler (class in pyhrf.jde.hrf), 90
serialize_attributes() (pyhrf.xmlio.UiNode method), 259
sessionsScans (pyhrf.core.FmriData attribute), 231
set_arg_translation() (pyhrf.xmlio.Initable method), 258
set_attribute() (pyhrf.xmlio.UiNode method), 259
set_axis_domain() (pyhrf.ndarray.xndarray method), 244
set_cluster_labels() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 7
set_extra_data() (pyhrf.core.FmriData method), 232
set_gzip_outputs() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
set_init() (pyhrf.xmlio.Initable method), 258
set_init() (pyhrf.xmliobak.xmlbase.XMLable method), 226
set_init() (pyhrf.xmliobak.xmlbase.XMLable2 method), 226
set_init_param() (pyhrf.xmlio.Initable method), 258
set_init_param() (pyhrf.xmliobak.xmlbase.XMLable2 method), 226
set_init_value() (pyhrf.sandbox.stats.GSVariable method), 144
set_initialization() (pyhrf.sandbox.stats.GibbsSampler method), 144
set_initialization() (pyhrf.sandbox.stats.GSVariable method), 144
set_int_tick_labels() (in module pyhrf.plot), 254
set_label() (pyhrf.xmlio.UiNode method), 259
set_leaf() (in module pyhrf.tools.misc), 177
set_MRI_orientation() (pyhrf.ndarray.xndarray method), 244
set_nb_iterations() (pyhrf.jde.samplerbase.GibbsSampler method), 124
set_orientation() (pyhrf.ndarray.xndarray method), 245
set_outputs() (pyhrf.sandbox.stats.GSVariable method), 144
set_pass_errors() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
set_root() (pyhrf.sandbox.data_parser.StructuredDataParser method), 128
set_ticks_fontsize() (in module pyhrf.plot), 254
set_true_value() (pyhrf.sandbox.stats.GibbsSampler method), 144
set_true_value() (pyhrf.sandbox.stats.GSVariable method), 144
set_true_values() (pyhrf.sandbox.stats.GibbsSampler method), 145
set_variable() (pyhrf.sandbox.stats.GibbsSampler method), 145
set_variables() (pyhrf.sandbox.stats.GibbsSampler method), 145
set_xticklabels() (in module pyhrf.plot), 254
setdefault() (pyhrf.tools.backports.OrderedDict method), 170
setDepths() (pyhrf.tools.misc.Pipeline method), 173
setDummyInputData() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
setFieldValues() (pyhrf.boldsynth.spatialconfig.StateField method), 16
setFieldValues0() (pyhrf.boldsynth.spatialconfig.StateField method), 16
setFinalValue() (pyhrf.jde.asl.ResponseSampler method), 39
setFinalValue() (pyhrf.jde.asl_physio.ResponseLevelSampler method), 45
setFinalValue() (pyhrf.jde.asl_physio.ResponseSampler method), 46
setFinalValue() (pyhrf.jde.asl_physio_1step.ResponseLevelSampler method), 51
setFinalValue() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 52
setFinalValue() (pyhrf.jde.asl_physio_1step_params.ResponseLevelSampler method), 59
setFinalValue() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
setFinalValue() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
setFinalValue() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
setFinalValue() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
setFinalValue() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 90
setFinalValue() (pyhrf.jde.hrf.HRFSampler method), 88
setFinalValue() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
setFinalValue() (pyhrf.jde.jde_multi_sess.NRLsBar_Drift_Multi_Sess_Sam method), 96
setFinalValue() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
setFinalValue() (pyhrf.jde.jde_multi_sujets.HRF_Sampler

method), 103
setFinalValue() (pyhrf.jde.jde_multi_sujets_alpha.HRF_GroupSampler method), 111
setFinalValue() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
setFinalValue() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 126
setLFDMat() (pyhrf.jde.jde_multi_sess.BOLDSampler_MultiSess method), 92
setLFDMat() (pyhrf.jde.jde_multi_sujets.BOLDSampler_MultiSubj method), 100
setLFDMat() (pyhrf.jde.jde_multi_sujets_alpha.BOLDSampler_MultiSubj method), 108
setLFDMat() (pyhrf.jde.models.BOLDSampler_Multi_Sess method), 118
setLFDMat() (pyhrf.jde.models.BOLDSamplerInput method), 118
setSamplerEngine() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 126
setSimulationData() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
setUp() (pyhrf.test.commandTest.MiscCommandTest method), 152
setUp() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
setUp() (pyhrf.test.graphtest.GraphTest method), 153
setUp() (pyhrf.test.iotest.FileHandlingTest method), 154
setUp() (pyhrf.test.iotest.GiftiTest method), 154
setUp() (pyhrf.test.iotest.NiftiTest method), 154
setUp() (pyhrf.test.iotest.RxCopyTest method), 154
setUp() (pyhrf.test.iotest.SPMIOTest method), 155
setUp() (pyhrf.test.iotest.xndarrayIOTest method), 155
setUp() (pyhrf.test.jdetest.ASLPhysioTest method), 155
setUp() (pyhrf.test.jdetest.ASLTest method), 155
setUp() (pyhrf.test.jdetest.JDETest method), 155
setUp() (pyhrf.test.jdetest.MultiSessTest method), 155
setUp() (pyhrf.test.jdetest.PartitionFunctionTest method), 156
setUp() (pyhrf.test.statsTest.PPMTest method), 156
setUp() (pyhrf.test.test_glm.NipyGLMTest method), 158
setUp() (pyhrf.test.test_jde_multi_subj.MultiSubjTest method), 158
setUp() (pyhrf.test.test_jde_vem_asl.VEMASLTest method), 158
setUp() (pyhrf.test.test_jde_vem_bold.VEMBOLDTest method), 159
setUp() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
setUp() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
setUp() (pyhrf.test.test_jde_vem_tools_UtilsC.VEMToolsTest method), 160
setUp() (pyhrf.test.test_ndarray.TestHtml method), 161
setUp() (pyhrf.test.test_ndarray.xndarrayTest method), 161
setUp() (pyhrf.test.test_paradigm.ParadigmTest method), 162
setUp() (pyhrf.test.test_parcellation.CmdParcellationTest method), 163
setUp() (pyhrf.test.test_parcellation.MeasureTest method), 163
setUp() (pyhrf.test.test_parcellation.ParcellationMethodTest method), 163
setUp() (pyhrf.test.test_plot.PlotCommandTest method), 163
setUp() (pyhrf.test.test_plot.PlotFunctionsTest method), 163
setUp() (pyhrf.test.test_rfir.RFIRTest method), 164
setUp() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
setUp() (pyhrf.test.test_treatment.CmdInputTest method), 165
setUp() (pyhrf.test.test_treatment.TreatmentTest method), 165
setUp() (pyhrf.test.test_xml.TestXML method), 166
setUp() (pyhrf.test.toolsTest.CachedEvalTest method), 167
setUp() (pyhrf.test.toolsTest.PipelineTest method), 168
setUp() (pyhrf.test.toolsTest.TableStringTest method), 168
setUp() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
setUp() (pyhrf.validation.valid_jde_asl.ASLTest method), 191
setUp() (pyhrf.validation.valid_jde_asl_physio.ASLPhysioHierarchicalTest method), 192
setUp() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 192
setUp() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 193
setUp() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSessT method), 194
setUp() (pyhrf.validation.valid_jde_bold_mono_subj_sess.JDETest method), 195
setUp() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 195
setUp() (pyhrf.validation.valid_rfir.RFIRTest method), 196
setUp() (pyhrf.validation.valid_rndm_field.PartitionFunctionTest method), 197
setUp() (pyhrf.validation.valid_rndm_field.PottsTest method), 197
setUp() (pyhrf.validation.valid_sandbox_parcellation.FeatureExtractionTest method), 197
setUp() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 197
setUp() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 198

setupGamma() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
setupTimeNrls() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
setVariable() (pyhrf.tools.aexpression.ArithmeticExpression method), 169
simulate_asl() (in module pyhrf.jde.asl), 40
simulate_asl_full_physio() (in module pyhrf.sandbox.physio), 139
simulate_asl_phylin_prf() (in module pyhrf.sandbox.physio), 139
simulate_asl_physio_rfs() (in module pyhrf.sandbox.physio), 139
simulate_bold() (in module pyhrf.jde.models), 122
simulate_fmri_data() (in module pyhrf.validation.valid_sandbox_parcellation), 199
simulate_sessions() (in module pyhrf.jde.jde_multi_sess), 98
simulate_single_session() (in module pyhrf.jde.jde_multi_sess), 98
simulate_single_subject() (in module pyhrf.jde.jde_multi_sujets), 106
simulate_single_subject() (in module pyhrf.jde.jde_multi_sujets_alpha), 114
simulate_subjects() (in module pyhrf.jde.jde_multi_sujets), 106
simulate_subjects() (in module pyhrf.jde.jde_multi_sujets_alpha), 115
simulate_subjects() (in module pyhrf.test.test_jde_multi_subj), 158
simulation (pyhrf.core.FmriData attribute), 231
simulation_save_vol_outputs() (in module pyhrf.boldsynth.scenarios), 13
SimulationTest (class in pyhrf.test.test_sandbox_physio), 164
site_taken() (pyhrf.parcellation.World method), 250
slow_func() (in module pyhrf.test.toolsTest), 169
sparsedot() (in module pyhrf.jde.nrl.habituation), 33
sparsedotdimun() (in module pyhrf.jde.nrl.habituation), 33
spatial_ward() (in module pyhrf.sandbox.parcellation), 134
spatial_ward_sk() (in module pyhrf.sandbox.parcellation), 134
spatial_ward_with_uncertainty() (in module pyhrf.sandbox.parcellation), 134
SpatialMapping (class in pyhrf.boldsynth.spatialconfig), 15
SpatialMapping2 (class in pyhrf.boldsynth.spatialconfig), 16
SpatialTest (class in pyhrf.test.test_parcellation), 163
spExtract() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
split() (pyhrf.ndarray.xndarray method), 245
split() (pyhrf.ui.treatment.FMRITreatment method), 183
split_and_save() (in module pyhrf.ndarray), 239
split_big_parcels() (in module pyhrf.parcellation), 253
split_data() (pyhrf.ui.analyser_ui.FMRIAnalyser method), 178
split_mask_into_cc_iter() (in module pyhrf.graph), 235
split_parcel() (in module pyhrf.parcellation), 253
SPMIOTest (class in pyhrf.test.iotest), 154
squared_error() (in module pyhrf.sandbox.parcellation), 135
squeeze() (pyhrf.ndarray.xndarray method), 245
squeeze_all_but() (pyhrf.ndarray.xndarray method), 245
stack_cuboids() (in module pyhrf.ndarray), 239
stack_trees() (in module pyhrf.tools.misc), 177
start() (pyhrf.grid.DispatchedTasksManager method), 236
start() (pyhrf.grid.HierarchicalTasksManager method), 236
start() (pyhrf.grid.OneTaskManager method), 236
start() (pyhrf.grid.RepeatedTasksManager method), 237
StateField (class in pyhrf.boldsynth.spatialconfig), 16
StatTest (class in pyhrf.validation.valid_sandbox_parcellation), 198
std() (pyhrf.ndarray.xndarray method), 245
stimDurations (pyhrf.core.FmriData attribute), 231
stop_criterion() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSampler method), 92
stop_criterion() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSampl method), 100
stop_criterion() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_Subj method), 108
stop_criterion() (pyhrf.jde.models.BOLDGibbsSampler method), 116
stop_criterion() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
stop_criterion() (pyhrf.jde.samplerbase.GibbsSampler method), 125
stop_criterion() (pyhrf.sandbox.stats.GibbsSampler method), 145
store_mask_sparse() (pyhrf.core.FmriData method), 233
StoreRes() (pyhrf.rfir.RFIREstim method), 256
string() (pyhrf.tools.message.MessageColor class method), 171
string() (pyhrf.tools.message.MessageNoColor class method), 171
string() (pyhrf.tools.message.NoMessage method), 171
stringDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
stringTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
StructuredDataParser (class in pyhrf.sandbox.data_parser), 127
sub() (pyhrf.boldsynth.spatialconfig.NeighbourhoodSystem method), 14

sub() (pyhrf.boldsynth.spatialconfig.SpatialMapping2 method), 16
sub_cuboid() (pyhrf.ndarray.xndarray method), 245
sub_cuboid_from_slices() (pyhrf.ndarray.xndarray method), 245
sub_graph() (in module pyhrf.graph), 236
subcptGamma() (in module pyhrf.jde.nrl.habituation), 33
substract() (pyhrf.ndarray.xndarray method), 245
subtractYtildeWithRelVar() (pyhrf.jde.nrl.bigaussian.NRLSamplerWithRelVar method), 23
sum() (pyhrf.ndarray.xndarray method), 245
sum_over_neighbours() (in module pyhrf.vbjde.vem_tools), 223
swap_layers() (in module pyhrf.tools.misc), 177
swapaxes() (in module pyhrf.tools.misc), 177
swapaxes() (pyhrf.ndarray.xndarray method), 245
SwendsenWangSampler_graph() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 6
sys (pyhrf.tools.message.Message attribute), 171

T

T (class in pyhrf.test.test_xml), 166
TableStringTest (class in pyhrf.test.toolsTest), 168
Talker (class in pyhrf.parcellation), 249
Task (class in pyhrf.grid), 237
TaskHierarchical (class in pyhrf.grid), 237
TaskList (class in pyhrf.grid), 237
tasks_help() (in module pyhrf.grid), 238
TasksManager (class in pyhrf.grid), 237
TasksStarter (class in pyhrf.grid), 237
tearDown() (pyhrf.test.commandTest.MiscCommandTest method), 152
tearDown() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
tearDown() (pyhrf.test.graphtest.GraphTest method), 153
tearDown() (pyhrf.test.iotest.FileHandlingTest method), 154
tearDown() (pyhrf.test.iotest.GiftiTest method), 154
tearDown() (pyhrf.test.iotest.NiftiTest method), 154
tearDown() (pyhrf.test.iotest.RxCopyTest method), 154
tearDown() (pyhrf.test.iotest.SPMIOTest method), 155
tearDown() (pyhrf.test.iotest.xndarrayIOTest method), 155
tearDown() (pyhrf.test.jdetest.ASLPhysioTest method), 155
tearDown() (pyhrf.test.jdetest.ASLTest method), 155
tearDown() (pyhrf.test.jdetest.JDETest method), 155
tearDown() (pyhrf.test.jdetest.MultiSessTest method), 155
tearDown() (pyhrf.test.test_glm.NipyGLMTest method), 158
tearDown() (pyhrf.test.test_jde_multi_subj.MultiSubjTest method), 158
tearDown() (pyhrf.test.test_jde_vem_asl.VEMASLTest method), 158
tearDown() (pyhrf.test.test_jde_vem_bold.VEMBOLDTest method), 159
tearDown() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
tearDown() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
tearDown() (pyhrf.test.test_jde_vem_tools_UtilsC.VEMToolsTest method), 160
tearDown() (pyhrf.test.test_ndarray.TestHtml method), 161
tearDown() (pyhrf.test.test_ndarray.xndarrayTest method), 161
tearDown() (pyhrf.test.test_paradigm.ParadigmTest method), 162
tearDown() (pyhrf.test.test_parcellation.CmdParcellationTest method), 163
tearDown() (pyhrf.test.test_plot.PlotCommandTest method), 163
tearDown() (pyhrf.test.test_rfir.RFIRTest method), 164
tearDown() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
tearDown() (pyhrf.test.test_treatment.CmdInputTest method), 165
tearDown() (pyhrf.test.test_treatment.TreatmentTest method), 165
tearDown() (pyhrf.test.test_xml.TestXML method), 166
tearDown() (pyhrf.test.toolsTest.CachedEvalTest method), 167
tearDown() (pyhrf.test.toolsTest.PipelineTest method), 168
tearDown() (pyhrf.validation.valid_jde_asl.ASLTest method), 191
tearDown() (pyhrf.validation.valid_jde_asl_physio.ASLPhysioHierarchical
method), 192
tearDown() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 192
tearDown() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 193
tearDown() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiS
method), 194
tearDown() (pyhrf.validation.valid_jde_bold_mono_subj_sess.JDETest method), 195
tearDown() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 195
tearDown() (pyhrf.validation.valid_rfir.RFIRTest method), 196
tearDown() (pyhrf.validation.valid_sandbox_parcellation.FeatureExtraction
method), 197
tearDown() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 197

test1Darray() (pyhrf.test.toolsTest.TableStringTest method), 168
test2Darray() (pyhrf.test.toolsTest.TableStringTest method), 168
test2Darray_latex() (pyhrf.test.toolsTest.TableStringTest method), 168
test3D() (pyhrf.test.boldsynthTest.Mapper1DTest method), 150
test3Darray() (pyhrf.test.toolsTest.TableStringTest method), 168
test4Darray() (pyhrf.test.toolsTest.TableStringTest method), 168
test_advanced() (pyhrf.test.iotest.RxCopyTest method), 154
test_all() (pyhrf.validation.valid_jde_asl.ASLTest method), 191
test_all() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 192
test_all() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 193
test_all() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_balanced_parcellation() (pyhrf.test.test_parcellation.SpatialTest method), 163
test_basic() (pyhrf.test.iotest.RxCopyTest method), 154
test_basic() (pyhrf.test.test_sampler.TrajectoryTest method), 164
test_basic_types() (pyhrf.test.test_xml.BaseTest method), 165
test_beta() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_beta_estim_obs_fields() (in module pyhrf.validation.valid_beta_estim), 191
test_bfs() (pyhrf.test.graphtest.GraphTest method), 153
test_bijection_from_classmethod_init() (pyhrf.test.test_xml.InitableTest method), 166
test_bijection_from_init() (pyhrf.test.test_xml.InitableTest method), 166
test_bijection_from_init_no_arg() (pyhrf.test.test_xml.InitableTest method), 166
test_bmixt_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSessTest method), 194
test_bold() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_bool() (pyhrf.test.test_xml.BaseTest method), 165
test_brf() (pyhrf.validation.valid_jde_asl.ASLTest method), 191
test_brf() (pyhrf.validation.valid_jde_asl_physio.ASLPhysioHierarchicalTest method), 192
test_brf() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_brf_basic_reg() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 192
test_brf_basic_reg() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 193
test_brf_physio_det() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 192
test_brf_physio_det() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 193
test_brf_physio_nonreg() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 192
test_brf_physio_nonreg() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_brf_physio_reg() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_brf_physio_reg() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_brf_var() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_brf_var() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_brls() (pyhrf.validation.valid_jde_asl.ASLTest method), 192
test_brls() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_brls() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_brls() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_buildcfg_contrasts() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
test_buildcfg_jde_loc_vol_default() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
test_buildcfg_jde_locav_surf_default() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
test_buildcfg_jde_locav_vol_default() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
test_buildFiniteDiffMatrix() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_buildFiniteDiffMatrix() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_burnin() (pyhrf.test.test_sampler.TrajectoryTest method), 164
test_cached() (pyhrf.test.toolsTest.PipelineTest method), 168
test_callback() (pyhrf.test.iotest.RxCopyTest method), 154

test_cartesian_apply() (pyhrf.test.toolsTest.CartesianTest method), 167
test_cartesian_apply_parallel() (pyhrf.test.toolsTest.CartesianTest method), 167
test_cartesian_eval() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_classmethod_init() (pyhrf.test.test_xml.InitableTest method), 166
test_code_digest() (pyhrf.test.toolsTest.CachedEvalTest method), 167
test_combine_domains() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_command_line() (pyhrf.test.test_glm.NipyGLMTest method), 158
test_comparison() (pyhrf.validation.valid_rndm_field.PartitionFunctionTest method), 197
test_compute_mat_X2() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_compute_mat_X2() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_computeFit() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_computeFit() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_convex_hull() (pyhrf.test.toolsTest.GeometryTest method), 167
test_count_homo_cliques() (pyhrf.test.boldsynthTest.FieldFuncsTest method), 150
test_count_homo_cliques1() (pyhrf.test.boldsynthTest.FieldFuncsTest method), 150
test_count_homo_cliques2() (pyhrf.test.boldsynthTest.FieldFuncsTest method), 150
test_create_conditions() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_create_evoked_physio_signal() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
test_create_neighbours() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_create_physio_brf() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
test_create_physio_prf() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
test_create_tbg_neural_efficacies() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
test_decorator_do_if_file_exist() (pyhrf.test.toolsTest.MiscTest method), 168
test_decorator_do_if_file_exist2() (pyhrf.test.toolsTest.MiscTest method), 168
test_decorator_do_if_file_exist_force() (pyhrf.test.toolsTest.MiscTest method), 168
test_default_jde_cmd_parallel_local() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_default_jde_small_simulation() (pyhrf.test.jdetest.ASLPhysioTest method), 155
test_default_jde_small_simulation() (pyhrf.test.jdetest.ASLTest method), 155
test_default_jde_small_simulation() (pyhrf.test.jdetest.MultiSessTest method), 156
test_default_jde_small_simulation() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSes
Test method), 194
test_default_treatment() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_default_treatment_parallel_cluster() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_default_treatment_parallel_LAN() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_default_treatment_parallel_local() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_distance() (pyhrf.test.toolsTest.GeometryTest method), 167
test_drift() (pyhrf.validation.valid_jde_asl.ASLTest method), 192
test_drift() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_drift() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_drift_and_var_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSes
method), 195
test_drift_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_ses
method), 195
test_drift_var() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_drift_var() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_drift_var_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi
method), 195
test_dry() (pyhrf.test.iotest.RxCopyTest method), 154
test_duplicates_targets() (pyhrf.test.iotest.RxCopyTest method), 154
test_E_step() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 195
test_entropyA() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_entropyA() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest

method), 160
test_entropyH() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_entropyH() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_entropyZ() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_entropyZ() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_equality() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_expansion() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_expectA() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_expectA() (pyhrf.test.test_jde_vem_tools_UtilsC.VEMToolsTest method), 160
test_expectH() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_expectH() (pyhrf.test.test_jde_vem_tools_UtilsC.VEMToolsTest method), 160
test_expectZ() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_expectZ() (pyhrf.test.test_jde_vem_tools_UtilsC.VEMToolsTest method), 160
test_explode() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_extrapolation() (pyhrf.validation.valid_rndm_field.PartitionFunctionTest method), 197
test_feature_extraction() (pyhrf.validation.valid_sandbox_parcellation.FeatureExtractionTest method), 197
test_fill() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_fir_glm() (pyhrf.test.test_glm.NipyGLMTest method), 158
test_flatten_and_expand() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_free_energy() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_frmi_vol() (pyhrf.test.iotest.DataLoadTest method), 153
test_from_lattice1() (pyhrf.test.graphtest.GraphTest method), 153
test_from_lattice2() (pyhrf.test.graphtest.GraphTest method), 153
test_from_lattice_toro() (pyhrf.test.graphtest.GraphTest method), 153
test_from_lattice_toro_huge() (pyhrf.test.graphtest.GraphTest method), 153
test_from_mesh() (pyhrf.test.graphtest.GraphTest method), 153
test_from_vol_ui_default() (pyhrf.test.core_test.FMRIDataTest method), 153
test_full_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess method), 195
test_full_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_sess.JDET method), 195
test_func_default_args() (pyhrf.test.toolsTest.PipelineTest method), 168
test_generate_features() (pyhrf.validation.valid_sandbox_parcellation.Featu method), 197
test_get_leaf() (pyhrf.test.toolsTest.treeToolsTest method), 169
test_gibbs() (pyhrf.validation.valid_rndm_field.PottsTest method), 197
test_glm_contrasts() (pyhrf.test.test_glm.NipyGLMTest method), 158
test_glm_default_real_data() (pyhrf.test.test_glm.NipyGLMTest method), 158
test_glm_with_files() (pyhrf.test.test_glm.NipyGLMTest method), 158
test_gls_default() (pyhrf.test.commandTest.MiscCommandTest method), 152
test_gls_recursive() (pyhrf.test.commandTest.MiscCommandTest method), 152
test_gls_recursive_group() (pyhrf.test.commandTest.MiscCommandTest method), 152
test_gm_cdf() (pyhrf.test.statsTest.PPMTest method), 156
test_gm_sample_active() (pyhrf.test.statsTest.PPMTest method), 156
test_gm_sample_half() (pyhrf.test.statsTest.PPMTest method), 156
test_gm_sample_inactive() (pyhrf.test.statsTest.PPMTest method), 156
test_gmm_from_forged_features() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 197
test_gmm_known_alpha0() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 198
test_gmm_known_weights_difvars_noise() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 199
test_gmm_known_weights_difvars_noisea() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 199
test_gmm_known_weights_noise() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 199
test_gmm_known_weights_noisea() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 199
test_gmm_known_weights_simu_1D()

(pyhrf.validation.valid_sandbox_parcellation.StatTest method), 199
test_gmm_likelihood() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 199
test_gradient() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_graph_is_sane() (pyhrf.test.graphtest.GraphTest method), 153
test_hemodynamic_parcellation_GMM_2D_high_SNR() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 197
test_hemodynamic_parcellation_wpu_2D_high_SNR() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 197
test_hrf_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSessTest method), 195
test_hrf_var_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSessTest method), 195
test_hrf_var_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_sess.JDETest method), 195
test_hrf_var_sampler_2() (pyhrf.validation.valid_jde_bold_mono_subj_sess.JDETest method), 195
test_hrf_with_var_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_sess.JDETest method), 195
test_hrf_with_var_sampler_2() (pyhrf.validation.valid_jde_bold_mono_subj_sess.JDETest method), 195
test_informedGMM_parameters() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 199
test_init() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_init() (pyhrf.test.test_xml.InitableTest method), 166
test_intersection_matrix() (pyhrf.test.test_parcellation.MeasureTest method), 163
test_jde_estim_from_treatment_pck() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_JDEMCMCAnalyzer_Uinode_bijection() (pyhrf.test.test_xml.InitableTest method), 166
test_JDEMCMCAnalyzerXML() (pyhrf.test.test_xml.InitableTest method), 166
test_jdevemanalyser() (pyhrf.test.test_jde_vem_asl.VEMASLTest method), 158
test_jdevemanalyser() (pyhrf.test.test_jde_vem_bold.VEMBOLDTest method), 159
test_la() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_label_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSessTest method), 195
test_labels() (pyhrf.validation.valid_jde_asl.ASLTest method), 192
test_labels() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_labels() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_labels() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_list_of_int() (pyhrf.test.test_xml.BaseTest method), 165
test_list_of_misc() (pyhrf.test.test_xml.BaseTest method), 166
test_list_of_str() (pyhrf.test.test_xml.BaseTest method), 166
test_load_fmri_surf_data() (pyhrf.test.iotest.GiftiTest method), 154
test_load_regnames_SPM12() (pyhrf.test.iotest.SPMIOTest method), 155
test_load_regnames_SPM5() (pyhrf.test.iotest.SPMIOTest method), 155
test_load_regnames_SPM8() (pyhrf.test.iotest.SPMIOTest method), 155
test_matrix() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_matrix() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_max_beta() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_max_L() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_max_L() (pyhrf.test.test_jde_vem_tools_UtilsC.VEMToolsTest method), 160
test_max_mu_sigma() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_max_sigma_noise() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 160
test_max_sigma_noise() (pyhrf.test.test_jde_vem_tools_UtilsC.VEMToolsT method), 160
test_max_sigmaH() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_max_sigmaH() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_max_sigmaH_prior() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_max_sigmaH_prior() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_maximum() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 160
test_maximum() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_MC_comp_pval_MI_100x100()

(pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_MC_comp_pfmethods_ML_10x10() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_MC_comp_pfmethods_ML_30x30() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_MC_comp_pfmethods_ML_3C_10x10() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_MC_comp_pfmethods_ML_3C_20x20() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_MC_comp_pfmethods_ML_3C_30x30() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_MC_comp_pfmethods_ML_3C_50x50() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_merge() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_merge_onsets() (pyhrf.test.test_paradigm.ParadigmTest method), 162
test_missing_tags_dest_basename() (pyhrf.test.iotest.RxCopyTest method), 154
test_missing_tags_dest_folder() (pyhrf.test.iotest.RxCopyTest method), 154
test_mixtdist() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 197
test_mp() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_mu() (pyhrf.validation.valid_jde_asl_physio.ASLPhysioTest method), 192
test_mu_brf() (pyhrf.validation.valid_jde_asl_physio.ASLPhysioTest method), 192
test_mu_brf_prf() (pyhrf.validation.valid_jde_asl_physio.ASLPhysioTest method), 192
test_mu_prf() (pyhrf.validation.valid_jde_asl_physio.ASLPhysioTest method), 192
test_multiple_output_values() (pyhrf.test.toolsTest.PipelineTest method), 168
test_multisession_simu() (pyhrf.test.core_test.FMRIDataTest method), 153
test_noise_var() (pyhrf.validation.valid_jde_asl.ASLTest method), 192
test_noise_var() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_noise_var() (pyhrf.validation.valid_jde_asl_physio_alpha1.ASLTest method), 194
test_noise_var() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_noise_var_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSes method), 195
test_noise_var_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_sess.JDETest method), 195
test_norm_bc() (pyhrf.validation.valid_sandbox_parcellation.StatTest method), 199
test_normpdf() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 160
test_normpdf() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_nrl_bar_only_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSes method), 195
test_nrl_bar_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_ method), 195
test_nrl_by_session_hrf() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSes method), 195
test_nrl_by_session_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSes method), 195
test_nrl_by_session_var_sampler() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSes method), 195
test_obs_2Dfield_MAP() (pyhrf.test.analysertest.BetaEstimESTest method), 150
test_obs_3Dfield_MAP() (pyhrf.test.analysertest.BetaEstimESTest method), 150
test_obs_field_ML() (pyhrf.test.analysertest.BetaEstimESTest method), 150
test_operations() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_ordered_dict() (pyhrf.test.test_xml.BaseTest method), 166
test_paradigm_csv() (pyhrf.test.iotest.DataLoadTest method), 153
test_paradigm_csv2() (pyhrf.test.iotest.DataLoadTest method), 153
test_paradigm_csv3() (pyhrf.test.iotest.DataLoadTest method), 153
test_paradigm_csv4() (pyhrf.test.iotest.DataLoadTest method), 153
test_parallel_local() (pyhrf.test.test_treatment.TreatmentTest method), 165

test_parcellation() (pyhrf.test.jdetest.JDETest method), 155
test_parcellation_distance() (pyhrf.test.test_parcellation.MeasureTest method), 163
test_parcellation_history() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_mmp_act_level_1D() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_mmp_act_level_2D() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_spatialWard_2() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_spatialWard_400_nonoise() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_spatialWard_400_variance() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_spatialWard_5_sklearn() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_spatialWard_act_level_1D() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_spatialWard_act_level_2D() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_spatialWard_variance_1D() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcellation_spatialWard_variance_2D() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_parcels_to_graphs() (pyhrf.test.graphtest.GraphTest method), 153
test_path_sampling() (pyhrf.validation.valid_rndm_field.PartitionFunctionTest method), 197
test_perf_baseline() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_perf_baseline() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_perf_baseline_var() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_perf_baseline_var() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_perfusion() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_phy_integrate_euler() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
test_pickle_classmethod() (pyhrf.test.test_xml.InitableTest method), 166
test_pickle_treatment() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_plot() (pyhrf.test.test_ndarray.TestHtml method), 161
test_plot_cuboid1d_as_curve() (pyhrf.test.test_plot.PlotFunctionsTest method), 164
test_plot_cuboid1d_as_image() (pyhrf.test.test_plot.PlotFunctionsTest method), 164
test_plot_cuboid2d_as_image() (pyhrf.test.test_plot.PlotFunctionsTest method), 164
test_plot_cuboid_as_curve() (pyhrf.test.test_plot.PlotFunctionsTest method), 164
test_plot_func_slice_func_only() (pyhrf.test.test_plot.PlotCommandTest method), 163
test_plot_func_slice_func_only_multiple_slices() (pyhrf.test.test_plot.PlotCommandTest method), 163
test_plot_func_slice_func_roi() (pyhrf.test.test_plot.PlotCommandTest method), 163
test_plot_func_slice_func_roi_anat() (pyhrf.test.test_plot.PlotCommandTest method), 163
test_plot_func_slice_func_roi_anat_multiple_slices() (pyhrf.test.test_plot.PlotCommandTest method), 163
test_polyFit() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 160
test_polyFit() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_PolyMat() (pyhrf.test.test_jde_vem_tools.VEMToolsTest method), 159
test_PolyMat() (pyhrf.test.test_jde_vem_tools_asl.VEMToolsTest method), 160
test_potts_gibbs() (pyhrf.test.boldsynthTest.FieldFuncsTest method), 150
test_ppm_a_mcmc() (pyhrf.test.statsTest.PPMTest method), 156
test_ppm_a_norm() (pyhrf.test.statsTest.PPMTest method), 156
test_ppm_g_apost() (pyhrf.test.statsTest.PPMTest method), 156
test_ppm_g_mcmc() (pyhrf.test.statsTest.PPMTest method), 156
test_ppm_g_norm() (pyhrf.test.statsTest.PPMTest

method), 156
test_prf() (pyhrf.validation.valid_jde_asl.ASLTest method), 192
test_prf() (pyhrf.validation.valid_jde_asl_physio.ASLPhysioHierarchicalTest method), 192
test_prf() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_prf_basic_reg() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_prf_basic_reg() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_prf_physio_det() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_prf_physio_det() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_prf_physio_nonreg() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_prf_physio_nonreg() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_prf_physio_reg() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_prf_physio_reg() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_prf_var() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_prf_var() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_prls() (pyhrf.validation.valid_jde_asl.ASLTest method), 192
test_prls() (pyhrf.validation.valid_jde_asl_physio.ASLTest method), 193
test_prls() (pyhrf.validation.valid_jde_asl_physio_alpha.ASLTest method), 194
test_prls() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_process_history_extension() (pyhrf.test.iotest.NiftiTest method), 154
test_pyhrf_extract_cc_vol() (pyhrf.test.graphtest.GraphTest method), 153
test_quick() (pyhrf.test.test_jde_multi_subj.MultiSubjTest method), 158
test_read_default_real_data_tiny() (pyhrf.test.iotest.GiftiTest method), 154
test_read_tex_gii_label() (pyhrf.test.iotest.GiftiTest method), 154
test_rearrange() (pyhrf.test.toolsTest.treeToolsTest method), 169
test_remote_dir_writable() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_remote_map_local() (pyhrf.test.test_parallel.ParallelTest method), 162
test_remote_map_local_cartesian_args() (pyhrf.test.test_parallel.ParallelTest method), 162
test_remote_map_serial() (pyhrf.test.test_parallel.ParallelTest method), 162
test_render_ward_tree() (pyhrf.validation.valid_sandbox_parcellation.ParcelTest method), 198
test_replacement() (pyhrf.test.iotest.RxCopyTest method), 154
test_results_small_simulation() (pyhrf.validation.valid_rfir.RFIRTest method), 196
test_rfir_on_small_simulation() (pyhrf.test.test_rfir.RFIRTest method), 164
test_rfir_on_small_simulation() (pyhrf.validation.valid_rfir.RFIRTest method), 196
test_save_as_gii() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_save_as_nii() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_save_nii_3D() (pyhrf.test.iotest.xndarrayIOTest method), 155
test_save_nii_4D() (pyhrf.test.iotest.xndarrayIOTest method), 155
test_save_nii_multi() (pyhrf.test.iotest.xndarrayIOTest method), 155
test_set_init_param() (pyhrf.test.test_xml.XMLableTest method), 166
test_set_leaf() (pyhrf.test.toolsTest.treeToolsTest method), 169
test_set_orientation() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_sigmaG() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_sigmaH() (pyhrf.validation.valid_jde_vem_asl.ASLTest method), 196
test_simple() (pyhrf.test.toolsTest.CachedEvalTest method), 167
test_simple_args() (pyhrf.test.toolsTest.CachedEvalTest method), 167
test_simple_bijection() (pyhrf.test.test_xml.TestXML method), 166
test_simulate_asl_full_physio() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
test_simulate_asl_full_physio_outputs() (pyhrf.test.test_sandbox_physio.SimulationTest method), 164
test_simulate_asl_physio_rfs() (pyhrf.test.test_sandbox_physio.SimulationTest

method), 164
test_simulation() (pyhrf.test.jdetest.ASLTest method), 155
test_simulation() (pyhrf.validation.valid_jde_bold_mono_subj_multi_sess.MultiSessTest method), 195
test_single_Onsager_MAP() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_single_Onsager_ML() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_single_PFES_MAP() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_single_PFES_ML() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_single_PFPS_MAP() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_single_PFPS_ML() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_single_surface_PFPS_ML() (pyhrf.validation.valid_beta_estim.ObsField2DTest method), 190
test_slow_func() (pyhrf.test.toolsTest.CachedEvalTest method), 167
test_spatialward_against_modelbasedspatialward() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_spatialward_against_ward_sk() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_spatialward_from_forged_features() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_split() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_split4DVol() (pyhrf.test.iotest.FileHandlingTest method), 154
test_split_ext() (pyhrf.test.iotest.FileHandlingTest method), 154
test_split_parcel() (pyhrf.test.test_parcellation.SpatialTest method), 163
test_split_vol_cc_2D() (pyhrf.test.graphtest.GraphTest method), 153
test_split_vol_cc_3D() (pyhrf.test.graphtest.GraphTest method), 153
test_spm12_option_parse() (pyhrf.test.test_treatment.CmdInputTest method), 165
test_spm5_option_parse() (pyhrf.test.test_treatment.CmdInputTest method), 165
test_spm8_option_parse() (pyhrf.test.test_treatment.CmdInputTest method), 165
test_squeeze() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_stack() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_stack_trees() (pyhrf.test.toolsTest.treeToolsTest method), 169
test_sub_cuboid() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_sub_cuboid_with_float_domain() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_sub_graph() (pyhrf.test.graphtest.GraphTest method), 153
test_sub_treatment() (pyhrf.test.test_treatment.TreatmentTest method), 165
test_suite() (in module pyhrf.test.jdetest), 156
test_suite() (in module pyhrf.test.test_glm), 158
test_sum_treatment() (pyhrf.test.jdetest.JDETest method), 155
test_SW_nrj() (pyhrf.validation.valid_rndm_field.PottsTest method), 197
test_sw_nrj() (pyhrf.validation.valid_rndm_field.PottsTest method), 197
test_SW_nrj_2C_3C() (pyhrf.validation.valid_rndm_field.PottsTest method), 197
test_swendsenwang() (pyhrf.validation.valid_rndm_field.PottsTest method), 197
test_swendsenwang() (pyhrf.test.boldsynthTest.FieldFuncsTest method), 150
test_table_header() (pyhrf.test.test_ndarray.TestHtml method), 161
test_to_latex_1d() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_to_latex_3d() (pyhrf.test.test_ndarray.xndarrayTest method), 161
test_to_latex_3d_col_align() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_to_latex_3d_hide_name_style() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_to_latex_3d_inner_axes() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_to_latex_3d_join_style() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_to_nipy_Block() (pyhrf.test.test_paradigm.ParadigmTest method), 162
test_to_nipy_Block_2sess() (pyhrf.test.test_paradigm.ParadigmTest method), 162

test_to_nipy_ER() (pyhrf.test.test_paradigm.ParadigmTest method), 162
test_to_nipy_ER_2sess() (pyhrf.test.test_paradigm.ParadigmTest method), 162
test_to_spm_mat_1st_level() (pyhrf.test.test_paradigm.ParadigmTest method), 162
test_TreatmentXML() (pyhrf.test.test_xml.InitableTest method), 166
test_tree_to_xndarray() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_tuple_of_misc() (pyhrf.test.test_xml.BaseTest method), 166
test_txt_1d_col_axes_only() (pyhrf.test.test_ndarray.TestHtml method), 161
test_txt_1d_row_axes_only() (pyhrf.test.test_ndarray.TestHtml method), 161
test_txt_tooltip() (pyhrf.test.test_ndarray.TestHtml method), 161
test_unstack_2D() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_unstack_empty_inner_axes() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_uspatialward_formula() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_uward_tree_save() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_var_tracking() (pyhrf.test.test_sampler.GibbsTest method), 164
test_vem_bold_constrained() (pyhrf.test.test_jde_vem_bold.VEMBOLDTest method), 159
test_vem_bold_constrained_python() (pyhrf.test.test_jde_vem_bold.VEMBOLDTest method), 159
test_voronoi_parcellation() (pyhrf.test.test_parcellation.SpatialTest method), 163
test_voronoi_with_seeds() (pyhrf.test.test_parcellation.CmdParcellationTest method), 163
test_walk_branches() (pyhrf.test.toolsTest.treeToolsTest method), 169
test_ward_distance_1D_v1() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_ward_distance_1D_v2() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_ward_distance_2D() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_ward_spatial_cmd() (pyhrf.test.test_parcellation.CmdParcellationTest method), 163
test_ward_spatial_real_data() (pyhrf.test.test_parcellation.CmdParcellationTest method), 163
test_ward_spatial_scikit() (pyhrf.test.test_parcellation.ParcellationMethodTest method), 163
test_ward_spatial_scikit_with_mask() (pyhrf.test.test_parcellation.ParcellationMethodTest method), 163
test_ward_tree_save() (pyhrf.validation.valid_sandbox_parcellation.Parcella method), 198
test_with_subfolders() (pyhrf.test.iotest.RxCopyTest method), 154
test_WNSGGMS() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
test_WNSGGMS_surf_cmd() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
test_wpu_from_forged_features() (pyhrf.validation.valid_sandbox_parcellation.ParcellationTest method), 198
test_write_tex_gii_2D_float() (pyhrf.test.iotest.GiftiTest method), 154
test_write_tex_gii_float() (pyhrf.test.iotest.GiftiTest method), 154
test_write_tex_gii_labels() (pyhrf.test.iotest.GiftiTest method), 154
test_write_tex_gii_time_series() (pyhrf.test.iotest.GiftiTest method), 154
test_xmapping() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_xmapping_inconsistent_domain() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_xmapping_inconsistent_mapping_value() (pyhrf.test.test_ndarray.xndarrayTest method), 162
test_xml_from_classmethod_init() (pyhrf.test.test_xml.InitableTest method), 166
test_xml_from_init() (pyhrf.test.test_xml.InitableTest method), 166
testAll2D() (pyhrf.test.toolsTest.DiagBlockTest method), 167
testBadTreeInit() (pyhrf.test.toolsTest.PipelineTest method), 168
testBasic() (pyhrf.test.toolsTest.CropTest method), 167
testBasic() (pyhrf.test.toolsTest.DictToStringTest method), 167

testCartesianBasic() (pyhrf.test.toolsTest.CartesianTest method), 167
testDefaultWithOutputs() (pyhrf.test.jdetest.JDETest method), 155
testDetectEstimDefault() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
testDynamicParamsHierachic() (pyhrf.test.test_xml.XMLableTest method), 166
testDynamicParamsSingleClass() (pyhrf.test.test_xml.XMLableTest method), 166
testExtrapolation2C() (pyhrf.test.jdetest.PartitionFunctionTest method), 156
testFrom1D() (pyhrf.test.toolsTest.DiagBlockTest method), 167
testFromNdarray() (pyhrf.test.toolsTest.DiagBlockTest method), 167
testGoodDepTreeInit() (pyhrf.test.toolsTest.PipelineTest method), 168
testHrfEstim() (pyhrf.test.commandTest.TreatmentCommandTest method), 152
TestHtml (class in pyhrf.test.test_ndarray), 161
testIncompleteMapping() (pyhrf.test.boldsynthTest.Mapper1DTest method), 150
testIrregularMapping() (pyhrf.test.boldsynthTest.Mapper1DTest method), 150
testLargerTargetGrid() (pyhrf.test.toolsTest.ResampleTest method), 168
testNumpy() (pyhrf.test.test_xml.BaseTest method), 165
testOnHierachicDict() (pyhrf.test.toolsTest.DictToStringTest method), 167
testOnNumpyArray() (pyhrf.test.toolsTest.DictToStringTest method), 167
testOnSpmMat() (pyhrf.test.toolsTest.DictToStringTest method), 167
testPeel() (pyhrf.test.toolsTest.PeelVolumeTest method), 168
testRepFrom1D() (pyhrf.test.toolsTest.DiagBlockTest method), 167
testRepFrom2D() (pyhrf.test.toolsTest.DiagBlockTest method), 167
testRepFromBlocks() (pyhrf.test.toolsTest.DiagBlockTest method), 167
testRepr() (pyhrf.test.toolsTest.PipelineTest method), 168
testResampleToGrid() (pyhrf.test.toolsTest.ResampleTest method), 168
testSimple() (pyhrf.test.statsTest.RPNormTest method), 156
TestXML (class in pyhrf.test.test_xml), 166
THE_ROOT (pyhrf.tools.misc.Pipeline attribute), 172
threshold_labels() (in module pyhrf.stats.misc), 146
threshold_W() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
threshold_W() (pyhrf.jde.wsampler.WSampler method), 127
ThresholdPPM() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 21
time_diff_str() (in module pyhrf.tools.misc), 177
TimeSlot (class in pyhrf.grid), 237
timeslot_help() (in module pyhrf.grid), 238
TimeSlotList (class in pyhrf.grid), 237
to_conquer() (pyhrf.parcellation.Ant method), 249
to_cuboid() (pyhrf.jde.samplerbase.Trajectory method), 126
to_cuboid() (pyhrf.sandbox.stats.Trajectory method), 145
to_dict() (pyhrf.core.FMRISessionSimulationData method), 230
to_dict() (pyhrf.core.FMRISessionSurfacicData method), 230
to_dict() (pyhrf.core.FMRISessionVolumicData method), 230
to_html_table() (pyhrf.ndarray.xndarray method), 245
to_latex() (pyhrf.ndarray.xndarray method), 245
to_nipy_paradigm() (pyhrf.paradigm.Paradigm method), 247
to_patrol() (pyhrf.parcellation.Ant method), 249
to_tree() (pyhrf.ndarray.xndarray method), 245
to_ui_node() (pyhrf.xmlio.Initable method), 258
to_xml() (in module pyhrf.xmlio), 259
to_xml() (in module pyhrf.xmliobak.xmlbase), 226
to_xml() (pyhrf.xmlio.UiNode method), 259
TopClass (class in pyhrf.test.test_xml), 166
tr (pyhrf.core.FmriData attribute), 231
track_obs_quantity() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 126
track_obs_quantity() (pyhrf.sandbox.stats.GibbsSampler method), 145
track_obs_quantity() (pyhrf.sandbox.stats.GSVariable method), 144
track_sampled_quantity() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 126
track_sampled_quantity() (pyhrf.sandbox.stats.GibbsSampler method), 145
track_sampled_quantity() (pyhrf.sandbox.stats.GSVariable method), 144
Trajectory (class in pyhrf.jde.samplerbase), 126
Trajectory (class in pyhrf.sandbox.stats), 145
TrajectoryTest (class in pyhrf.test.test_sampler), 164
TreatmentCommandTest (class in pyhrf.test.commandTest), 152
TreatmentTest (class in pyhrf.test.test_treatment), 165
tree() (in module pyhrf.tools.misc), 177

tree_items() (in module pyhrf.tools.misc), 177
tree_leaves() (in module pyhrf.tools.misc), 177
tree_rearrange() (in module pyhrf.tools.misc), 177
tree_to_xndarray() (in module pyhrf.ndarray), 240
treeBranches() (in module pyhrf.tools.misc), 177
treeBranchesClasses() (in module pyhrf.tools.misc), 177
treeToolsTest (class in pyhrf.test.toolsTest), 169
TriGaussMixtureParamsSampler (class in pyhrf.jde.nrl.trigaussian), 34
truncRandn() (in module pyhrf.stats.random), 149
tty_check() (pyhrf.tools.misc.AnsiColorizer method), 172
tupleDOMWriter() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
tupleTagDOMReader() (pyhrf.xmliobak.xmlbase.TypedXMLHandler static method), 225
type_info() (pyhrf.xmlio.UiNode method), 259
TYPE_LABEL_ARRAY (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_BOOL (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_CELL (pyhrf.xmliobak.xmlmatlab.MatlabXMLHandler attribute), 226
TYPE_LABEL_DICT (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_DOUBLE (pyhrf.xmliobak.xmlmatlab.MatlabXMLHandler attribute), 226
TYPE_LABEL_FLOAT (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_INT (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_KEY_VAL_PAIR (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_LIST (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_NONE (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_ODICT (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_STRING (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_TUPLE (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TYPE_LABEL_XML_INCLUDE (pyhrf.xmliobak.xmlbase.TypedXMLHandler attribute), 224
TypedXMLHandler (class in pyhrf.xmliobak.xmlbase), 223

U

UiNode (class in pyhrf.xmlio), 258
UnboundSpatialMapping (class in pyhrf.boldsynth.spatialconfig), 16
undrift() (in module pyhrf.tools.misc), 177
unformat_nrl_file() (in module pyhrf.sandbox.data_parser), 128
UniformGenerator (class in pyhrf.stats.random), 147
unknown_host_status (pyhrf.grid.HostsManager attribute), 236
unknown_status (pyhrf.grid.HostsManager attribute), 236
unprotect_xml_attr() (in module pyhrf.xmlio), 260
unserialize_attributes (pyhrf.xmlio.UiNode attribute), 259
unstack() (pyhrf.ndarray.xndarray method), 246
unstack_trees() (in module pyhrf.tools.misc), 177
update() (pyhrf.sandbox.stats.Trajectory method), 145
update() (pyhrf.tools.backports.OrderedDict method), 170
update_all() (pyhrf.tools.misc.Pipeline method), 173
update_all_hosts() (pyhrf.grid.HostsManager method), 236
update_host_status() (pyhrf.grid.HostsManager method), 236
update_observables() (pyhrf.sandbox.stats.GSVariable method), 144
update_quantity() (pyhrf.tools.misc.Pipeline method), 173
update_subgraph() (pyhrf.tools.misc.Pipeline method), 173
updateClassCounts() (pyhrf.boldsynth.spatialconfig.StateField method), 16
updateGammaTimeNRLs() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
updateGlobalObservables() (pyhrf.jde.jde_multi_sess.BOLDGibbs_Multi_SessSampler method), 92
updateGlobalObservables() (pyhrf.jde.jde_multi_sujets.BOLDGibbs_Multi_SubjSampler method), 100
updateGlobalObservables() (pyhrf.jde.jde_multi_sujets_alpha.BOLDGibbs_Multi_SubjSamp method), 108
updateGlobalObservables() (pyhrf.jde.models.BOLDGibbsSampler method), 116
updateGlobalObservables() (pyhrf.jde.models.BOLDGibbsSampler_AR method), 117
updateGlobalObservables() (pyhrf.jde.samplerbase.GibbsSampler method), 125
updateNorm() (pyhrf.jde.asl.DriftCoeffSampler method), 36
updateNorm() (pyhrf.jde.asl.ResponseSampler method),

39
updateNorm() (pyhrf.jde.asl_physio.DriftCoeffSampler method), 42
updateNorm() (pyhrf.jde.asl_physio.ResponseSampler method), 46
updateNorm() (pyhrf.jde.asl_physio_1step.DriftCoeffSampler method), 48
updateNorm() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 52
updateNorm() (pyhrf.jde.asl_physio_1step_params.DriftCoeffSampler method), 54
updateNorm() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 59
updateNorm() (pyhrf.jde.asl_physio_det_fwdm.DriftCoeffSampler method), 62
updateNorm() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
updateNorm() (pyhrf.jde.asl_physio_hierarchical.DriftCoeffSampler method), 69
updateNorm() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
updateNorm() (pyhrf.jde.asl_physio_joint.DriftCoeffSampler method), 76
updateNorm() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
updateNorm() (pyhrf.jde.drift.DriftARSampler method), 86
updateNorm() (pyhrf.jde.drift.DriftSampler method), 86
updateNorm() (pyhrf.jde.drift.DriftSamplerWithRelVar method), 87
updateNorm() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 90
updateNorm() (pyhrf.jde.hrf.HRFSampler method), 88
updateNorm() (pyhrf.jde.hrf.HRFwithHabSampler method), 90
updateNorm() (pyhrf.jde.jde_multi_sess.Drift_MultiSess_Sampler method), 93
updateNorm() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
updateNorm() (pyhrf.jde.jde_multi_sujets.Drift_MultiSubj_Sampler method), 100
updateNorm() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
updateNorm() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 103
updateNorm() (pyhrf.jde.jde_multi_sujets_alpha.Drift_MultiSubj_Sampler method), 109
updateNorm() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampler method), 111
updateNorm() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
updateObsersables() (pyhrf.jde.asl.ResponseLevelSampler method), 39
updateObsersables() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 90
updateObsersables() (pyhrf.jde.hrf.HRFSampler method), 88
updateObsersables() (pyhrf.jde.jde_multi_sess.BiGaussMixtureParams_Mu method), 93
updateObsersables() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
updateObsersables() (pyhrf.jde.jde_multi_sujets.HRF_Group_Sampler method), 102
updateObsersables() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 103
updateObsersables() (pyhrf.jde.jde_multi_sujets_alpha.BiGaussMixturePara method), 109
updateObsersables() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Group_Sampl method), 111
updateObsersables() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
updateObsersables() (pyhrf.jde.nrl.bigaussian.BiGaussMixtureParamsSamp method), 19
updateObsersables() (pyhrf.jde.nrl.bigaussian.NRLSampler method), 22
updateObsersables() (pyhrf.jde.nrl.bigaussian_drift.BiGaussMixtureParams method), 26
updateObsersables() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
updateObsersables() (pyhrf.jde.samplerbase.GibbsSamplerVariable method), 126
updateObsersables() (pyhrf.jde.wsampler.W_Drift_Sampler method), 127
updateObsersables() (pyhrf.jde.wsampler.WSampler method), 127
updateParameters() (pyhrf.xmliobak.xmlbase.XMLParamDrivenClass method), 226
updateVarYmDrift() (pyhrf.jde.drift.DriftARSampler method), 86
updateXh() (pyhrf.jde.hrf.HRF_two_parts_Sampler method), 90
updateXh() (pyhrf.jde.hrf.HRFSampler method), 88
updateXh() (pyhrf.jde.jde_multi_sess.HRF_MultiSess_Sampler method), 94
updateXh() (pyhrf.jde.jde_multi_sujets.HRF_Sampler method), 103
updateXh() (pyhrf.jde.jde_multi_sujets_alpha.HRF_Sampler method), 111
updateXh() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
updateXResp() (pyhrf.jde.asl.ResponseSampler method), 39
updateXResp() (pyhrf.jde.asl_physio.ResponseSampler method), 46
updateXResp() (pyhrf.jde.asl_physio_1step.ResponseSampler method), 52
updateXResp() (pyhrf.jde.asl_physio_1step_params.ResponseSampler method), 60

updateXResp() (pyhrf.jde.asl_physio_det_fwdm.ResponseSampler method), 66
updateXResp() (pyhrf.jde.asl_physio_hierarchical.ResponseSampler method), 74
updateXResp() (pyhrf.jde.asl_physio_joint.ResponseSampler method), 80
updateYtilde() (pyhrf.jde.nrl.habituation.NRLwithHabSampler method), 33
User (class in pyhrf.grid), 238

V

VAL_INI (pyhrf.jde.jde_multi_sujets_alpha.HRFVarianceSubjectSampler attribute), 110
VAL_INI (pyhrf.jde.jde_multi_sujets_alpha.RHGroupSampler attribute), 113
values() (pyhrf.tools.backports.OrderedDict method), 170
var() (pyhrf.ndarray.xndarray method), 246
variables (pyhrf.tools.aexpression.ArithmeticExpression attribute), 169
VariableTypeException, 126
Variance_GaussianNRL_Multi_Sess (class in pyhrf.jde.jde_multi_sess), 96
Variance_GaussianNRL_Multi_Sess (class in pyhrf.jde.nrl.bigaussian), 24
Variance_GaussianNRL_Multi_Subj (class in pyhrf.jde.jde_multi_sujets), 104
Variance_GaussianNRL_Multi_Subj (class in pyhrf.jde.jde_multi_sujets_alpha), 113
VEMASLTest (class in pyhrf.test.test_jde_vem_asl), 158
VEMBOLDTest (class in pyhrf.test.test_jde_vem_bold), 159
VEMToolsTest (class in pyhrf.test.test_jde_vem_tools), 159
VEMToolsTest (class in pyhrf.test.test_jde_vem_tools_asl), 160
VEMToolsTest (class in pyhrf.test.test_jde_vem_tools_UtilsC), 160
Verbose (class in pyhrf), 1
verbose() (pyhrf.parcellation.Talker method), 249
verbose_array() (pyhrf.parcellation.Talker method), 249
viewitems() (pyhrf.tools.backports.OrderedDict method), 170
viewkeys() (pyhrf.tools.backports.OrderedDict method), 170
viewvalues() (pyhrf.tools.backports.OrderedDict method), 170
Visit_graph_noeud() (in module pyhrf.parcellation), 249

W

W_BOLDGibbsSampler (class in pyhrf.jde.models), 119
W_Drift_BOLDGibbsSampler (class in pyhrf.jde.models), 119
W_Drift_Sampler (class in pyhrf.jde.wsampler), 127
wait_for_end_or_cmd() (pyhrf.grid.DispatchedTasksManager method), 236
wait_to_be_ready() (pyhrf.grid.TasksManager method), 237
walkCluster() (in module pyhrf.boldsynth.pottsfield.swendsenwang), 7
Ward (class in pyhrf.sandbox.parcellation), 130
ward_tree() (in module pyhrf.sandbox.parcellation), 135
ward_tree_save() (in module pyhrf.sandbox.parcellation), 135
WardAgglomeration (class in pyhrf.sandbox.parcellation), 130
warning() (pyhrf.tools.message.MessageColor class method), 171
warning() (pyhrf.tools.message.MessageNoColor class method), 171
warning() (pyhrf.tools.message.NoMessage method), 171
WN_BiG_ASLSamplerInput (class in pyhrf.jde.asl), 39
WN_BiG_ASLSamplerInput (class in pyhrf.jde.asl_physio), 46
WN_BiG_ASLSamplerInput (class in pyhrf.jde.asl_physio_1step), 52
WN_BiG_ASLSamplerInput (class in pyhrf.jde.asl_physio_1step_params), 60
WN_BiG_ASLSamplerInput (class in pyhrf.jde.asl_physio_det_fwdm), 66
WN_BiG_ASLSamplerInput (class in pyhrf.jde.asl_physio_hierarchical), 74
WN_BiG_ASLSamplerInput (class in pyhrf.jde.asl_physio_joint), 80
WN_BiG_BOLDSamplerInput (class in pyhrf.jde.models), 119
WN_BiG_Drift_BOLDSamplerInput (class in pyhrf.jde.models), 119
World (class in pyhrf.parcellation), 249
write() (pyhrf.tools.message.MessageColor class method), 171
write() (pyhrf.tools.message.NoMessage method), 172
write_configuration() (in module pyhrf.configuration), 229
write_list() (pyhrf.tools.message.MessageColor class method), 171
write_list() (pyhrf.tools.message.NoMessage method), 172
write_xml() (in module pyhrf.xmlio), 260
write_xml() (in module pyhrf.xmliobak.xmlbase), 226
writeDOMData() (pyhrf.xmliobak.xmlbase.TypedXMLHandler method), 225
WSampler (class in pyhrf.jde.wsampler), 126

X

XMLable (class in pyhrf.xmliobak.xmlbase), 226
XMLable2 (class in pyhrf.xmliobak.xmlbase), 226

XMLableTest (class in pyhrf.test.test_xml), 166
xmlComment (pyhrf.ui.treatment.FMRITreatment attribute), 183
XmlInitable (in module pyhrf.xmlio), 259
XMLParamDrivenClass (class in pyhrf.xmliobak.xmlbase), 225
XMLParamDrivenClassInitException (class in pyhrf.xmliobak.xmlbase), 226
xndarray (class in pyhrf.ndarray), 240
xndarray_like() (in module pyhrf.ndarray), 246
xndarray_like() (pyhrf.ndarray.xndarray static method), 246
xndarrayIOTest (class in pyhrf.test.iotest), 155
xndarrayMapper1D (class in pyhrf.boldsynth.spatialconfig), 17
xndarrayTest (class in pyhrf.test.test_ndarray), 161

Z

Z_Entropy() (in module pyhrf.vbjde.vem_tools), 208
ZeroGenerator (class in pyhrf.stats.random), 147