CanlabCore Documentation

Release 1.0

Tor Wager

March 31, 2016

Contents

The CANlab Core Tools package is a set of Matlab functions that are designed to serve miscellaneous purposes. Many of them have stand-alone command line function, but the toolbox also contains functions that are used in other CANlab toolboxes, and so this package should be downloaded and put on the Matlab path when using any of the toolboxes.

Contents:

CHAPTER 1

Object Oriented Tools

1.1 canlab_dataset

@canlab_dataset.**bars**(*obj*, *varnames*, *varargin*)

Bar plot for canlab_dataset object

Usage

```
[dat, descrip, colors, h1, s1] = bars(obj, varnames, [optional inputs])
```

Inputs

obj: canlab_dataset object

varnames: Cell string of variable names to plot

Optional Inputs

colors: defined colors (default are set by scn_standard_colors.m)

nofig: do not generate figure

Takes any optional inputs to barplot_colored.m

Outputs

dat: data matrix for each variable

descrip: the description for this variable

colors: selected colors (default are set by scn_standard_colors.m)

h1: figure handle

s1: axis handle

Examples

```
create_figure('NPS values - All subjects');

varnames = {'15' '13' '11' ' 9' '16' '14' '12' '10'};
xvals = [1 2 4 5 8 9 11 12];
colors = {[1 0 0] [0 1 0] [1 0 0] [0 1 0] [1 0 0] [0 1 0] [1 0 0] [0 1 0]};
bars(LevoNPS, varnames, 'x', xvals, 'colors', colors, 'XTickLabels', varnames, 'within', 'nofig'
```

@canlab_dataset.**concatenate**(*D*, *varargin*)

Concatenates Subject-level and Event-level data across all subjects

Usage

```
[names ids dat] = concatenate(D, [optional inputs])
```

Inputs

D: canlab_dataset object

Optional Inputs

a logical array a vector of 1/0 values to use as wh_keep

Outputs

names: cell array of variable names

ids: subject IDs matching data rows in dat

dat:

subjects*events x variables matrix

- subject number, event number are included
- all subject-level and event-level data are included
- this format appropriate for, e.g., SAS/HLM

descrips: cell array of variable descriptions

Examples

```
[names, ids, flatdat] = concatenate(D);
id_numbers = flatdat(:, 1);

wh_subjs = true(size(D.Subj_Level.id));
wh_subjs([13 18 19]) = false;
[names, ids, dat] = concatenate(D, wh_subjs);
```

@canlab_dataset.**get_var**(*D*, *varname*, *varargin*)

Get Subject-level or Event-level variable from dataset D and return in rect matrix and cell array. Multiple variables can be requested, but all data requested must be either numeric or text, and not a combination of the two.

Usage

```
[dat, datcell, wh_level, descrip] = get_var(D, varname, [opt inputs])
```

Inputs

D: a canlab_dataset object

varname:

the name of a variable to get from dataset

- Looks for var name at either level, returns error if exists at both levels
- can be a cell array of multiple var names in this case, dat is a n x m matrix, where n=subjs and m=variables requested

Optional inputs a logical array vector of 1/0 values to use as wh_keep

conditional: to be followed by a cell array; the first cell is the name of the variable to be conditionally selected upon, the second cell contains the condition which must be met. Example: get_var(D, 'DeltaDon', 'conditional', {'trained' 1}) will get DeltaDon whenever trained==1. Currently only implemented for event-level data. Could be expanded to include multiple conditions.

Outputs

dat:

rect matrix of subjects X events (X variables)

- good for plotting individuals, means/std. errors across subjects
- is actually a cell matrix if textual data is requested.

datcell:

1 x subjects cell array, each cell containing event data for one subject

- good for input into some stats functions, e.g., glmfit_multilevel and igls.m

wh_level: 1 = 'Subject'; 2 = 'Event';

descrip: the description for this variable

`@canlab_dataset.`**`glm`**(*D*, *Yvarname*, *Xvarnames*, *wh_keep*)

predict Y from X using GLM

Usage

```
out = glm(D, Yvarname, Xvarnames, wh_keep)
```

Inputs

D: a canlab_dataset object

Yvarname: the name of a variable to predict. must be subject level

Xvarnames: the name(s) of predictor variables. if multiple, put in cell array. must be subject_level

wh_keep: a logical vector of 1/0 values

Outputs

out: structure containing same output as for glmfit() out.b: a vector of coefficient estimates out.dev: the deviance of the fit out.stat: see glmfit documentation for stat structure fields

Examples

```
out = glm(D, 'DeltaDon_avg', prednames, wh_keep)
```

`@canlab_dataset.`**`glm_multilevel`**(*D*, *Yvarname*, *Xvarnames*, *wh_keep*)

Predict Y from X using GLM

Usage

```
[b, dev, stat] = glm_multilevel(D, Yvarname, Xvarnames, wh_keep)
```

Inputs

D: a canlab_dataset object

Yvarname: the name of a variable to predict. must be event level

Xvarnames: the name(s) of predictor variables. if multiple, put in cell array. must be event level

wh_keep: a logical vector of 1/0 values

Outputs

b: a vector of coefficient estimates (same as for glmfit())

dev: the deviance of the fit (same as for glmfit())

stat: structure containing stats fields (see glmfit() documentation)

`@canlab_dataset.`**`histogram`**(*D*, *varargin*)

Histogram of one variable in dataset

- can be either event-level or subject-level
- event-level data is plotted as concatenated events across subject-level
- both variables must be valid names (case-sensitive)

Usage

```
fig_han = histogram(D, [optional inputs]);
```

Inputs

D: canlab_dataset

Optional Inputs **nofig': suppress creation of new figure

Outputs

fig_han: figure handle

`@canlab_dataset.`**`mediation`**(*D*, *xvarname*, *yvarname*, *mvarname*, *varargin*)

Run single or multilevel mediation analysis on a canlab_dataset object. Calls mediation.m (see mediation.m) in mediation toolbox

Usage

```
[paths, stats] = mediation(D, xvarname, yvarname, mvarname, [optional inputs])
```

Inputs

D: is a canlab_dataset object

xvarname: X, the initial variable (valid variable name in the dataset)

yvarname: Y, the outcome variable (valid variable name in the dataset)

mvarname: M, the potential mediator ((valid variable name in the dataset)

Optional Inputs Takes any optional inputs to mediation.m e.g., 'noverbose', 'dosave', 'names', 'M', 'L2M', 'covs', others

wh_keep:

followed by 1/0 vector of subjects to keep.

- must be same length as subjects
- subjects with value 0 will be excluded

rankdata: ranks all data before mediation; "Nonparametric"

Outputs

paths: see mediation.m from mediation toolbox

stats: see mediation.m from mediation toolbox

Examples

```
[paths, stats] = mediation(D, 'Group', 'DeltaDon', 'DeltaDist', 'M2', 'DeltaTend', 'wh_keep', wh
```

`@canlab_dataset.`**`plot_var`**(*D*, *varname*, *varargin*)

Plot the mean and standard error of a variable across events.

Usage

```
[meandat, stedat] = plot_var(D, varname, [optional inputs])
```

Inputs

D: a canlab_dataset object

varname:

the name of a valid variable to get from dataset -Looks for var name at either level, returns Event level if exists at both levels

Optional inputs

subjtype:

followed by name of grouping variable

- must be categorical subject-level variable
- if entered, plot lines or bars based on these categories
- 'eventmeans' will plot bars; without, it will plot line plots across events with standard error shading
- the grouping variable's description, if it exists, will be split along commas, and those values will be used as column lables

eventmeans:

calculate and plot subject means across event-level variables

- if entered, will plot bar plots of means by condition

wh_keep:

followed by 1/0 vector of subjects to keep.

- must be same length as subjects
- subjects with value 0 will be excluded

color: followed by one color for all bars, or cell array with names of colors cell for each line/bar

nofig: don't make a new figure

other: other varargin are passed directly to barplot_columns. So for example, '95CI' will make 95% confidence interals, instead of SE bars.

Outputs

meandat: mean values

stedat: standard error values

Examples

```
plot_var(D, 'Frustration')
plot_var(D, 'RT')
plot_var(D, 'RT', 'eventmeans');
plot_var(D, 'RT', 'subjtype', 'Placebo');
plot_var(D, 'RT', 'eventmeans', 'subjtype', 'Placebo');
plot_var(D, 'RT', 'eventmeans', 'subjtype', 'Placebo', 'color', {'r' 'b'});
```

`@canlab_dataset.`**`print_summary`**(*D*, *varargin*)

Prints summaries for every variable, or specified variables

Usage

```
print_summary(D, [optional inputs])
```

Inputs

D: a canlab_dataset object

Optional Inputs

subj: followed by a cell array of subject level var names, to only see those vars

event: followed by a cell array of event level var names, to only see those vars

if either varargin is unspecified, all variables will be printed

`@canlab_dataset.`**`read_from_excel`**(*dat*, *ExperimentFileName*, *SubjectFileList*, *varargin*)

Read from datafile into canlab_dataset format - currently requires file extensions .xls or .xlsx, but in the future will use importdata to take .csv or .txt extensions as well.

Datafiles require column headers

- **Design file requires: id, names, units, descrip**
  - other columns can be added
  - **ONLY between subject columns identified in the 'names'** column are added.
  - **See Sample_canlab_dataset_experiment_level.xlsx for an** example design file
- **Subject files require no specific column headers, but all column** headers must be identical across all subjects. - Enter NaN for data field in file if no value for that column within a specific Event - ALL columns of subject files are written to canlab_dataset

Usage

```
dat = read_from_excel(dat, ExperimentFileName, SubjectFileList, [optional inputs])
```

Inputs

dat: a canlab_dataset object

ExperimentFileName: the absolute path of the experiment data file

SubjectFileList:

list of absolute paths for individual subject files

- plays well with filenames()

Optional Inputs

fmri: Indicates construction of canlab_dataset object using 'fmri' code. Suppresses overwrite warnings specific to 'fmri' inputs.

Outputs

dat: canlab_dataset object with uploaded values

Examples

```
% To output a file into a raw fmri dataset

DesignFile = fullfile(pwd,'Sample_canlab_dataset_experiment_level.xlsx');
SubjectFiles = filenames(fullfile(pwd,'Sample_canlab_dataset_subject*.xlsx'));
dat = canlab_dataset('fmri');
dat = read_from_excel(dat,DesignFile,SubjectFiles,'fmri');
```

@canlab_dataset.**scattermatrix**(*D*, *wh_level*, *wh_vars*)

Scatterplot matrix of pairwise event-level variables

Usage

```
fig_han = scattermatrix(D, wh_level, wh_vars)
```

Inputs

D: a canlab_dataset object

wh_level: 1 (Subject) or 2 (Event)

Outputs

fig_han: figure handle

Examples

```
fig_han = scattermatrix(D);

wh = [5:9];
fig_han = scattermatrix(D, 2, wh);

f = scattermatrix(D, 2, {'Choice' 'RT' 'Pain' 'SwitchNext' 'Frustration' 'Anxiety' 'Control'});
```

@canlab_dataset.**scatterplot**(*D*, *v1*, *v2*, *varargin*)

Scatterplot of two variables in dataset

- can be either event-level or subject-level
- event-level data is plotted as multi-line plot, one line per subject
- both variables must be valid names (case-sensitive)

Usage

```
fig_han = scatterplot(D, varname1, varname2, [optional inputs])
```

Inputs

D: a canlab_dataset object

v1: x variable

v2: y variable

Optional Inputs

nofig: suppress creation of new figure

subjtype: group by the following variable name

wh_keep: followed by logical

colors: followed by colors.

dorobust: do robust corr. if enabled, colors will not work and subjtype grouping will not work well until the function plot_correlation_samefig is updated, at some point in the future.

Outputs

fig_han: figure handle

Examples

```
scatterplot(D, 'Anxiety', 'Frustration');
fig_han = scatterplot(D, D.Subj_Level.names{1}, D.Subj_Level.names{2});
scatterplot(D, D.Event_Level.names{1}, D.Event_Level.names{2});
```

`@canlab_dataset.`**`spm2canlab_dataset`**(*obj*, *subject*, *spm*)

Extract Event_Level data from subjects' SPM.mat files to add data to canlab_dataset object.

Usage

```
obj = spm2canlab_dataset(obj, subject, spm)
```

Inputs

obj: Canlab_dataset object (see canlab_dataset)

subject: Subject list (it could be one subject [in a string format], or it could be multiple subjects in cell array)

spm: This could be loaded SPM (struct), or one path for one subject's SPM.mat file (string), or multiple loaded SPM or paths in cell array

Outputs

obj: Canlab_dataset object with new data

Examples

```
subj = {'dpsp002','dpsp003'};
spm = {'dpsp002_SPM.mat', 'dpsp003_SPM.mat'};

D = canlab_dataset; % if D doesn't exist yet
D = spm2canlab_dataset(D, subj, spm);
```

See also canlab_dataset spm_mat2batchinput

`@canlab_dataset.`**`ttest2`**(*D*, *varname*, *wh_keep1*, *wh_keep2*, *varargin*)

Two sample ttest for two samples of one subject-level variable

Usage

```
ttest2(D, varname, wh_keep1, wh_keep2, [optional inputs])
```

Inputs

D: a canlab_dataset object

varname: the name of a valid variable to get from dataset

wh_keep1: subjects forming first sample

wh_keep2: subjects forming second sample

Optional Inputs

noverbose: will suppress print out of results and bargraph

varargin: other variables passed directly to MATLAB's ttest2

Outputs same as MATLAB's ttest2 output

`@canlab_dataset.`**`write_text`**(*D*, *varargin*)

"Flatten" dataset and write text files with header and data For all Event-level and Subject-level data. Files are created in the current working directory.

Usage

```
function [headername, dataname, fid] = write_text(D, [optional inputs])
```

Inputs

D: a canlab_dataset object

Optional Inputs the first varargin parameter is the delimiter. Comma-delimited by default.

Outputs

headername: filename of header output file

dataname: filename of data output file

fid: file ID (currently does not look to be used)

1.2 fmri_data

`@fmri_data.`**`canlab_connectivity_preproc`**(*dat*, *varargin*)

This function prepares data for connectivity analysis by removing nuisance variables and temporal filtering (high, low, or bandpass filter). This also can extract values from given masks and return averaged activity or pattern expression values.

Usage

```
[preprocessed_dat, roi_val] = canlab_connectivity_preproc(dat, varargin)
```

Features

- can regress out nuisance variables with any additional nuisance matrix
- can remove signal from ventricle and white matter (calls canlab_extract_ventricle_wm_timeseries.m and canlab_create_wm_ventricle_masks.m)

- can do temporal filtering, including high-pass, low-pass, or bandpass filtering (it uses conn_filter.m from conn toolbox; see subfunction below)
- can extract data from given ROIs, and return averaged value or pattern expression value (dot-product).

Steps in order [with defaults]:

1. Remove nuisance covariates (and linear trend if requested)
2. Remove ventricle and white matter - needs structural images
3. Windsorize based on distribution of full data matrix
4. High/low/bandpass filter
5. Extract region-by-region average ROI or pattern expression data

Inputs

dat: fmri_data object with data

dat.covariate: basic nuisance matrix

Optional Inputs

additional_nuisance When you have additional nuisance variables that you want regress out from the data, you can use this option. This option should be followed by a nuisance matrix (or values). The matrix should have the same number of rows with the number of images.

vw When you want to regress out signals from ventricle and white matter, you can use this option. To use this option, You should provide the directory where the subjects' data are saved using the 'datdir' (for example, see below). Requires specific subdirectory structure (CANlab) - see code.

You can also choose what to use to remove ventricle and white matter signal between raw data or top 5 PCA components (default). You can just put 'raw' if you want to use raw signal than PCA compoenents. also see: canlab_extract_ventricle_wm_timeseries.m canlab_create_wm_ventricle_masks.m) - *Example:* 'vw', 'datdir', subject_dir, 'raw'

windsorize: Windsorizing entire data matrix to k x STD. - *Example:* 'windsorize', 5 (windsorize to 5 STD)

linear_trend: This option will include the linear trend to nuisance variables.

hpf', 'lpf', or 'bpf: This option will do temporal filtering. - 'hpf': high pass filter. This option should be followed by

the lower bound of the frequency (e.g., .01 Hz [= 100 sec]).

- **'lpf': low pass filter. This option should be followed by** the upper bound of the frequency (e.g., .25 Hz [= 4 sec]).
- **'bpf': bandpass filter. This should be followed by lower** and upper bounds of the frequency (e.g., [.01 .25]). After the frequency value, you need to provide TR.
- ***Example:* 'hpf', .01, TR** 'bpf', [.01 .25], TR

extract_roi: This option will extract data from ROIs specified. This option should be followed by one or more masks. For one mask (potentially multiple ROIs, enter a char array with the mask name. For multiple masks (1 or more), enter in a cell array of mask names. You can specify methods with 'roi_methods' option. - 'average_over' (default): calculate averaged value across the ROIs. - 'pattern_expression': calculate dot-products between

pattern mask and data

- **'unique_mask_values' (default): will divide a mask into** multiple regions that have different discrete values.
- **'contiguous_regions': will divide a mask into multiple** contiguous regions.
- **'whole': will do average_over or pattern_expression across** all the voxels within the mask.
- ***Example:*** **'extract_roi', mask, 'contiguous_regions'** 'extract_roi', mask, 'pattern_expression'

no_preproc: If you want to skip the preprocessing part, and want to extract ROI values only, you can use this option.

Outputs

preprocessed_dat: fmri_data object after removing nuisance variables and filtering temporal confounds.

roi_val: returns values extracted from ROIs in cell arrays (if there are many different ROIs). Each cell will have roi_val.dat, roi_val.mask_name, and roi_val.methods.

Examples

```
roi_masks = which('weights_NSF_grouppred_cvpcr.img');
[preprocessed_dat, roi_val] = canlab_connectivity_preproc(dat, 'vw', 'datdir',
       subject_dir, 'bpf', [.008 .25], TR, 'extract_roi', roi_masks, 'pattern_expression');
```

@fmri_data.**create**(*obj*, *varargin*)

Create an object from an empty obj structure, assigning fieldname/value pairs as optional arguments.

Usage

```
[obj = create(obj, varargin)
```

Used in fmri_data.m class constructor. if 'noverbose' is entered, suppress output

@fmri_data.**extract_roi_averages**(*obj*, *mask_image*, *varargin*)

This fmri_data method a extracts and averages data stored in an fmri_data object from a set of ROIs defined in a mask.

If no mask_image is entered, it uses the mask defined with the fmri_data object as a default.

If mask_image is a new image file name, this method:

1. Defines an fmri_mask_image object using create_fmri_mask
2. Maps to the space in fmri_data object using resample_to_image_space

Regions to average over can be either regions of contiguous voxels bounded by voxels with values of 0 or NaN, which are considered non-data values, or regions defined by unique integer codes in the mask image (i.e., for atlas images with unique codes for each defined region.)

Mask/Atlas image does NOT have to be in the same space as the images to extract from. It will be remapped/resliced. NOTE: Mask is *reloaded* from original data if space is remapped, and you cannot use manual thresholding of the mask. This is a feature of the map_to_image_space method and scn_map_image

Extracted data is returned in single data format.

Usage

```
[[cl, clroimean, clpattern] = extract_roi_averages(fmri_data obj, [mask_image], [average_over])
```

Inputs

1. char array of strings containing 4D image file names (data extracted from these)
2. mask_image to extract from.

Optional inputs

how to average Default = 'unique_mask_values' to average over unique integer codes in the mask image bounded by voxels of 0 or NaN (non-data values) (i.e., for atlas images with unique codes for each defined region) Alt. option = 'contiguous_regions' to average over contiguous voxels

pattern_expression Use values in mask images to get weighted average within each region, rather than simple average. See also apply_mask with 'pattern_expression' option.

Optional outputs (varargout): [cl, cl_roimean, cl_roipattern] = ... roimean: pattern expression is average over ROI (unit vector) roipattern: pattern expression is dot product of activity and mean-centered pattern weights

nonorm Turn off L1 norm in pattern expression.

Examples

```
imgs_to_extract_from = filenames('w*.nii','char');
mask_image = which('anat_lbpa_thal.img');
[cl, clroimean, clpattern] = extract_image_data(imgs_to_extract_from, mask_image);

region_obj = extract_roi_averages(data_obj, mask_char_name, 'pattern_expression', 'contiguous_re
```

Related functions For an non-object-oriented alternative, see extract_image_data.m

`@fmri_data.`**`horzcat`**(*varargin*)

Implements the horzcat ([a b]) operator on image_vector objects across voxels. Requires that each object has an equal number of columns and voxels

Usage

```
function s = horzcat(varargin)
```

Example

```
c = [dat1 dat2];
```

`@fmri_data.`**`hrf_fit`**(*obj*, *TR*, *Runc*, *T*, *method*, *mode*)

HRF estimation on fmri_data class object

HRF estimation function for a single voxel;

Implemented methods include: IL-model (Deterministic/Stochastic), FIR (Regular/Smooth), and HRF (Canonical/+ temporal/+ temporal & dispersion)

Inputs

obj fMRI object

TR time resolution

Runs expermental design

T length of estimated HRF ij seconds

type Model type: 'FIR', 'IL', or 'CHRF'

mode Mode

Model Types

1. **Fit HRF using IL-function**

   Choose mode (deterministic/stochastic)

   - 0 - deterministic aproach
   - 1 - simulated annealing approach

   Please note that when using simulated annealing approach you may need to perform some tuning before use.

2. **Fit HRF using FIR-model**

   Choose mode (FIR/sFIR)

   - 0 - FIR
   - 1 - smooth FIR

3. **Fit HRF using FIR-model**

   Choose mode (FIR/sFIR)

   - 0 - FIR
   - 1 - smooth FIR

Examples

SIMULATE DATA AND RUN

```
%params for sim and fitting
TR = 2;   % repetition time (sec)
n = 200;  % time points measured (for simulation) must be multiple of 10
T = 30;   % duration of HRF to estimate (seconds)
nconds = 2; % num conditions
nevents = 8; % events per condition

% Create fake data
h = spm_hrf(TR);
y = zeros(n, 1);

% onsets - indicator
Condition = {};
for i = 1:nconds
    Condition{i} = zeros(n,1);
    wh = randperm(n);
    Condition{i}(wh(1:nevents)) = 1;

    ytmp{i} =  conv(Condition{i}, h);
    ytmp{i} = ytmp{i}(1:n);
end

y = sum(cat(2, ytmp{:}), 2);

dat = fmri_data('VMPFC_mask_neurosynth.img');  % AVAILABLE ON WIKI IN MASK GALLERY
```

```
    dat = threshold(dat, [5 Inf], 'raw-between');

    v = size(dat.dat, 1); % voxels in mask
    dat.dat = repmat(y',v, 1) + .1 * randn(v, n);

    % Fit data - estimate HRFs across the brain mask
    [params_obj hrf_obj] = hrf_fit(dat,TR, Condition, T,'FIR', 1);

    hrf = fmri_data('HRF_timecourse_cond0001.img');
    hrf = remove_empty(hrf);
    create_figure('hrfs', 1, 2);
    plot(hrf.dat');
    title('Condition 1')
    hrf = fmri_data('HRF_timecourse_cond0002.img');
    hrf = remove_empty(hrf);
    subplot(1, 2, 2);
    plot(hrf.dat');
    title('Condition 2')
```

@fmri_data.**plot** (*fmridat*, *plotmethod*)

Plot means by condition plot(fmri_data_object, 'means_for_unique_Y')

Inputs

Plot methods:

- plot data matrix
- plot(fmri_data_object)

Usage

```
    plot(fmridat, [plotmethod])
```

Outputs

5 plots and an SPM orthviews presentation of the data. In the below and elsewhere, "image" connotes a 3D brain volume captured every TR.

subplot 1: the fMRI data itself. Color is intensity of signal.

subplot 2: presented as a histogram of values for every voxel collected. The low values are typically out-of-brain voxels, as there is no signal there.

subplot 3: each point is an image. The point's X value is the mean intensity of every voxel in that image, and the Y value is the stdev of intensities for all voxels in that image.

subplot 4: covariance between images

subplot 5: each point is an image (case = image). X value is image number in the run, Y is image mean intensity, and the size of the circular marker represents stdev for that image

Orthviews: mean and STD for a given voxel averaged over time. Note that the values for mean and STD here are higher than in the plots above. That is because mean and STD are calculated here by voxel, but in the plots above they are calculated by image. Images also include out-of-brain areas.

@fmri_data.**predict** (*obj*, *varargin*)

Predict outcome (Y) from brain data and test cross-validated error rate for an fmri_data object

Usage

```
[cverr, stats, optional_outputs] = predict(obj, varargin)
```

Features

- flexible specification of algorithm by function name
- k-fold cross-validation, default = 5-fold, can enter custom fold membership
- folds are stratified on outcome
- choice of multiple error metrics (class loss, mse, etc.)
- by default, chooses error metric based on outcome type (classes vs. continuous-valued)
- returns all outputs for each fold returned by the algorithm in optout cell array variable
- bootstrapping of weights built in [optional keyword]
- select variable number of components (for pcr-based techniques)

Inputs

obj is mandatory, rest are optional

obj: fmri_data or image_vector object, with fields .dat (data used to predict) and .Y (outcome)

Optional inputs (with their default values)

nfolds = 5 number of folds

nfolds = [vector of integers] can also input vector of integers for holdout set IDs

error_type = mcr mcr, mse: misclassification rate or mean sq. error

algorithm_name = 'cv_regress' name of m-file defining training/test function

useparallel = 1 Use parallel processing, if available; follow by 1 for yes, 0 for no

bootweights = 0 bootstrap voxel weights; enter bootweights do bootstrapping of weight maps (based on all observations)

savebootweights save bootstraped weights (useful for combining across multiple iterations of predict())

bootsamples = 100 number of bootstrap samples to use

numcomponents = xxx: save first xxx components (for pca-based methods)

nopcr for cv_lassopcr and cv_lassopcrmatlab: do not do pcr, use original variables

lasso_num = xxx followed by number of components/vars to retain after shrinkage

hvblock = [h,v] use hvblock cross-validation with a block size of 'h' (0 reduces to v-fold xval) and number of test observations 'v' (0 reduces to h-block xval)

rolling = [h,v,g] use rolling cross-validation with a block size of 'h' (0 reduces to v-fold xval) and number of test observations 'v' (0 reduces to h-block xval), and a training size of g * 2 surrounding hv

verbose = 1 Set to 0 to suppress output to command window

platt_scaling calculate cross-validated platt scaling if using SVM. Softmax parameters [A,B] are in other_output{3}

Algorithm choices You can input the name (as a string array) of any algorithm with the appropriate inputs and outputs. i.e., this can either be one of the built-in choices below, or the name of another m-file. The format for algorithm functions is : [yfit, other_outputs] = predfun(xtrain, ytrain, xtest, optional_inputs) Each algorithm can take/interpret its own optional inputs. For bootstrapping of weights, algorithms MUST RETURN 3 OUTPUTS (programming 'feature')

To choose an algorithm, enter 'algorithm_name' followed by a text string with a built-in algorithm name, or a function handle for a custom algorithm Built-in algorithm choices include:

cv_multregress: [default] multiple regression

cv_univregress: Average predictions from separate univariate regression of outcome on each feature

cv_svr: Support vector regression with Spider package; requires spider

cv_pcr: Cross-validated principal components regression

cv_lassopcr: Cross-val LASSO-PCR; can enter 'lasso_num' followed by components to retain by shrinkage NOTE: can enter 'EstimateParams' to use shrankage lasso method based on the estimated optimal lambda that minimizes the mean squared error (MSE) of nested cross-validation models. Output of nested cv model is saved in stats.other_output_cv{:,3}. Output includes 'Lambda' parameter and min MSE value.

cv_lassopcrmatlab: Cross-val LASSO-PCR; can enter 'lasso_num' followed by components to retain by shrinkage NOTE: this uses the matlab implementation of LASSO, but can also run ridge or elastic net. Reduces to PCR when no lasso_num is entered by default. Use MSE for predicting continuous data and MCR for classifying binary data. NOTE: You can input any optional inputs that lassoglm takes. Enter 'Alpha', (0,1] as optional inputs to run ridge (Alpha approaches 0, but excluding 0), lasso (Alpha = 1), or elastic net (Alpha between 0 and 1) NOTE: Requires Matlab R2012a and higher. NOTE: Optional input: 'EstimateParams' - this will use grid search and nested cross validation to estimate Lambda and Alpha. Output is saved in stats.other_output_cv{:,3}. Output includes 'Alpha' parameter which is the elastic net mixture value between l1 and l2 regularization, 'Lambda' parameter, which is amount of LASSO regularization/shrinkage, and 'errorMatrix', which is the amount of error for each parameter combination. Use imagesc(obj.stats_other_output_cv{:,3}.errorMatrix) to view matrix. Min of this matrix is the best fitting parameters.

cv_svm: Cross-val support vector machine using Spider package NOTE: This is sensitive to scale of outputs! Use -1 , 1 NOTE: Optional inputs: Slack var parameter: 'C', 1 [default], 'C', 3 etc. Distance from hyperplane saved in stats.other_output_cv{:,2}. Recommend using the reordered cross-validated distance from hyperplane saved in stats.other_output{3} stats.dist_from_hyperplane_xval = cross-validated distance from hyperplane stats.weight_obj = voxel (variable) weight object e.g., orthviews(stats.weight_obj) Intercept for calculating dist from hy is in stats.other_output_cv{:,3} e.g., dist_hy = stats.weight_obj.dat' * obj.dat, where obj is a new set of test images NOTE: To run nonlinear SVM using radial basis function. Add 'rbf' followed by size of sigma (e.g., 2). NOTE: To estimate some of the parameters using nested cross validation add 'EstimateParams' as optional input. NOTE: To run multiclass SVM (i.e., one vs rest) add 'MultiClass' as optional input. Important - Obj.Y must be a matrix (data x class) with a column of 1 and -1 indicating each class. For example, if using 3 classes, then obj.Y must have 3 columns. NOTE: To run a balanced SVM where the number of cases for each class are unequal (i.e., one vs rest) add 'Balanced' as optional input, followed by a numerical value indicating the ridge amount (e.g., 0.01).

cv_multilevel_glm: Runs glmfit_multilevel. Must pass in ''subjIDs" followed by an array specifying which subject each trial belongs to Subjects' trials must all be "adjacent", i.e., don't put some of subject 1's trials at the beginning and other trials at the end – subjIDs does not

handle this case correctly. Also, 2ND LEVEL PREDICTORS NOT CURRENTLY SUPPORTED. code can be expanded to support this. mean-centering X and/or Y will NOT impact the predictor betas. Note that it WILL impact the intercept esimate as well as how much variance is explained (pred_outcome_r). Stratified CV partition not supported either, pass in custom holdout set.

Outputs

Y: Copy of outcome data to be predicted

algorithm_name: Name of algorithm; see options above

function_call: String of the command evaluated to call the prediction function

function_handle: Handle for the command evaluated to call the prediction function

yfit: Predicted outcome data (cross-validated)

err: Residuals/misclassification vector (cross-validated)

error_type: Name of error metric used for cverr

cverr: Cross-validated error

nfolds: Number of folds in stratified cross-validation, or vector of integers for membership in custom holdout set of each fold

- if k = 1, will estimate weights for full data object and not crossvalidate (useful for bootstrapping)

cvpartition: Cross-val partition object or structure with fold info

teIdx: Cell array of logical vectors with test samples in each fold

trIdx: Cell array of logical vectors with training samples in each fold

other_output: Other outputs returned by the algorithm; number and nature depend on algo choice; e.g., beta weights, svr weights, etc. For many algorithms, other_output{1} is a vector of weights on variables (e.g., voxels)

other_output_descrip: String description of other outputs

other_output_cv: Other outputs for each cross-validation fold

other_output_cv_descrip: Other output from algorithm - for each CV fold

mse: For regression only; mean squared error

rmse: For regression only; root mean squared error

meanabserr: For regression only; mean absolute error

pred_outcome_r: For regression only; prediction-outcome correlation

WTS: bootstrapped weights on voxels

weight_obj: for some algorithms, an fmri_data object with the predictive weights (from full sample)

Examples

```
obj = fmri_data;
obj.dat = randn(30, 50); %    30 voxels, 50 images (observations)
obj.Y = obj.dat' * rand(30, 1) + randn(50, 1); %    toy Y, linear combo of X plus noise
[cverr, stats, regression_outputs] = predict(obj);

Simulated example with 100 observations, 1000 voxels, with bootstrapping
```

```
    dat = fmri_data;
    dat.Y = rand(100, 1);
    dat.dat = repmat(dat.Y', 1000, 1) + 10*rand(1000, 100);
    [err,stats] = predict(dat, 'bootweights', 'algorithm_name', 'cv_lassopcr');

    [cverr, stats, regression_outputs] = predict(obj, 'nfolds', 3, 'error_type', 'meanabserr');
    [cverr, stats, regression_outputs] = predict(obj, 'algorithm_name', 'cv_univregress', 'error_typ
    [cverr, stats, optout] = predict(obj, 'algorithm_name', 'cv_lassopcr', 'lasso_num', 5, 'nfolds',
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_svm', 'nfolds', 5, 'error_type', 'ms
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_svm', 'rbf', 2, 'nfolds', 5, 'error_
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_svm', 'rbf', 2, 'EstimateParams', 'n
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_svm', 'nfolds', 5, 'MultiClass', 'er

    Elastic net with first 10 components:
    [cverr, stats, optout] = predict(dat_masked, 'algorithm_name', 'cv_lassopcrmatlab', 'nfolds', 5,

    Ridge with first 10 components:
    [cverr, stats, optout] = predict(dat_masked, 'algorithm_name', 'cv_lassopcrmatlab', 'nfolds', 5,

    Lasso with all components, but shrink to retain 2 components only:
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_lassopcrmatlab', 'nfolds', whfolds,
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_lassopcr', 'nfolds', whfolds, 'lasso

    Lasso with the shrinkage methods based on the estimated optimal lambda that minimizes MSE of nes
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_lassopcr', 'nfolds', whfolds, 'estim
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_lassopcr', 'nfolds', 5, 'estimatepar

    Lasso without doing PCR:
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_lassopcrmatlab', 'nfolds', whfolds,
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_lassopcr', 'nfolds', whfolds, 'lasso
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_lassopcr', 'nfolds', 5, 'estimatepar

    Lasso pcr using hvblock cross-validation on time-series, h = 3, v = 5;
    [cverr, stats, optout] = predict(dat, 'algorithm_name', 'cv_lassopcr', 'hvblock',[3,5]);

    Output display:
    orthviews(stats.weight_obj)
    line_plot_multisubject(stats.yfit, stats.Y, 'subjid', id_numbers);
```

See also predict_test_suite method for fmri_data, which runs predict with multiple options and summarizes output.

xval_regression_multisubject, xval_lasso_brain

`@fmri_data.`**`predict_test_suite`**(*dat*, *varargin*)

Run a set of cross-validated prediction algorithms on an fmri_data object and plot the outcome.

Usage

```
    [allcverr, allyhat] = predict_test_suite(dat, [optional inputs])
```

Functionality

- Requires matlab 2012a or later for full functionality
- Handles categorical or continuous outcomes automatically

Inputs

dat: an fMRI data object. dat.Y must be assigned, and must have continuous or binary outcomes assigned.

Optional

quick: Skip extended output

nfolds: Followed by number of folds or custom holdout vector (default = 5-fold balanced)

Examples

```
predict_test_suite(dat, 'nfolds', subjid);
```

`@fmri_data.`**`regress`**(*dat*, *varargin*)

Regression method for fmri_data object

Regress dat.X on dat.dat at each voxel, and return voxel-wise statistic images. Each column of dat.X is a predictor in a multiple regression, and the intercept is the last column. Intercept will automatically be added if not detected unless 'nointercept' is specified.

This function can also create a map of brain regions that predict the dat.Y vector using the 'brainony' option. This is essentially a univariate version of the 'predict' command. Warning: this is very slow as it loops through all voxels.

Regression is OLS by default, but can be robust using 'robust' flag.

Creates thresholded plot by default

Usage

```
out = regress(dat, varargin)
```

Inputs

dat: should be an fmri_data object with X field defined. dat.X can be a design_matrix() object.

Optional Inputs

[threshold, 'unc']: p-value threshold string indicating threshold type (see help statistic_image.threshold for options)

nointercept: Do not add intercept to model

nodisplay: Do not plot thresholded results using orthviews

brainony: univariate approach to predict obj.Y from brain data

residual: Output residual as fmri_data() object

noverbose: Suppress verbose outputs

robust: Run a robust regression (default is OLS). Robust is considerably slower than OLS

Outputs

out: A structure containing stats_img and fmri_data objects.

out.b: stats_img object of beta values estimated from regression

out.t: stats_img object of t-values with input threshold

out.df: fmri_data object of degrees of freedom

out.sigma: fmri_data object of variance of residual

out.residual: fmri_data object of residual data after model has been regressed out (optional).

Examples

```
% Run regression with liberal threshold
out = regress(dat, .05, 'unc');

% Run regression with conservative threshold and save residual
out = regress(dat, .001, 'unc', 'residual);

% Run robust regression with fdr threshold
out = regress(dat, .05, 'fdr','robust');

% Run a regression predicting behavior from brain at liberal threshold
out  = regress(data_comb, .05, 'unc', 'brainony')

% Re-threshold at different values
out.t = threshold(out.t, .05, 'fdr');
out.t = threshold(out.t, .001, 'unc');

% Re-display results of thresholding
orthviews(out.t);

% Write out beta image to current directory
out.b.fullpath = fullfile(pwd,'beta.nii');
write(out)
```

@fmri_data.**rescale**(*fmridat*, *meth*, *varargin*)

Rescales data in an fmri_data object Data is observations x images, so operating on the columns operates on images, and operating on the rows operates on voxels (or variables more generally) across images.

Usage

```
fmridat = rescale(fmridat, meth)
```

Inputs

Methods:

- centervoxels
- zscorevoxels
- centerimages
- zscoreimages
- rankvoxels
- windsorizevoxels
- percentchange
- tanh

Appropriate for multi-session (time series) only:

- session_global_percent_change
- session_global_z
- session_multiplicative

See also fmri_data.preprocess

@fmri_data.**saveplots**(*fmri_dat*, *varargin*)
Output dir

@fmri_data.**signtest**(*dat*, *varargin*)
Sign test for each voxel of an fmri_data object returns voxel-wise statistic images.

Usage

```
[out, statimg] = signtest(dat, [p-val threshold], [thresh_type])
```

Inputs

dat: Should be an fmri_data object with .dat field containing voxels x observations matrix

Optional inputs in [ ] above are **p-value threshold:**

string indicating threshold type (see help statistic_image.threshold for options)

Outputs

out: is a structure of information about the sign test

statimg:

is a statistic_image object that can be thresholded and plotted/imaged. statimg.dat contains signed direction values,

.p contains p-values

c Tor Wager, 2011 ..

See also: fmri_data.regress

@fmri_data.**ttest**(*fmridat*, *pvalthreshold*, *thresh_type*)
T-test on fmri_data class object

Usage

```
statsimg = ttest(fmridat, pvalthreshold, thresh_type)
```

Inputs

p-value threshold: p-value, e.g., .05 or .001 or [.001 .01 .05]

thresh_type: 'uncorrected', 'fwe', or 'fdr'

Examples

```
%T-test, Construct a stats_image object, threshold and display:
statsimg = ttest(fmridat, .001, 'unc');
orthviews(statsimg);

%Re-threshold and display:
statsimg = threshold(statsimg, .000001, 'unc');
orthviews(statsimg);

statsimg = threshold(statsimg, .01, 'fdr');
orthviews(statsimg);
```

Note for two-sample T-test, use fmri_data.regress

@fmri_data.**windsorize**(*obj*, *varargin*)

Windsorize an fMRI data object to madlimit Median Absolute Deviations. Default = 5 MADs. Works across rows and columns. Registers this step in history.

Usage

```
obj = windsorize(obj, [madlimit])
```

1.3 fmri_mask_image

@fmri_mask_image.**resample_to_image_space**(*obj*, *sampleto*, *varargin*)

Resamples data in an fmri_mask_image object (obj) to the space of another image (e.g., a functional image, for data extraction) The volInfo field will be the same as the sampleto volume info. The mask will have zeros in obj.dat for out-of-mask voxels. THIS FUNCTION USES SCN_MAP_IMAGE AND REQUIRES THAT THE ORIGINAL IMAGE BE AVAILABLE ON DISK. Multiple resamplings will break the function because the new space will be different from the original one on disk. Use the more general resample_space.

NOTE: Mask is *reloaded* from original data if space is remapped, and you cannot use manual thresholding of the mask. This is a feature of the map_to_image_space method and scn_map_image

Usage

```
obj = resample_to_image_space(obj, sampleto <img name or image_vector object>)
```

Inputs

obj: must be an fmri_mask_image object

sampleto:

can be either:

1. An image name to sample to
2. Another fmri_mask_image object (but image must exist on path!)

Optional inputs

mask: Apply sampleto as mask so that only voxels in the sampleto mask are retained in obj.dat.

THIS FUNCTION WORKS, BUT IS DEPRECATED BECAUSE RESAMPLE_SPACE IS MORE GENERAL. resample_space does not require the resampling of the original image from disk, which this does. resample_space is slower, though.

See Also: resample_space, for a method that does not require images to exist on disk on the path.

1.4 fmri_model

@fmri_model.**build**(*obj*)

Build the design matrix (xx) for an fmri_model object

We assume that the same conditions are modeled for each session. We assume that you have one basis set per condition (this is different from SPM, which only allows a single basis set across all conditions)

Usage

```
obj = build(fmri_model_obj)
```

@fmri_model.**build_single_trial**(*obj*, *inputhrf*)

Build a single-trial design matrix (xx) for an fmri_model object

We assume that the same conditions are modeled for each session We assume that you have one basis set per condition (this is different from SPM, which only allows a single basis set across all conditions)

This is used in single_trial_estimates, which assumes that you have estimated an initial model and saved image data.

The idea behind this is somewhat different from other canlab single-trial analyses, in that it takes in a single, custom HRF for each condition, rather than using a basis set. In single_trial_estimates, custom HRFs are created for each voxel by using the condition- and voxel-specific hrf estimates stored during model fitting.

The sequence would be:

1. robustfit(my_model), to fit average model and get HRF est for each voxel
2. single_trial_estimates(my_model), to use this function to build single-trial design matrices and fit them.

Usage

```
obj = build_single_trial(fmri_model_obj, inputhrf)
```

Inputs

inputhrf: should be a cell array of length nconds (number of conditions).

@fmri_model.**get_condition_assignments**(*obj*)

Condition assignments

- Indicator matrix coding for which columns in X belong to the same modeled condition, and are part of the same HRF fit
- There is one set of columns for each condition modeled, and one set of columns for each parametric modulator of each condition
- Because parametric modulators may not exist for all conditions, we need to build this dynamically for modulators.

Design matrix build (which calls method get_session_X) builds columns in this order:

All within Session: Regressors of interest, basis functions within conditions Parametric modulators, basis functions within conditions Covariates of no interest

Then: Baselines (session/run intercepts)

This method is called automatically in the build method.

@fmri_model.**get_session_X**(*obj*, *s*)

Get design matrix (predictors) for one session of fmri_model object, using basis functions defined in the object and onsets for one session (s).

Usage

```
[Xs, delta, C, B, names] = get_session_X(obj, session number)
```

@fmri_model.**plot**(*obj*)

Plot an fmri_model object

Usage

```
plot(obj)
```

@fmri_model.**replace_basis_set**(*obj*, *condition_num*, *xBF_hires*)

Replace a basis set in an fmri_model object with another one of your choosing.

This allows one to use a custom basis set, and also to use different basis sets for different trial types.

Each condition across all sessions must be modeled with the same basis set. That is, there can be only one basis set per condition, e.g., one for anticipation (used in each session) and one for pain.

Usage

```
obj = replace_basis_set(obj, condition_num, xBF_hires)
```

Examples

```
% generate a custom spline basis set and use that for Condition 1,
% and the standard one for Condition 2:

[xBF_hires, xBF] = fmri_spline_basis(2, 'length', 12, 'nbasis', 3, 'order', 3, 'plot');

%save this to get info that is not typically in basis set until after
%model is built.
```

@fmri_model.**robustfit**(*fmri_model_obj*, *fmri_data_obj*, *varargin*)

Robust fit for a model object to data object

Usage

```
robustfit(fmri_model_obj, fmri_data_obj, [optional args])
```

Features spatial smoothing of weights at 12 mm FWHM ridge regression ***not yet***

Preproc scaling

1. Remove covariates using ridge reg; ridge trace for full model
2. scale to % signal change across time (cols) OR rank time points (for w/i ss predictions??) AND/OR rank or center rows (images; for 'shape' analysis

Example %sig across time, rank across rows: relative % sig change

Different models of noise lead to different ideas about optimal preproc If large diffs in nuisance scaling in BOLD across individuals, ranking cols may be good idea. but then individual diffs in overall activity will be removed...

Options

tune: tuning const for robust reg

iter: 'maxiterations', robust reg /WLS iterations. 1 = OLS only!

smooth: 'spatial_smooth_fwhm', 0 or smoothing kernel for weights

nosmooth: spatial_smooth_fwhm = 0;

stats: 'calculate_stats', calculate_stats = 1; IN DEVELOPMENT

noresiduals: write_residuals = 0;

noplots: save_plots = 0;

@fmri_model.**rotate_to_pca**(*obj*)
Rotate design matrix columns within all conditions to principal component projection.

@fmri_model.**single_trial_estimates**(*obj*, *fmri_data_obj*)
Write single trial estimates associated with an estimated fmri_model object. must have estimated the model (robustfit(obj); see fmri_model.robustfit) and saved hrf*.img images for each condition.

Also input an fmri_data object with time series data.

This function writes images, one 4-D image for each condition, with the number of frames equalling the number of trials (onsets) for that condition.

It does this by constructing a separate design matrix for each voxel, which is based on the HRF estimates for that voxel for each condition. Fits for all conditions are added to the same model, so that their colinearity influences the single-trial parameter estimates.

1.5 fmridisplay

@fmridisplay.**addblobs**(*obj*, *cl*, *varargin*)
This is a method for fmridisplay objects that adds blobs to one or more montages and other surface plot(s).

Usage

```
obj = addblobs(obj, cl, varargin)
```

See addthreshblobs and multi_threshold methods for a multiple thresholds version render_blobs does most of the hard work.

Inputs

obj: an fmridisplay object

cl: a region object. If you're using an fmri_data object pass in region(fmri_data_obj)

Optional inputs

There are many optional inputs that control display features of blobs. These are determined by render_blobs

COLOR:

'color': followed by color vector, e.g., [0 1 1]

'maxcolor': followed by color vector for max color range, e.g., [0 1 1]

'mincolor': followed by color vector for min color range, e.g., [0 0 1]

'onecolor': force solid-color blobs

'splitcolor': Positive and negative values are mapped to different colormaps. Default is +=hot, -=cool colors. Followed optionally by cell array with vectors of 4 colors defining max/min for +/- range, e.g., {[0 0 1] [.3 0 .8] [.8 .3 0] [1 1 0]}

'OUTLINING:

**''outline'

'linewidth': followed by width value, e.g., 1

Note: add 'no_surface' to stop updating the existing surface blobs

'COLOR RANGE:

''cmaprange': followed by range of values, e.g., [0 40], [-3 3]. Used in color and transparency setting under some circumstances.

'TRANSPARENCY: {'trans', 'transparent','scaledtransparency', 'constanttrans', [val], 'transvalue', [val]}

'trans': Transparent blobs; with no other input, transparency = 0.75 (1 is opaque, 0 is transparent/invisible)

'scaledtransparency': Transparency is a function of voxel value, lower values are more transparent

'transvalue': Followed by width value, e.g., 1. also 'constanttrans'

Other Options

'smooth': Smooth blobs

'contour':

'no_surface': Do not add blobs to surface handles, if they exist

CONTROL OF WHICH MONTAGE

'wh_montages': followed by vector of montage numbers as they appear in the list of registered montages in the fmridisplay object

CONTROL OF WHICH SURFACE

'wh_surfaces':

followed by vector of surface numbers as they appear in the list of registered surfaces in the fmridisplay object

Examples

```
obj = addblobs(obj, cl, 'color', [0 1 1]);
obj = addblobs(obj, cl, 'color', [0 0 1], 'outline');
obj = addblobs(obj, cl, 'color', [0 1 0], 'outline', 'linewidth', 1, 'smooth');
obj = addblobs(obj, cl, 'color', [1 0 0], 'smooth', 'cmaprange', [0 40]);
obj = addblobs(obj, cl, ... 'wh_montages', 1);
obj = addblobs(obj, cl, 'splitcolor', ... 'cmaprange', ... 'trans');
```

Add only to montage 2 in vector of montages in obj.montage

```
obj = addblobs(obj, cl, 'which_montages', 2);
```

Map values to the colormap red->yellow.

This uses the default percentile-based mapping so that 20% of voxels will have the low color and 20% will have the high color, and the rest will be in between:

```
obj = addblobs(obj, cl, 'maxcolor', [1 1 0], 'mincolor', [1 0 0]);
```

Same, but now Map a specific range of values in image ([0 to .05])

```
obj = addblobs(obj, cl, 'maxcolor', [1 1 0], 'mincolor', [1 0 0], 'cmaprange', [0 .05]);
```

Separate positive and negative activations and map to a split colormap;

See render_blobs

```
o2 = addblobs(o2, cl, 'splitcolor', {[0 0 1] [.3 0 .8] [.8 .3 0] [1 1 0]}, 'wh_montages', 1);
```

It is possible to transparency-map values in a statistic image so you can show 'unthresholded' statistic values. e.g.:

```
    o2 = addblobs(o2, cl, 'splitcolor', {[0 0 1] [0 1 1] [1 .5 0] [1 1 0]}, 'cmaprange', [-2 2], 'tr
```

@fmridisplay.**addpoints**(*obj*, *xyz*, *varargin*)

Plots points on fmridisplay objects (e.g., montages of slices)

Usage

```
    newax = addpoints(obj, xyz, varargin)
```

Registers handles with the object (referred to as obj)

- enter xyz as n x 3 list of coordinates in mm to plot (world space)
- Points or text labels or both
- Flexible slice spacing, colors, marker sizes/styles, axis layout (one row/standard square)
- axial, saggital, or coronal orientation handled automatically
- Multiple different sets of points can be plotted in different colors/text labels

Optional Inputs

Takes all inputs of plot_points_on_slice. See help for additional documentation of options.

{'text', 'textcodes'}: cell array of text values corresponding to points

{'condf' 'colorcond'}: vector of integers to define color conditions

'close_enough': mm within which to plot; defined automatically based on slice distance if not entered

'color': string, 'b', or vector, [1 0 0], to define colors; cell if condf is used, e.g., {'b' 'g'}

{'marker', 'MarkerStyle'}: e.g., 'o', 'v', 's'

{'MarkerSize', 'markersize'}:

{'MarkerFaceColor', 'markerfacecolor'}: see color above

Examples

Plot points (i.e., coordinate locations) for xyz coords:

```
    o2 = addpoints(o2, DB.xyz, 'MarkerFaceColor', 'b', 'Marker', 'o', 'MarkerSize', 4);
    o2 = addpoints(o2, DB.xyz, 'text', DB.textcodes, 'condf', DB.condf, 'color', {'b' 'g'});
    o2 = removepoints(o2);
```

@fmridisplay.**addthreshblobs**(*obj*, *statimg*, *varargin*)

Usage

```
    obj = addthreshblobs(obj, statimg, varargin)
```

Add blobs from a statistical image object at multiple thresholds to montage and other surface plots.

It passes options specified in varargin onto addblobs, see help addblobs for options. In addition it has the parameters 'thresh' and 'pruneclusters' are used and have default values.

Inputs

obj: fmridisplay object, e.g. from a montage

statimg: statistics_image object

'thresh',{'fdr',0.01}: cell array of p-value thresholds in ascending order. can be 'fdr', or uncorrecetd p-value. defaults to:

thresh = {'fdr','0.001','0.01'}

'pruneclusters', [0 or 1]: Prune clusters that do not have at least one voxel surving at the most stringent threshold. Defaults to 1.

Examples

Add blobs from statimg at 3 significance levels [FDR, 0.001 unc., 0.01 unc.]. Prune clusters.

```
obj = addthreshblobs(obj, statimg);
```

Add blobs from statimg at 3 significance levels [FDR, 0.001 unc., 0.01 unc.]. Prune clusters.

```
obj = addthreshblobs(obj, statimg, 'thresh', {'fdr', 0.001, 0.01});
```

Add blobs from statimg at 2 significance levels [FDR, 0.001 unc.]. Do not prune clusters

```
obj = addthreshblobs(obj, statimg, 'thresh', {'fdr', 0.001},'pruneclusters',0);
```

set defaults for thresholds and cluster pruning

`@fmridisplay.`**`legend`**(*obj*, *varargin*)

Creates legend for fmridisplay object Adds legend axis handles to obj.activation_maps{:}

Usage

```
obj = legend(obj, varargin)

obj = legend(obj, 'figure') % new figure
```

`@fmridisplay.`**`montage`**(*obj*, *varargin*)

Creates montage of slices

- Solid brain slices or contour outlines
- Points or text labels or both
- Flexible slice spacing, colors, marker sizes/styles, axis layout (one row/standard square)
- axial or saggital orientation

Usage

```
obj = montage(obj, varargin)
```

Takes all inputs of plot_points_on_slice.

Optional Inputs

{'noslice', 'nodraw'}: drawslice = 0;

'color': color = varargin{i+1}; varargin{i+1} = [];

'marker': marker = varargin{i+1}; varargin{i + 1} = [];

{'wh_slice'}: wh_slice = varargin{i+1};

{'close', 'closeenough', 'close_enough'}: close_enough = varargin{i+1};

{'sagg','saggital','sagittal'}: orientation = 'sagittal';

{'MarkerSize', 'markersize'}: markersize = varargin{i+1};

{'MarkerFaceColor', 'markerfacecolor'}: facecolor = varargin{i+1};

'solid': disptype = 'solid';

'overlay': ovl = varargin{i + 1}; NOTE! DO NOT ENTER THIS HERE; ENTER WHEN YOU INITIALIZE FMRIDISPLAY OBJECT

{'text', 'textcodes'}: textcodes = varargin{i + 1};

{'condf' 'colorcond'}: condf = varargin{i + 1};

In addition:

'onerow': arrange axes in one row

'slice_range': [min max] values in mm for slices to plot

'wh_slice' or 'custom_coords':

followed my mm values for slices desired

e.g., for cluster/region centers, xyz = cat(1, cl.mm_center) o2 = montage(o2, 'axial', 'wh_slice', xyz, 'onerow'); o2 = montage(o2, 'saggital', 'wh_slice', xyz, 'onerow');

'spacing': followed by inter-slice spacing in mm

Outputs

obj: an fmridisplay object

Properties

- overlay: [1x105 char]
- SPACE: [1x1 struct]
- activation_maps: {[1x1 struct]}
- montage: {[1x1 struct]}
- surface: {}
- orthviews: {}
- history: {}
- history_descrip: []
- additional_info: ''

Examples:

```
o2 = fmridisplay; % create starting fmridisplay container object
```

Define new axes in existing figure, and use those for montage:

```
axh = axes('Position', [0.05 0.4 .1 .5]);
o2 = montage(o2, 'saggital', 'wh_slice', xyz(1,:), 'existing_axes', axh);

o2 = montage(o2, 'saggital', 'slice_range', [-10 10], 'onerow');
o2 = montage(o2, 'axial', 'slice_range', [-40 50], 'onerow', 'spacing', 4);
o2 = montage(o2, 'axial', 'slice_range', [-20 30], 'onerow', 'spacing', 8);
o2 = montage(o2, 'axial', 'wh_slice', xyz, 'onerow');

% Parasaggital only:
o2 = montage(o2, 'saggital', 'slice_range', [-4 4], 'onerow', 'spacing', 8);
```

Add/remove blobs and points with fmridisplay.addblobs, fmridisplay.addpoints, fmridisplay.removeblobs, fmridisplay.removepoints

See also

fmridisplay, cluster_orthviews, montage_clusters and variants

`@fmridisplay.`**`surface`**(*obj*, *varargin*)

Adds surfaces of brain to figure

Usage

```
obj = surface(obj, varargin)
```

Inputs

obj: fmridisplay

Outputs

obj: an fmridisplay object

Properties

- overlay: ''
- SPACE: ''
- activation_maps: {}
- montage: {}
- surface: {[1x1 struct]}
- orthviews: {}
- history: {}
- history_descrip: []
- additional_info: ''

Examples o2 = surface(o2, axes, [0.15 0.28 .15 1], 'direction', 'hires right', 'orientation', 'lateral');

See help fmridisplay

`@fmridisplay.`**`transparency_change`**(*o2*, *multval*)

Change the transparency of blobs in an fmridisplay object

Inputs

multval: multiply transparency values by this.

values < 1 makes blobs more transparent, > 1 makes blobs more opaque

1.6 fmridisplay_helper_functions

`fmridisplay_helper_functions.`**`clusters2mask2011`**(*cl*, *varargin*)

Returns 3-D mask of voxels included in a cl structure.

Usage

```
[mask, mask2] = clusters2mask2011(cl, [dim])
```

Mask values are coded with the cluster index. That is, the non-zero entries in mask are integers that reflect the index of the unique contiguous cluster to which each voxel belongs.

Any non-zero value indicates membership in a cluster Uses VOXEL values from cl, so define cl in the space you wish to have for mask first!

Optional A 2nd argument will be treated as 'dim', dimensions in voxels of the new mask image. If empty, uses max value in cluster to determine automatically, but then the mask may not match image dimensions desired for .img/.nii reading/writing purposes.

If a second output is requested, the second output (maskz) is a mask like the first, but the values in the mask reflect the numeric value stored in the cl.Z field (whether Z-scores or other values, depending on how cl is constructed.)

fmridisplay_helper_functions.**define_sampling_space**(*V*, *varargin*)

Define the sampling space of an image, with an upsampled space to 0.5 mm resolution

Usage

```
SPACE = define_sampling_space(V, [upsamplefactor])
```

Inputs

V: spm-style .mat structure, e.g., from spm_vol

V.mat: 4 x 4 matrix of voxel sizes and mm coords for the bottom back left vox

V.dim: dimensions of image

Outputs

Xo, Yo: Meshgrid for original voxel space

X, Y: Meshgrid for upsampled voxel space at 0.5 mm resolution

Xmm, Ymm: Meshgrid for upsampled space in mm

xcoords, ycoords: mm coordinates for rows and cols for slice locations

new_voxSize: new voxel size in mm for upsampled space

usfactor: Upsampling factor for new sampleing space

Examples

```
overlay = which('SPM8_colin27T1_seg.img');  % spm8 seg cleaned up
V = spm_vol(overlay);
SPACE = define_sampling_space(V)

% Define mm sampling space in original voxel coord resolution
SPACE = define_sampling_space(V, 1)

original (o) and new (X, Y) grid space
xcoords, ycoords: mm coords centered on origin
```

fmridisplay_helper_functions.**display_slice**(*dat*, *wh_slice*, *SPACE*, *varargin*)

Resample slice data in dat to SPACE and display

Usage

```
Z = display_slice(dat, wh_slice, SPACE, varargin)
```

fmridisplay_helper_functions.**map_to_world_space**(*V*)

Usage

```
SPACE = map_to_world_space(V)
```

Inputs

V: spm-style .mat structure, e.g., from spm_vol

V.mat: 4 x 4 matrix of voxel sizes and mm coords for the bottom back left vox

V.dim: dimensions of image

Outputs SPACE structure, with fields:

Xmm, Ymm, Zmm: Meshgrid for voxel volume in mm space

xcoords, ycoords, zcoords: mm coordinates for rows, cols, slices

fmridisplay_helper_functions.**render_blobs**(*currentmap*, *mymontage*, *SPACE*, *varargin*)

This is a helper function for fmridisplay objects, called by the addblobs method

Usage

```
[blobhan, cmaprange, mincolor, maxcolor] = render_blobs(currentmap, mymontage, SPACE, varargin)
```

See fmridisplay.m and addblobs.m method in fmridisplay for more details and options.

Inputs

currentmap: see addblobs method. Montage within fmridisplay object.

mymontage: ditto

SPACE: space of map to sample to (object display SPACE in fmridisplay object)

Optional Inputs There are many optional inputs that control display features of blobs.

COLOR:

'color': followed by color vector, e.g., [0 1 1]

maxcolor': followed by color vector for max color range, e.g., [0 1 1]

mincolor': followed by color vector for min color range, e.g., [0 0 1]

onecolor': force solid-color blobs

splitcolor': Positive and negative values are mapped to different colormaps. Default is +=hot, -=cool colors. Followed optionally by cell array with vectors of 4 colors defining max/min for +/- range, e.g., {[0 0 1] [.3 0 .8] [.8 .3 0] [1 1 0]}

OUTLINING:

'outline'

'linewidth': followed by width value, e.g., 1

COLOR RANGE:

'cmaprange': followed by range of values, e.g., [0 40], [-3 3]. Used in color and transparency setting under some circumstances.

TRANSPARENCY:

{'trans', 'transparent','scaledtransparency', 'constanttrans', [val], 'transvalue', [val]}

'trans': Transparent blobs; with no other input, transparency = 0.75 (1 is opaque, 0 is transparent/invisible)

'scaledtransparency': Transparency is a function of voxel value, lower values are more transparent

'transvalue': Followed by width value, e.g., 1. also 'constanttrans'

OTHER OPTIONS:

'smooth': Smooth blobs

'contour':

Orientation: 'sagittal', 'coronal', 'axial'

Outputs [blobhan, cmaprange, mincolor, maxcolor]

All used in addblobs.m

Use addblobs; do not run this function directly unless you are programming with it.

See also: fmridisplay/addblobs, fmridisplay, fmridisplay/multi_threshold

`fmridisplay_helper_functions.`**`resample_space`**(*dat*, *V*, *targetsp*)

Usage

```
[resampled_dat, SPACEto] = resample_space(dat, V, [target V or target SPACE])
```

Inputs

imdat: 3-D volume data

V: spm-style .mat structure for dat, e.g., from spm_vol

V.mat:

4 x 4 matrix of voxel sizes and mm coords for the bottom back left vox

V.dim: dimensions of image

targetsp:

target V: spm-style .mat structure defining space to transform to

—OR –

target SPACE: target SPACE, with Xmm, Ymm, Zmm; see map_to_world_space.m

Outputs

resampled_dat: data sampled in new space

SPACE structure, with fields:

Xmm, Ymm, Zmm: Meshgrid for voxel volume in mm space

xcoords, ycoords, zcoords: mm coordinates for rows, cols, slices

V: Vol info structure for image in new space

1.7 image_vector

@image_vector.**apply_mask**(*dat*, *mask*, *varargin*)

Apply a mask image (image filename or fmri_mask_image object) to an image_vector object stored in dat.

This can be used to:

- Mask an image_vector or fmri_data object with a mask
- Obtain "pattern expression" for a weight map (entered as the mask, here) in a series of images stored in dat.

The mask or weight map does not have to be in the same space as the dat; it will be resampled to the space of the data in dat.

To extract pattern expression values for each ROI within a mask use extract_roi_averages()

Optional Inputs

pattern_expression: calculate and return the cross-product of each image in dat and the values in the mask. This is useful if comparing expression values that are comprised of different datasets or differing number of voxels.

correlation: calculate the pearson correlation coefficient of each image in dat and the values in the mask.

norm_mask: normalize the mask weights by L2 norm, for patt expression only.

ignore_missing: use with pattern expression only. Ignore weights on voxels with zero values in test image. If this is not entered, the function will check for these values and give a warning.

invert: Invert the mask so that out-of-mask voxels are now in (using the mask as an 'exclude mask' rather than an include-mask. If pattern expression is requested, the behavior is different, and it inverts the sign of in-mask pattern weights.

Examples

```
[dat, mask] = apply_mask(dat, mask)
[dat, mask] = apply_mask(dat, mask image name)
[dat, mask] = apply_mask(dat, mask image vector object)
[pattern_exp_values] = apply_mask(dat, weight map image, 'pattern_expression', 'ignore_missing')
[pattern_exp_values] = apply_mask(dat, weight map image, 'pattern_expression', 'ignore_missing',
```

See also

extract_roi_averages, to get individual region averages / local pattern expression apply_nps, which does whole-pattern and local regional expression

@image_vector.**check_image_filenames**(*obj*, *varargin*)

Check whether images listed in obj.fullpath actually exist

Usage

```
obj = check_image_filenames(obj, ['noverbose'])
```

Behavior

- If there are no file names, do nothing.
- If file names are entered and full path is not, attempt to find full path.

- If full path info is entered, check to see if files exist. Return output in obj.files_exist, and print a warning if only some exist.

Image names should be stored in .fullpath abbreviated image names may be stored in image_names.

Note

fullpath should have full path to each volume in a string matrixm, with trailing ,volume# for 4-D images as per SPM style expanded list.

image_names should have image name only for each volume

`@image_vector.`**`compare_space`**(*obj*, *obj2*)

Compare spaces of two image_vector objects

Usage

```
function isdiff = compare_space(obj, obj2)
```

Returns 0 if same, 1 if different spaces, 2 if no volInfo info for one or more objects. 3 if same space, but different in-mask voxels in .dat or volInfo.image_indx

`@image_vector.`**`extract_gray_white_csf`**(*obj*)

Extracts mean values (values) and top 5 component scores (components) from each of gray, white, and CSF masks. Images must be in standard MNI space for this to apply.

Usage

```
[values, components] = extract_gray_white_csf(obj)
```

Inputs

obj: an image_vector (e.g., fmri_data) object

`@image_vector.`**`extract_roi_averages`**(*obj*, *mask*, *varargin*)

This image_vector method a extracts and averages data stored in an fmri_data object from a set of ROIs defined in a mask. It is *slightly* different from the fmri_data method, as fmri_data has more fields.

This version requires the mask_image to be in the same space as the obj.

Regions to average over can be either regions of contiguous voxels bounded by voxels with values of 0 or NaN, which are considered non-data values, or regions defined by unique integer codes in the mask image (i.e., for atlas images with unique codes for each defined region.)

Mask/Atlas image does NOT have to be in the same space as the images to extract from. It will be remapped/resliced.

Extracted data is returned in single data format.

Usage

```
cl = extract_roi_averages(image_vector obj, mask, [average_over])
```

Inputs

- char array of strings containing 4D image file names (data extracted from these)
- mask_image to extract from.

Optional inputs

average_over:

- Default: 'contiguous_regions' to average over contiguous voxels bounded by voxels of 0 or NaN (non-data values)
- Alt. option = 'unique_mask_values' to average over unique integer codes in the mask image (i.e., for atlas images with unique codes for each defined region)

Examples

```
imgs_to_extract_from = filenames('w*.nii','char');
mask_image = which('anat_lbpa_thal.img');
[cl, imgdat] = extract_image_data(imgs_to_extract_from, mask_image);
```

See also

For an non-object-oriented alternative, see extract_image_data.m

`@image_vector.`**`fastmontage`**(*dat*, *varargin*)

Creates 3 separate montage views - ax, cor, sagg in a special figure window

Usage

```
fastmontage(dat, [myview], ['spacing', slicespacing], ['vertical'])
```

Examples

```
fastmontage(dat);
fastmontage(dat, 'coronal');
fastmontage(dat, 'saggital', 'spacing', 10);
fastmontage(dat, 'saggital', 'spacing', 10, 'vertical');
fastmontage(dat, 'saggital', 'slices_per_row', 12);
```

`@image_vector.`**`flip`**(*dat*, *varargin*)

Flips an image_vector object left to right

Optional Inputs Input 'mirror' to make a symmetrical image, averaging the left and right hemispheres

Examples

```
dat = flip(dat, ['mirror'])
```

`@image_vector.`**`get_wh_image`**(*dat*, *wh*)

For an image_vector with multiple images (cases, contrasts, etc.), select a subset.

Usage

```
function obj_out = get_wh_image(obj1, wh)
```

Inputs

obj1: An image_vector object

wh: An array indicating which images

Examples

```
my_image_vector = get_wh_image(dat, 3) %to get 3rd image
my_image_vector = get_wh_image(dat, [1 3]) %to get 1st and 3rd image
```

check that wh is in range

@image_vector.**histogram**(*obj*)

@image_vector.**history**(*dat*)
Display history for image_vector object

@image_vector.**horzcat**(*varargin*)
Implements the horzcat ([a b]) operator on image_vector objects across voxels. Requires that each object has an equal number of columns and voxels

Usage

```
function s = horzcat(varargin)
```

Examples

```
c = [dat1 dat2];
```

@image_vector.**ica**(*fmridat_obj*, *varargin*)

Spatial ICA of an fmri_data object

- icadat = ica(fmridat_obj, [number of ICs to save])
- icadat is also an fmri_data object, with .dat field voxels x components

Notes

- icasig = W * mixedsig
- icasig = icadat.dat' = W * fmridat_obj.dat'

A is scaled version of fmridat_obj.dat' * icadat.dat

A and W are stored in additional_info field of icadat

@image_vector.**image_math**(*obj1*, *varargin*)
Perform simple mathematical and boolean operations on image objects

Usage

```
obj_out = image_math(obj1, [optional inputs, e.g., a 2nd object, keywords])
```

For objects: Type methods(object_name) for a list of special commands Type help object_name.method_name for help on specific methods.

Inputs

obj1: An image_vector object

Optional Inputs

obj2: An additional image_vector object

{'add', 'plus'}:

Keyword to perform image-wise addition of images in obj1 and obj2. Assumes these are paired/matched objects.

{'subtract', 'minus'}:

Keyword to perform image-wise subtraction of images in obj1 and obj2

{'cat', 'concatenate'}:

Concatenate obj1 and obj2 image-wise. Requires same number of voxels in both image sets. Returns effects codes of 1, -1 in obj_out.Y.

{'power'}:

Keyword to raise data to power element-wise; obj.dat = obj.dat.^b; Followed by exponent to apply (b)

Outputs

obj_out: The result - an image_vector object

DEFAULTS AND INPUTS ..

`@image_vector.`**`image_similarity_plot`**(*obj*, *varargin*)

Point-biserial correlations between images in fmri_data obj and set of 'spatial basis function' images (e.g., 'signatures' or pre-defined maps)

Usage:

```
stats = image_similarity_plot(obj, 'average');
```

This is a method for an image_vector object

Inputs

obj: An image object with one or more images loaded

Optional inputs

average: Calculate average over images in obj with standard errors Useful if obj contains one image per subject and you want to test similarity with maps statistically. Default behavior is to plot each individual image.

bucknerlab Use 7 network parcellation from Yeo et al. as basis for comparisons

kragelemotion Use 7 emotion-predictive models from Kragel & LaBar 2015 for basis of comparisons

compareGroups Perform multiple one-way ANOVAs with group as a factor (one for each spatial basis); requires group as subsequent input

group Indicates group membership for each image

noplot Omits plot (print stats only)

Outputs

stats:

Structure including:

- .r, Correlations in [7 networks x images in obj] matrix
- .t, T-test (if 'average' is specified)
- .line_handles Handles to polar plot lines so you can customize
- .fill_handles Handles to polar plot fills so you can customize
- .table_spatial, ANOVA table with subject as row factor and spatial basis as column factor (one way repeated measures ANOVA, requires 'average' to be specified)
- .multcomp_spatial, multiple comparisons of means across different spatial bases, critical value determined by Tukey-Kramer method (see multcompare)

table_group multiple one-way ANOVA tables (one for each spatial basis) with group as column factor (requires 'average' to be specified)

multcomp_group mutiple comparisons of means across groups, one output cell for each spatial basis, critical value determined by Tukey-Kramer method (see multcompare)

Examples

```
% corrdat is an fmri_data object with 18 images from searchlight
% correlation in it.  Then:
stats = image_similarity_plot_bucknermaps(corrdat, 'average');

% t_diff is a thresholded statistic_image object
stats = image_similarity_plot_bucknermaps(t_diff);
```

See also

tor_polar_plot

List dates and changes here, and author of changes 11/30/2015 (Phil Kragel)

•added anova (rm) comparing means across spatial bases

•added anova (1-way) comparing means across groups for each spatial basis (e.g., for each buckner network)

12/15/2015 (Phil Kragel)

- added option to omit plotting

DEFAULTS AND INPUTS ..

`@image_vector.`**`image_similarity_plot_bucknermaps`**(*obj*, *varargin*)

Point-biserial correlations between images in fmri_data obj and Bucker Lab 7-network maps, with polar plot

Usage

```
stats = image_similarity_plot_bucknermaps(obj, 'average');
```

This is a method for an image_vector object

Inputs

obj: An image object with one or more images loaded

Optional inputs

average: Calculate average over images in obj with standard errors Useful if obj contains one image per subject and you want to test similarity with maps statistically. Default behavior is to plot each individual image.

Outputs

stats:

Structure including:

- .r, Correlations in [7 networks x images in obj] matrix
- .t, T-test (if 'average' is specified)
- .line_handles Handles to polar plot lines so you can customize
- .fill_handles Handles to polar plot fills so you can customize

Examples

```
    % corrdat is an fmri_data object with 18 images from searchlight
    % correlation in it.  Then:
    stats = image_similarity_plot_bucknermaps(corrdat, 'average');

    % t_diff is a thresholded statistic_image object
    stats = image_similarity_plot_bucknermaps(t_diff);
```

See also

tor_polar_plot

DEFAULTS AND INPUTS ..

@image_vector.**interpolate**(*dat*, *varargin*)

Interpolate over missing values in image_vector object

Usage

```
    dat = interpolate(dat, varargin)
```

Input image_vector object (dat; e.g., an fmri_data object)

Use when there are some missing values in the mask image Performs 3-D linear interpolation to fill in all values in the original mask.

e.g., For a standard brain image space that is 91 x 109 x 91, you may have 300,000 in-mask values. Only 150,000 of these may be defined in the image, however, and the rest are missing (0 or NaN). This function will return a dat image with non-missing values for all 300,000 voxels (the "in-mask" space). It will not return values for all voxels in the 91 x 109 x 91 space, however.

Note

This function does not upsample the data now, but could be extended to do so fairly easily.

@image_vector.**mean**(*obj*, *varargin*)

Create an image_vector object with mean values for each voxel (cols) across images (rows) of an fmri_data object.

Usage

```
    function m = mean(obj, [optional args])
```

m is an image_vector object whose data contains the mean values.

Optional Inputs

- 'write', followed by file name
- 'path', followed by location for file (default = current directory)
- 'orthviews' -> show orthviews for this image, same as orthviews(m)
- 'histogram' -> show histogram for this image, same as histogram(m)
- 'plot' -> do both

Examples

```
    % If sdat is an fmri_data object with multiple images,
    m = mean(sdat, 'plot', 'write', anatmeanname, 'path', maskdir);
```

@image_vector.**minus**(*obj1*, *obj2*)
Implements the minus (-) operator on image_vector objects across voxels. Requires that each object has an equal number of columns and voxels

@image_vector.**montage**(*image_obj*, *varargin*)
Create a montage of an image_vector (or statistic_image or fmri_data) object

*Usage:

```
[fig_handle or o2 fmridisp object] = montage(image_obj, [optional arguments])
```

Optional inputs

fmridisplay: for fmridisplay object style montage [default]

scnmontage: for circa 2008-style SCN lab montage for each image vector

Examples

```
o2 = montage(mask);
```

@image_vector.**orthviews**(*image_obj*, *varargin*)
Orthviews display (SPM) for CANlab image_vector (or fmri_data, statistic_image) object

*Usage:

```
orthviews(image_obj, varargin)
```

Optional Inputs

posneg: input generates orthviews using solid colors.

largest_region: to center the orthviews on the largest region in the image

@image_vector.**plot_current_orthviews_coord**(*dat*)
Retrieves and plots the image data series at the current crosshairs in spm_orthviews

@image_vector.**plus**(*obj1*, *obj2*)
Implements the plus (+) operator on image_vector objects across voxels. Requires that each object has an equal number of columns and voxels

Examples

```
c = dat1 + dat2;
```

@image_vector.**power**(*obj*, *b*)
Implements the power (^) operator on image_vector objects across voxels.

Examples

```
c = dat1^2;
```

Programmer Notes: Created 3/14/14 by Luke Chang ..

@image_vector.**preprocess**(*obj*, *meth*, *varargin*)
Preprocesses data in an fmri_data object

Data is observations (i.e., voxels, subjects) x images, so operating on the columns operates on images, and operating on the rows operates on voxels (or variables more generally) across images.

Inputs meth: Options

resid: Residualize voxels with respect to covariates Uses obj.covariates, obj.dat. Adds intercept automatically. You can tell it to add the mean response per voxel back in: obj = preprocess(obj, 'resid', [add mean back in flag])

hpfilter: High-pass filter and remove run intercepts and first two images per run. Uses obj.dat, obj.images_per_session obj = preprocess(obj, 'hpfilter', HPlen in s, TR)

windsorize: Windsorize entire data matrix to 3 STD

windsorizevoxels: Windsorize each time series in data matrix to 3 STD

session_outliers: Identify session-wise (run-wise) outliers with significant based on mahalanobis distance with FDR-corrected P-values in chi-square test. Impute session grand mean outliers.

outliers: Identify outlier time points for each session based on mahalanobis distance (see above) across global mean for slices and spatial STD for slices, as in scn_session_spike_id. Outliers at 3 SD based on timeseries added to obj.covariates.

outliers_rmssd: Identify outlier time points for each session based on root-mean-square successive differences between images (across voxels.) this is the std (across voxels) of the successive diffs across images. Outliers at 3.5 SD based on timeseries added to obj.covariates.

smooth:

Smoothed images with Gaussian filter

- obj = preprocess(obj, 'smooth', FWHM in mm)

NOTE SMOOTHING KERNEL MAY BE IN VOX, AS VOL INFO IS NOT PASSED IN

interp_images: Interpolate all voxels in a series of images specified by logical vector whout.

- obj = preprocess(obj, 'interp_images', whout);

Examples

```
% two complementary ways to get and plot outliers:
dat = preprocess(dat, 'outliers', 'plot');
subplot(5, 1, 5); % go to new panel...
dat = preprocess(dat, 'outliers_rmssd', 'plot');
```

@image_vector.**read_from_file**(*obj*)

Reads data from image filenames into obj.dat

Try obj = check_image_filenames(obj) first.

This is automatically called if you create a new image_vector object with names but do not directly enter data. e.g., the commands below will load data:

•name = 'salientmap.nii';

•img = image_vector('image_names', name);

@image_vector.**rebuild_volinfo_from_dat**(*dat*, *newdat*)

Will rebuild volInfo (the image space, or sometimes "mask") from a vectorized image. In other words, will rebuild dat.volInfo from newdat.

Also resets all voxels to be significant, if a statistic image

Input

dat: an image_vector

newdat: a vector that MUST be size of ENTIRE image (dat.volInfo.nvox)

Output

dat: dat.dat contains the non-zero values of newdat, and dat.volInfo is correctly defining the image space

`@image_vector.`**`reconstruct_image`**(*obj*)

Reconstruct a 3-D or 4-D image from image_vector object obj

voldata is and X x Y x Z x Images matrix vectorized_voldata is the same, with all voxels vectorized

This output has one element for every voxel in THE ENTIRE IMAGE, and so can be very memory-intensive. But it's useful for lining up voxels across images with different masks/in-mask voxels.

This function returns output in memory; see image_vector.write for writing .img files to disk.

Outputs

voldata: 3-D recon volume

vectorized_voldata: volume in column vetor, iimg_xxx function format

xyz_coord_struct:

has fields with coordinate information in mm (world) space

- x, y, z : vectors of coordinates in mm for each of the 3 dimensions of the image
- X, Y, Z : output matrices from meshgrid with mm coordinates, for volume visualization. These can be passed to surf or isocaps functions for volume visualization in world space (mm).

`@image_vector.`**`remove_empty`**(*dat*, *varargin*)

remove vox: logical vector of custom voxels to remove, VOX x 1

remove im: logical vector of custom images to remove, 1 x IMAGES

indices of removed data will be stored in removed_voxels and removed_images fields, to preserve ability to later reconstruct into 3D images

Usage

```
dat = remove_empty(dat, [logical vector of custom voxels to remove], [logical vector of imgs to
```

Indicator vectors stored in: removed_images removed_voxels

See also replace_empty

force logical

`@image_vector.`**`reparse_contiguous`**(*obj*, *varargin*)

Re-construct list of contiguous voxels in an image based on in-image voxel coordinates. Coordinates are taken from obj.volInfo.xyzlist.

Results are saved in obj.volInfo.cluster.

xyzlist can be generated from iimg_read_img, and is done automatically by object-oriented fMRI image classes (fmri_image, image_vector, statistic_image)

If 'nonempty' is entered as an optional argument, will use only voxels that are non-zero, non-nan in all columns of obj.dat.

Usage

```
obj = reparse_contiguous(obj, ['nonempty'])
```

.cluster and .xyzlist should both always be length v in-mask voxels if 'nonempty' is entered, then .dat should be length v in-mask voxels too

@image_vector.**replace_empty**(*obj*, *varargin*)

Replace empty/missing values in an image data object

Usage

```
obj = replace_empty(obj, [optional keywords])
```

Replace missing values in obj.dat stored in obj.removed_voxels and obj.removed_images with zeros. This returns obj.dat in a format that can be reconstructed into a 3-D or 4-D image matrix for brain visualization.

Optional keywords

'voxels' or 'images': replace only missing voxels/images

See also remove_empty, zeroinsert, nanremove, naninsert

@image_vector.**resample_space**(*obj*, *sampleto*, *varargin*)

Resample the images in an fmri_data object (obj) to the space of another image (sampleto; e.g., a mask image). Works for all image_vector objects.

Usage

```
obj = resample_space(obj, sampleto, [sampling method])
```

Sampleto may be one of these:

1. a volInfo structure (the image does not have to exist on the path)
2. an image_vector, fmri_data, fmri_mask_image object
3. a string with the name of an image

Can enter resampling method as optional input. Takes any input to interp3:

'nearest' - nearest neighbor interpolation 'linear' - linear interpolation (default) 'spline' - spline interpolation 'cubic' - cubic interpolation as long as the data is uniformly

spaced, otherwise the same as 'spline'

Examples

```
label_mask = fmri_data(which('atlas_labels_combined.img'));
label_mask = resample_space(label_mask, ivec, 'nearest') % resamples and masks label image
```

@image_vector.**resample_time**(*obj*, *source_TR*, *target_TR*, *varargin*)

Resample the time-series images (source_time_interval) in an fmri_data object (obj) to the different time series (target_time_interval). Works for all image_vector objects.

•obj = resample_time(obj, source_time_interval, target_time_interval, varargin)

Optional Inputs

meth (Interpolation methods):

You can enter resampling method as optional input. Takes any input to

- 'nearest' - nearest neighbor interpolation
- 'linear' - linear interpolation (default)
- 'spline' - spline interpolation

- **'cubic' - cubic interpolation as long as the data is uniformly** spaced, otherwise the same as 'spline'

slice: A fraction of the slice timing correction. The default is 0.5, meaning if your TR is 2s, the time point of your TR image will be considered as the middle point of the TR bins. You can use this option to use different time points. If you are upsampling your data (i.e., your target TR is shorter than your source TR), you need to discard the first column of your data. This function will return the first time point data as NaN.

Examples

```
dat = fmri_data('/Volumes/RAID1/labdata/current/BMRK3/Imaging/spatiotemp_biomarker/STmarker1.img
dat = resample_time(dat, 2, 1.3)

% with options:
dat = resample_time(dat, 2, 1.3, 'meth', 'linear', 'slice', .3)
```

`@image_vector.`**`sagg_slice_movie`**(*dat*, *varargin*)

Movie of successive differences (sagittal slice) Enter an image_vector or fmri_data object (usually with time series)

Usage

```
sagg_slice_movie(dat, [full_path_of_movie_output_file,image_skip_interval])
```

*Optional Inputs:

movie_output_file:

a char array detailing the full path to save the movie file

image_skip_interval:

An integer value describing the interval between images in each subsequent frame of the movie

(default = 1)

Examples

```
 sagg_slice_movie(fmri_dat, ...
            '/Volumes/engram/labdata/fmri_data/Study1/Subj1/qc_images', 5)

This would save an movie based on the images in fmri_dat to the
above directory, with an interval of 5 images between each
frame (so, the movie would show image 1, 6, 11, 16, etc)
```

`@image_vector.`**`searchlight`**(*dat*, *varargin*)

Run searchlight multivariate prediction/classification on an image_vector or fmri_data object OR two objects, for cross-prediction.

Usage

```
[list outputs here] = function_name(list inputs here, [optional inputs])
[results_obj, stats, indx] = searchlight(dat, [optional inputs])
```

Features

- Runs searchlight with standard, pre-defined algorithms
- Custom-entry definition of holdout sets

- Can re-use searchlight spheres after initial definition
- Custom-entry definition of any spheres/regions of interest
- Uses Matlab's parallel processing toolbox (parfor)

Type help image_vector.searchlight to display this help information

Inputs

dat: image_vector or fmri_data object with data

dat.Y: required: true outcomes for each observation (image) in dat

:Optional Inputs:* Keyword followed by input variable:

r: searchlight radius, voxels

dat2: second dataset, for cross-prediction

indx: sparse logical matrix. each COLUMN is index of inclusion sets for each region/sphere in searchlight This takes a long time to calculate, but can be saved and re-used for a given mask

holdout_set: Followed by integer vector of which observations belong to which holdout set, for cross-validation. This is passed into fmri_data.predict.m. Default is empty.

Outputs

results_obj: fmri_data object with results maps

stats: selected statistics for each sphere in searchlight

indx: sparse logical matrix. each COLUMN is index of inclusion sets for each region/sphere in searchlight * this can be re-used for all data with the same mask/structure. *

Examples

```
% Define a sensible gray-matter mask:
dat = fmri_data(which('scalped_avg152T1_graymatter.img'));
dat = threshold(dat, [.8 Inf], 'raw-between');
dat = trim_mask(dat);

% Create fake data and holdout indicator index vector
dat.dat = randn(dat.volInfo.n_inmask, 30);
dat.Y = dat.dat(111111, :)' + .3 * randn(30, 1);
holdout_set = ones(6, 1); for i = 2:5, holdout_set = [holdout_set; i*ones(6, 1)]; end

% Run, and run again with existing indx
pool = parpool(12);  % initialize parallel processing (12 cores)
[results_obj, stats, indx] = searchlight(dat, 'holdout_set', holdout_set);
results_obj = searchlight(dat, 'holdout_set', holdout_set, 'indx', indx);
```

See also

region.m, fmri_data.predict.m

DEFAULTS AND INPUTS ..

`@image_vector.`**`slices`**(*obj*, *varargin*)

Create a montage of single-slice results for every image in an image_vector object

Usage

```
    o = slices(obj, 'orientation', [orientation], 'slice', [slice_mm], 'nimages', [nimgs])
```

obj is an image_vector, fmri_data, or statistic_image object with multiple images (only the first 64 will display), which are stored as columns in its .dat field.

Optional Inputs

orientation: can be followed by 'saggital', 'axial', or 'coronal'

slice_mm: is followed by the mm coord of the slice to display; default = 0

nimgs: can be followed by the number of images to display, 1:nimgs

names: is followed by a cell array of names for the images.

color: is followed by color vector or string specification. default is color-mapped with split colors (hot/cool) for pos and neg effects.

outline: is followed by a color vector for outline around blobs.

The output, o, is an fmridisplay object.

This function uses fmridisplay objects, and may be memory-intensive for older computers.

Common Errors:

This function uses the volInfo.cluster field. If you create a mask in an ad hoc way, this field may not be updated. use this to fix:

•mask = reparse_contiguous(mask);

Examples

```
    slices(dat);
    slices(dat, 'orientation', 'axial');
    slices(dat, 'slice', -5);                  % display sagg at x = -5
    o = slices(dat, 'names', terms); % use 'terms' var as names

    o2 = slices(all_chi2_images, 'orientation', 'saggital', 'slice', 0);
```

`@image_vector.`**`surface`**(*obj*, *varargin*)

[all_surf_handles, pcl, ncl] = surface(obj)

Usage:

- [all_surf_handles, pcl, ncl] = surface(r, ['cutaways', any optional inputs to surface_cutaway])

This function uses region.surface to create surface figures. See help region.surface for options.

Examples

```
    % Create an initial surface plot from an fmri_data object:
    han = surface(regionmasks{2});

    % Now add a second region in green:
    cluster_surf(region(regionmasks{2}), {[0 1 0]}, han, 5);

    % Use optional arguments taken by surface_cutaway:
    poscm = colormap_tor([1 .3 0], [1 1 0]); % orange to yellow
    [all_surf_handles, pcl, ncl] = surface(t, 'cutaway', 'ycut_mm', -30, 'pos_colormap', poscm, 'exi
    [all_surf_handles2, pcl, ncl] = surface(t, 'foursurfaces', 'pos_colormap', poscm, 'neg_colormap'
    [all_surf_handles2, pcl, ncl] = surface(t, 'foursurfaces', 'existingfig', 'color_upperboundperce
```

```
    % Use mediation_brain_surface_figs and re-make colors
    all_surf_handles = mediation_brain_surface_figs([]);
    surface(t2, 'cutaway', 'surface_handles', all_surf_handles, 'color_upperboundpercentile', 95, 'c
```

@image_vector.**threshold**(*obj*, *input_threshold*, *thresh_type*, *varargin*)

Threshold image_vector (or fmri_data or fmri_obj_image) object based on raw threshold values. For statistical thresholding, convert to a statistic_image object and see the threshold method for that object.

Usage

```
    obj = threshold(obj, input_threshold, thresh_type, [optional arguments])
```

This is a method for an image_vector object

Thresholding is not reversible. For statistic_image objects it is.

For objects: Type methods(object_name) for a list of special commands Type help object_name.method_name for help on specific methods.

Inputs

obj: image_vector object

input_threshold: Vector of 2 values defining data value bounds at which to threshold, e.g., [0 Inf] or [-3 3]

thresh_type: String: 'raw-between' or 'raw-outside'

Optional Inputs Argument or argument followed by value:

k: Followed by extent threshold cluster size, default = 1

trim_mask: Reduce the mask in obj.voInfo based on thresholding

noverbose: Suppress verbose output

Outputs

obj: thresholded image_vector object

Examples

```
    % Retain positive values, cluster extent > 100 voxels
    obj = threshold(obj, [0 Inf], 'raw-between', 'k', 100)

    % Retain voxels with absolute value > 3
    obj = threshold(obj, [-3 3], 'raw-outside')
```

See also

statistic_image.threshold, statistic_image.multi_threshold

@image_vector.**trim_mask**(*obj*)

Exclude empty voxels from mask information in obj.volInfo structure, and re-make obj.volInfo

@image_vector.**union**(*dat1*, *dat2*, *outputname*)

Union and intersection masks for two image_vector objects

Usage

```
    [dat_union, dat_intersection] = union(dat1, dat2, outputname)

     dat = union(dat1, dat2, outputname)
     outputname = character array name for union image
                  INCLUDE .img at the end.
```

@image_vector.**write**(*obj*, *varargin*)

Write an image_vector object to an Analyze image. Option to write thresholded image, for statistic_image objects.

obj.dat should contain data, with one COLUMN for each 3-D frame in the 4-D image to be written.

Usage

```
    write(obj)  -> writes to the image(s) specified in obj.fullpath
    write(obj, 'thresh') -> for statistic_image objects, writes thresholded
    write(obj, 'fname', '~/Documents/test.nii')  -> writes the image(s) to specific path
```

Optional Inputs

mni: resample image to standard MNI FOV (91x109x91) uses mri_data.resample_space

keepdt: output image will be keep original data type (default = float32)

fname: writes out image to specific file name. 'fname' must be followed by image name with path

Examples

```
    % If m is an image_vector object m.X(m.X < .12) = 0; % apply an
    % arbitrary but reasonable custom threshold
    orthviews(m);

    % write the thresholded image to disk:
    anatmeanname = 'mean_gray_matter_mask.img';
    m.filename = anatmeanname;
    m.fullpath = fullfile(maskdir, anatmeanname);
    write(m)
```

1.8 region

@region.**check_extracted_data**(*cl*)

Checks the data, just in case of space/programming issues, by re-extracting the region average data from 5 random regions using spm_get_data.m, and compares it to the already-saved values

Inputs

cl: must be a valid region object (see region.m) and cl(1).source_images must still be on the path.

You should not need to run this regularly – but you should if you suspect things have gone awry.

@region.**extract_data**(*r*, *data_obj*)

Extract data from image_vector object (data_obj) for voxels specified by a region object (r). Returns extracted data and averages.

Usage

```
region_obj = extract_data(region_obj, data_obj)
```

Type methods(region) for a list of special commands for region object Type help object_name.method_name for help on specific methods.

Features data_obj does not have to be in the same space, uses mm coordinates

Inputs

r: a region object

data_obj: an image_vector or fmri_data object to extract data from does not have to be in the same space, uses mm coordinates

Outputs

r: a region object, with data attached

`@region.`**`merge`**(*cl*, *wh_merge*)

Merge two or more regions together in a region object. Combines fields from all clusters in the named series with the first one in the series.

Usage

```
wh_merge = [3 4];
cl = merge(cl, wh_merge)
```

`@region.`**`posneg_separate`**(*cl*, *varargin*)

Separate a region object (cl) into clusters with positive and negative peak values, based on max (peak) value in .val or .Z field (default = val)

Usage

```
[pcl, ncl] = posneg_separate(cl, ['Z'])
```

Returns pcl and ncl, region structures with positive- and negative-valued peaks, respectively, copied from the original cl input.

Optional Input

Z: To use .Z field

Note You may have to use reparse_continguous to get this to work right.

```
r = reparse_continguous(r);
[pcl, ncl] = posneg_separate(r);
```

`@region.`**`region2imagevec`**(*cl*)

Convert a region object to an image_vector object, replacing the voxels and reconstructing as much info as possible.

The .dat field of the new "ivecobj" is made from the cl.all_data field. if this is empty, uses cl.val field, then cl.Z as a backup. Mask information is available in ivecobj.volInfo.

Usage

```
ivecobj = region2imagevec(cl)
```

`@region.`**`region2imagevec2tmp`**(*cl*)

Convert a region object to an image_vector object, replacing the voxels and reconstructing as much info as possible.

The .dat field of the new "ivecobj" is made from the cl.all_data field. if this is empty, uses cl.val field, then cl.Z as a backup. Mask information is available in ivecobj.volInfo.

Usage

```
ivecobj = region2imagevec(cl)
```

@region.**region2struct**(*cl*)

Convert a region object to a simple structure, primarily for compatibility with other, older CANlab tools.

See also cluster2region, for the reverse transformation

@region.**reparse_continguous**(*cl*)

Re-define regions in region object based on contiguous blobs

Usage

```
clout = reparse_continguous(cl)
```

@region.**subdivide_by_atlas**(*r*, *varargin*)

Usage

```
r = subdivide_by_atlas(r, [atlas name])
```

Inputs

r: a region object, defined using region(mask)

atlas name: Optional mask image with integer codes defining in-mask regions. Default is 'atlas_labels_combined.img'

Output A region object with separate clusters for each contiguous blob, subdivided by regions labeled in atlas.

Example

```
r = subdivide_by_atlas(r);
r(cat(1, r.numVox) < 20) = []; % get rid of small regions
cluster_orthviews(r, 'unique');
```

@region.**subdivide_by_local_max**(*r*, *varargin*)

Subdivide regions into sub-regions based on local peak Z-scores/maxima

Usage

```
subregions = subdivide_by_local_max(r, ['mm_distance', value], ['noorthviews'])
```

For objects: Type methods(object_name) for a list of special commands Type help object_name.method_name for help on specific methods.

Optional Inputs

mm_distance: Followed by mm distance minimum for dividing subclusters

noorthviews: Suppress display of orthviews

Outputs

subregions: subdivided region object

See also

region.subdivide_by_atlas, subclusters_from_local_max, cluster_local_maxima

@region.**surface**(*r*, *varargin*)

Surface method for region object - renders blobs on multiple types of 3-D surface

Usage

```
[all_surf_handles, pcl, ncl] = surface(r, ['cutaways', any optional inputs to surface_cutaway])
```

Inputs

r: A region object

cutaway:

String command for rendering cutaways instead of the default

- default is call to mediation_brain_surface_figs
- cutaways calls surface_cutaway
- all optional arguments are passed to surface_cutaway

rightsurface:

String command for rendering a right frontal cortical view complementary to 'cutaways'

foursurfaces: Compact plots of four surfaces

Other optional inputs to surface_cutaway e.g., 'pos_colormap'

Outputs

all_surf_handles: surface patch handles

pcl: region object with positive-only clusters

ncl: region object with negative-only clusters

Example

```
% Use surface(r), with optional arguments taken by surface_cutaway:
poscm = colormap_tor([1 .3 0], [1 1 0]); % orange to yellow
[all_surf_handles, pcl, ncl] = surface(r, 'cutaway', 'ycut_mm', -30, 'pos_colormap', poscm, 'exi
[all_surf_handles2, pcl, ncl] = surface(r, 'foursurfaces', 'pos_colormap', poscm, 'neg_colormap'
[all_surf_handles2, pcl, ncl] = surface(r, 'foursurfaces', 'existingfig', 'color_upperboundperce

% use mediation_brain_surface_figs and re-make colors
all_surf_handles = mediation_brain_surface_figs([]);
surface(r, 'cutaway', 'surface_handles', all_surf_handles, 'color_upperboundpercentile', 95, 'cc
```

:See also:*

surface_cutaway, cluster_surf, mediation_brain_surface_figs ..

DEFAULTS AND INPUTS

@region.**table**(*cl*, *varargin*)

Print a table of all regions in a region object (cl)

Usage

```
[poscl, negcl] = table(cl, [optional inputs])
```

Optional inputs

k: Print only regions with k or more contiguous voxels

nosep: do not separate cl with pos and neg effects based on peak in .val

names: name clusters before printing to table and output; saves in .shorttitle field

forcenames: force naming of cl by removing existing names in .shorttitle field

Outputs Returns region objects for cl with pos and neg effects, limited by size if entered and named if entered as optional input

1.9 statistic_image

`@statistic_image.`**`conjunction`**(*si1*, *si2*, *varargin*)

Returns the conjunction of two statistic_images. considers positive and negative activations separately.

Inputs Two thresholded statistic images. Optional 3rd argument: -1 to

get only negative conjunction, or 1 to get only positive conjunction

Output A statistic_image with all voxels suprathreshold (in the same direction) in both input

images. Voxel values are set to 1 and -1, to indicate direction.

`@statistic_image.`**`convert2mask`**(*stats_image_obj*)

Converts each image in a statistic_image object into a mask object, based on significant voxels in the .sig field.

Example

```
cl = region(convert2mask(timg), group)
```

`@statistic_image.`**`multi_threshold`**(*dat*, *varargin*)

Multiple threshold function for statistic_image object for visualization

Usage

```
[o2, sig, pcl, ncl] = multi_threshold(dat, [optional inputs])
```

Inputs

dat: a statistic_image object

Optional Inputs

poscolors: followed by cell array of colors for positive values, one per thresh

negcolors: followed by cell array of colors for negative values, one per thresh

thresh: followed vector of p-value thresholds, one per thresh

sizethresh:

followed by vector of cluster sizes, one per thresh

- this 'prunes' by default, so sizes after first can be 1 voxel
- Default thresholds: thresh = [.001 .005 .05], 10 voxels at .001, "pruned"

nodisplay: suppress fmridisplay

o2:

followed by an existing fmridisplay object

- will remove blobs and re-use montages

Outputs

o2: handle to fmridisplay object created by default

sig:

vector of significant voxels at each thresh, for each region

- cell array of images in object with matrix of values

for each threshold

pcl:

positive valued clusters cell, one cell per threshold

- FIRST image in object only
- pass into mediation_brain_surface_figs.m

ncl:

positive valued clusters cell, one cell per threshold

- FIRST image in object only
- pass into mediation_brain_surface_figs.m

Examples

```
[o2, sig, poscl, negcl] = multi_threshold(hr_intercept, 'nodisplay');
mediation_brain_surface_figs(poscl, negcl);

% Create empty montage set and (re)use it:
o2 = canlab_results_fmridisplay([], 'compact2', 'noverbose');
o2 = multi_threshold(out.t, 'o2', o2);
```

See also

mediation_brain_surface_figs, iimg_multi_threshold, mediation_brain_results

`@statistic_image.`**`orthviews`**(*image_obj*, *varargin*)

Orthviews display (SPM) for CANlab object

Usage

```
cl = orthviews(image_object)

% OR

cl = orthviews(image_object, handle_number of existing orthviews)
```

Output is clusters structure (see also region.m)

Pass in 'largest_region' to center the orthviews on the largest region in the image

Example

```
% T-test, Construct a stats_image object, threshold and display:
statsimg = ttest(fmridat, .001, 'unc');

% Re-threshold and display:
statsimg = threshold(statsimg, .000001, 'unc');
```

```
    orthviews(statsimg);

    statsimg = threshold(statsimg, .01, 'fdr');
    orthviews(statsimg);

    % Create an orthviews and view at multiple thresholds in different panes:
    overlay = which('SPM8_colin27T1_seg.img');
    spm_check_registration(repmat(overlay, n, 1));
    statsimg = ttest(fmridat);
    statsimg = threshold(statsimg, .001, 'unc');
    orthviews(statsimg, 'handle', 1);

    statsimg = threshold(statsimg, .000001, 'unc');
    orthviews(statsimg, 'handle', 2);
```

See also statistic_image.multi_threshold

@statistic_image.**reparse_contiguous**(*obj*, *varargin*)

Re-construct list of contiguous voxels in an image based on in-image voxel coordinates. Coordinates are taken from obj.volInfo.xyzlist. Results are saved in obj.volInfo.cluster. xyzlist can be generated from iimg_read_img, and is done automatically by object-oriented fMRI image classes (fmri_image, image_vector, statistic_image)

Usage

```
    obj = reparse_contiguous(obj, ['nonempty'])
```

If 'nonempty' is entered as an optional argument, will use only voxels that are non-zero, non-nan in the first column of obj.dat.

The statistic_image object version of reparse_contiguous uses the significance of the first image in the object (obj.sig(:, 1)) as a filter as well, so clustering will be based on the latest threshold applied. it is not usually necessary to enter 'nonempty'.

Example

```
    % Given timg, a statistic_image object:
    test = reparse_contiguous(timg, 'nonempty');
    cl = region(test, 'contiguous_regions');
    cluster_orthviews(cl, 'unique')
```

@statistic_image.**select_one_image**(*obj*, *wh*)

@statistic_image.**threshold**(*stats_image_obj*, *input_threshold*, *thresh_type*, *varargin*)

Threshold statistic_image object based on statistical threshold values.

Usage

```
    stats_image_obj = threshold(stats_image_obj, pvalthreshold or other thresh, thresh_type, ['k', e
```

This is a method for an statistic_image object. Thresholding is reversible.

For objects: Type methods(object_name) for a list of special commands Type help object_name.method_name for help on specific methods.

Inputs

stats_image_obj: statistic_image object

input_threshold: [pvalthreshold or other thresh] A numeric value corresponding to the threshold desired. Either a p-value or a range of raw values, depending on the threshold type.

thresh_type:

Threshold type which can be one of:

- 'fdr' : FDR-correct based on p-values already stored in image .p field
- 'fwe' : FWE-correct; not implemented
- 'bfr' : Bonferroni correction (FWE).
- 'unc' : Uncorrected p-value threshold: p-value, e.g., .05 or .001
- 'raw-between' : threshold raw image values; save those > input_threshold(1) and < input_threshold(2)
- 'raw-outside' : threshold raw image values; save those < input_threshold(1) or > input_threshold(2)

Optional Inputs

k: Followed by cluster extent in voxels: extent-based thresholding of any of the above

noverbose: Suppress verbose output

mask:

Followed by name of mask or fmri_mask_image object

- this will affect corrected significance levels

Outputs

stats_image_obj: thresholded statistic_image object

Example

```
% Retain sig pos or neg results at p < .001 uncorrected, cluster extent >= 100 voxels
obj = threshold(obj, .001, 'unc', 'k', 100)

% Retain sig pos or neg results at q < .05 FDR, cluster extent >= 10 voxels
obj = threshold(obj, .05, 'fdr', 'k', 10)

% Retain voxels with absolute statistic/data value > 3
obj = threshold(obj, [-3 3], 'raw-outside')

dat = threshold(dat, 0.001, 'unc', 'k', 35, 'mask', which('scalped_avg152T1_graymatter_smoothed.
dat = threshold(dat, 0.001, 'unc', 'k', 35, 'mask', maskobj);
```

See also

image_vector.threshold, statistic_image.multi_threshold

CHAPTER 2

Toolboxes

2.1 Cluster_contig_region_tools

Cluster_contig_region_tools.**anat_subclusters**(*cl*, *varargin*)

Clusters voxels within 'clusters' structure based on anatomical locations in space. Outputs subgroups of smaller clusters.

Usage

```
clout = anat_subclusters(cl,[resume at],[output cl to resume])
```

Cluster_contig_region_tools.**cluster2region**(*cl*)

Transform a CANlab/SCANlab "clusters" structure into a region object, the standard in 2011 toolbox functions and beyond.

Cluster_contig_region_tools.**cluster2subclusters**(*cl_in*, *class*)

Take a single cluster cl_in and separate into subclusters based on vector of integers class

Class must code unique subclusters subcluster order is only preserved if class contains all integers from 1 to nclasses:

i.e., class 3 will only be in subcluster 3 if there are no missing class numbers in class

Cluster_contig_region_tools.**cluster_close_enough**(*cl_match_to*, *cl_match*, *mind*)

Finds whether each cluster center in cl_match is within mind mm of a cluster center in cl_match_to.

Useful for selecting a list of clusters that are not close to another list to, e.g., make a table of. or this could be used to find clusters in a set of correlated clusters that are close to centers in activated clsuters.

Cluster_contig_region_tools.**cluster_export_pngs**(*cl*, *useexisting*, *overlay*, *xhairson*)

Save png images of SPM orthviews windows for each cluster in a set (cl structure)

Usage

```
cl = cluster_export_pngs(cl,[useexisting],[overlayimagename],[xhairson])
```

names from cl(x).shorttitle are used useexisting is optional: 1 uses existing orthviews display (default), 0 creates a new one with the clusters

Example

```
% use existing
cluster_export_pngs(cl, 1, EXPT.overlay);
```

Cluster_contig_region_tools.**cluster_find_index**(*cl*, *varargin*)

Ever see an interesting blob when visualizing a clusters structure, but don't know which index number in the clusters structure vector it corresponds to?

With this function, find the index number of the closest cluster to one you specify graphically by clicking on.

Usage

```
function [wh_cluster, min_distance] = cluster_find_index(cl, [keep display flag, 1/0])
```

Cluster_contig_region_tools.**cluster_interp**(*cl*, *varargin*)

Interpolates voxels and mm in a clusters structure (cl) to match the image dimensions and voxel sizes of a target mask image.

Usage

```
function cl = cluster_interp(cl,maskimg,[keep sep clusters flag])
```

Example

```
cl = cluster_interp(cl,maskimg,1);

%default mask 2 x 2 x 2, SPM2 default:
cl = cluster_interp(cl,[],1);
```

maskimg = which('scalped_avg152T1_graymatter_smoothed.img');

Cluster_contig_region_tools.**cluster_intersection**(*varargin*)

Computes the intersection of the clusters passed in.

Usage

```
intersect_cl = cluster_intersection(cl1, cl2, cl3, ...)

% simple intersection
cl = cluster_intersection(robust0001_poscl(2), robust0002_poscl(4), robust0003_poscl(17));

% intersection between sets of clusters
% alternatively, see cluster_set_intersection.m
cl = cluster_intersection(clusters2CLU(robust0001_poscl), clusters2CLU(robust0002_poscl));
```

Note: Only works with single clusters. To compute the intersection between sets of clusters, use cluster_set_intersection()

Cluster_contig_region_tools.**cluster_local_maxima**(*cl*, *dthresh*, *verbose*)

Clusters are chosen so that they must be at least dthresh mm apart default is 10 mm

Usage

```
[xyz, XYZmm, Z, class] = cluster_local_maxima(cl, [dthresh], [verbose])
```

verbose output: 1/0, default is 0

additional optional outputs (slower):

class: vector of integers for which subcluster this cluster belongs to

Cluster_contig_region_tools.**cluster_set_intersection**(*varargin*)

Computes the intersection of the sets of clusters passed in.

Usage

```
    intersect_cl = cluster_set_intersection(cls1, cls2, cls3, ...)
    cl = cluster_intersection(robust0001_poscls, robust0002_poscls, robust0003_poscls);
```

Note: Designed for sets of clusters. To compute the intersection between individual clusters, use cluster_intersection(). cluster_set_intersection will work, but is not needed.

Cluster_contig_region_tools.**cluster_table**(*clusters*, *varargin*)

Print output of clusters in table

Option to print text labels from Carmack atlas.

Database loading is done from talairach_info.mat which should be in the path.

To speed up performance, load talairach_info.mat in the base workspace or calling function and include xyz, L3 and L5 as inputs to cluster_table.

Example

```
    % create subclusters on the fly, prompt for labels
    cluster_table(cl);

    % no subclusters, no labels
    cluster_table(cl, 0, 0);

    % do subclusters, no labels
    cluster_table(cl, 1, 0);

    create subclusters on the fly, do labels
    cluster_table(cl, 1, 1);

    % 3 input variables following 'tal_info' are interpreted as xyz, L3,
    % and L5 from talairach_info.mat.
    cluster_table(..., 'tal_info', xyz, L3, L5);

    % loads labels from taldata.mat (Talairach database) instead of
    % talairach_info.mat. Note that you should use the 'tal_info' call
    % above if xyz, L3, and L5 have already been loaded to theworkspace
    % from taldata.mat. Also, if the talairach database is being used,
    % your cl.XYZmm values MUST correspond to the TALAIRACH, NOT MNI,
    % database, or the labels will be innaccurate.
    cluster_table(..., 'talairach');

    % print table to ASCII file, 'filename', instead of to the matlab
    % command window.
    cluster_table(..., 'writefile','filename');

    % any set of inputs from above, also print clusters.myfield in output
    cluster_table(..., 'myfield');
```

Cluster_contig_region_tools.**cluster_table_successive_threshold**(*cl*, *varargin*)

Cluster table of a cell array of clusters cl{1} cl{2} etc. Prints table of cl{1} and then any additional regions in cl{2:n} that are not within 10 mm of a previously printed cluster

Also: merges clusters in set within 10 mm

Table titles are hard-coded to be consistent with meta-analysis toolbox right now

Usage

```
cl = cluster_table_successive_threshold(cl,[sizethr])
```

Example

```
% Print a series of tables with custom fields:
cl = cluster_table_successive_threshold(cl,5,'myfield1','myfield2')
```

Use *merge_nearby_clusters* and *subclusters_from_local_max* or some other way to get clusters appropriately separated and distanced before running. see Meta_cluster_tools for code to run this for meta-analysis.

Cluster_contig_region_tools.**clusters2CLU**(*clusters*, *varargin*)

Inputting an M matrix will transform the coordinates by that M, to convert between voxel sizes, etc.

Usage

```
function CLU = clusters2CLU(clusters,[opt] M)
```

Cluster_contig_region_tools.**clusters2mask**(*cl*, *V*, *varargin*)

This function has 2 modes! If V is a structure:

Usage

```
[m,V,cl] = clusters2mask(cl,V,[opt: write Z-scores])
```

Converts clusters structure to a mask image, given V structure with V.mat field. V.mat is an SPM mat file. V.dim is dims of image uses cl.XYZmm m is mask img data, V is mask vol info.

Also replaces cl.XYZ (voxels)

If V is a vector of mask dimensions: converts clusters to mask image using existing XYZ and dims of mask

See also voxels2mask, for a faster function that uses XYZ voxel coords

Example

```
% Save an image file with just one cluster from a set (#7 in this ex.)
cl = mask2clusters('roi_group1.img');
V = spm_vol('roi_group1.img');  % we need .mat and .dim from this, or

% just dim
[m,V,cl] = clusters2mask(cl(7),struct('mat',cl(1).M,'dim',V.dim));
%or
[m,V,cl] = clusters2mask(cl(7),V);

clusters2mask(cl,struct('mat',V.mat,'dim',V.dim),0,'spm2_hy.img');

%for SPM5:
clusters2mask(cl,struct('mat',V.mat,'dim',V.dim, 'dt', V.dt),0,'spm2_hy.img');
clusters2mask(cl,
struct('mat',MC_Setup.volInfo.mat,'dim',MC_Setup.volInfo.dim, 'dt', MC_Setup.volInfo.dt),0,'acc_
```

Cluster_contig_region_tools.**image2clusters**(*varargin*)

Menu-driven function for getting clusters from an image file (e.g., a t-image)

Usage

```
cl = image2clusters([overlay image name])
```

Can also return clusters active in two contrasts, sorted by increases in both, decreases in both, inc in first, dec in first useful for testing whether something is both activated and correlated! e.g., see active_plus_corr_scatterplot_plugin

% :Example:

```
[pospos,negneg,posneg,negpos] = image2clusters(overlay)
```

Cluster_contig_region_tools.**mask2clusters**(*P*, *varargin*)

Extracts clusters and con img data from mask

Use with *mask_intersection.m*

To get clusters but not extract data, enter only one argument.

To get clusters and choose extraction imgs with the GUI, enter an empty [] 2nd argument.

Usage

```
[clusters,CLU,subclusters] = mask2clusters(img mask file with voxels,[imgs to extract data from]
```

DOES *NOT* CONVERT BETWEEN DIFFERENT VOXEL SIZES AND POSITIONS BETWEEN IMNAMES AND SPM/VOL STRUCTS

See also roi_probe

If no imgs are entered, Z-scores are values from mask

If df is entered, values in mask img are converted to Z-scores with spm_t2z.m

If extract img names are empty and df is entered, assume we're using values from mask as t-values and convert to Z-scores

WARNING: for spm2 compatibility, ABSOLUTE VALUES of voxel sizes are returned; e.g., ignores analyze flipping in SPM2.

% Matlab 6.5/OSX bug gives seg fault or something if mask is too big.

Example

```
cl = mask2clusters('myimage.img',[img string mtx],[]); % no z-score
conversion, extracts data from [img string mtx]

cl = mask2clusters('rob_tmap_0002_filt_t_3-05_k10_neg.img')

% This one works with already-loaded image data and a mat matrix:
V = spm_vol('rob_tmap_0002_filt_t_3-05_k10_neg.img'); dat = spm_read_vols(V);
cl = mask2clusters(dat,V.mat);
```

Cluster_contig_region_tools.**mask2struct**(*maskname*, *varargin*)

Usage

```
function V = mask2struct(maskname,crit_t,cl_size)
```

Inputs

maskname: name of spmT, con, or filtered image without .img extension, in single quotes

Optional Inputs

crit_t, cl_size:: critical t and cluster size at which to mask

Outputs structure compatible with SPM viewing and with cluster definition algorithm tor_extract_rois

Example

```
    % to extract clusters:
    [clusters] = tor_extract_rois(maskname,V,V);

    % to display:
    spm_image   (and choose anatomical)
    spm_orthviews('AddBlobs',1,V.XYZ,V.Z,V.mat)
    spm_orthviews('AddColouredBlobs',1,V.XYZ,V.Z,V.mat,[0 0 1])

    % to overlay on Talairach atlas
    fixed_TSU(clusters)
```

Cluster_contig_region_tools.**merge_clusters**(*c2m*, *subcl*)

Function for synchronizing the field list of cluster structures and merging them

Usage

```
    subclusters = merge_clusters(clusters_to_match, subclusters_to_change)
```

Cluster_contig_region_tools.**merge_nearby_clusters**(*cl*, *thr*, *varargin*)

Merge sets of clusters whose centers are all within thr mm of each other uses parcel_complete_sets.m

Usage

```
    newcl = merge_nearby_clusters(cl, thr)
```

Example

```
    % The command below runs the function recursively until all clusters are
    % greater than thr mm apart
    newcl = merge_nearby_clusters(cl, thr, 'recursive')
```

Cluster_contig_region_tools.**subclusters_from_local_max**(*cl*, *dist_thresh*)

Breaks apart a cluster into smaller clusters

Usage

```
    subcl = subclusters_from_local_max(cl, dist_thresh)
```

Cluster_contig_region_tools.**xyz2clusters**(*xyz*, *P*)

Converts a 3-column x, y, z list of mm coordinates to a clusters structure given P, the filename of an analyze .img file to provide dimensions and voxel sizes.

Usage

```
    function cl = xyz2clusters(xyz,P)
```

Uses this info from the image:

- VOL.M - spm-style mat matrix
- VOL.VOX - voxel sizes
- SPM.Z - now 1s; could stores values in the original image in clusters.Z

The following is created internally:

- SPM.XYZmm - mm coords, you input these
- SPM.XYZ - voxel coords

2.2 diagnostics

diagnostics.**BiasPowerloss**(*tc*, *X*, *c*, *beta*, *df*, *z*, *pval*)

Calculate the approximate bias and power loss due to mis-modeling This works with the Mismodeling Toolbox described by Loh et al. 2008

Usage

```
[b bias pl Pc Pe] = BiasPowerloss(tc, X, c, beta, df, z, pval)
```

Inputs

tc: fMRI time course

X: design matrix for multiple regression

c: contrast of interest

beta: (mismodeled) beta value

df: degrees of freedom

z: p-value calculated from ResidScan

pval: cut-off p-value

Outputs

b: updated (correct) beta value

bias: bias

pl: power loss

References Loh, J. M., Lindquist, M. A., Wager, T. D. (2008). Residual Analysis for Detecting Mis-modeling in fMRI. Statistica Sinica, 18, 1421-1448.

Update design matrix using correct model

diagnostics.**ResidScan**(*res*, *FWHM*)

Calculates P(M>=t) where M is the max value of the smoothed residuals. In this implementation the residuals are smoothed using a Gaussian kernel.

Usage

```
function [z sres] = ResidScan(res, FWHM)
```

Inputs

res: residual time course

FWHM: Full Width Half Maximum (in time units)

Outputs

z: pvalues

sres: smoothed residuals

sres_ns: smoothed residuals (non standardized)

diagnostics.**add_nuisance_to_SPMcfg**(*Xn*)

Adds a matrix Xn to the end of covariates of interest in xX structure in SPMcfg.mat

Usage

```
function add_nuisance_to_SPMcfg(Xn)
```

Inputs

oXn: should contain ALL nuisance covariates and intercepts as in output of tor_get_physio.m

This function is automatically run by tor_get_physio.m

diagnostics.**batch_efficiency**(*dwcard*)

Start in directory above individual model/results directories

Usage

```
function batch_efficiency(dwcard)
```

Inputs

dwcard: is a wildcard for directories to probe, e.g., 'subject*'

diagnostics.**batch_t_histograms**(*varargin*)

Creates page(s) of t stat histograms for each subject level contrast in set of subject level analyses using image_intensity_histograms.

Usage

```
batch_t_histograms([options])
```

Optional Inputs

{analysis_dirs}:

run on all contrasts in directories of cell array {analysis_dirs}

(DEFAULT: use all directories in working directory containing spmT_*.img files)

'o', 'output_directory':** specify output directory to contain saved .png files

diagnostics.**canlab_qc_metrics1**(*epi_names*, *mask_name*, *varargin*)

Calculate quality control metrics on a 4-D EPI Analyze or Nifti file

Standard CANlab usage is to enter a single 4-D ravol* for one run, and the brain mask implicit_mask.img created in canlab_preproc.m

Inputs

epi_names: Names of (usually one) 4-D EPI file, in cell or string, full path

mask_name:

Name of brain mask image, string, full path IF EMPTY: Uses implict masking (better) and calculates ghost/signal outside mask

Optional Inputs

noplot: skip plots

noverbose: skip output printout to screen

printfile: followed by name of file to print output to, full path

noheader: suppress printing of header (var names) in output

Missing values and basic info

num_images: number of images

missing_vox: Voxels in mask with NaN values or zero values at every time point

missing_images: Images with NaN values or zero values at every voxel

missing_values: NaN or zero values in valid images / voxels. Zeros could be interpreted as values of zero in analysis, causing artifacts in results if these are actually invalid values.

Missing voxels will often be ignored in analyses in most software, but Missing images/values could cause problems

Basic signal to noise

perc-mean-ghost: mean signal outside the mask / mean total signal

mean_snr:

mean Cohen's d (signal/noise, SNR) across time (temporal SNR) within the mask.

Mean divided by standard deviation across time at each voxel, averaged. Higher is better.

snr_inhomogeneity: standard deviation of SNR within the mask. Lower is better.

snr_inhomogeneity95: 95% confidence range for SNR within the mask. Lower is better.

rms_successive_diffs: Essentially a high-pass filtered version of SNR, expressed as a fraction of the overall mean and averaged across voxels. Lower is better.

rms_successive_diffs_inhomogeneity: standard deviation of the above across voxels. Lower is better.

Left-right asymmetry

signal_rms_asymmetry: Voxel-wise left/right root mean square asymmetry in mean signal across time, expressed as a fraction of the mean SNR. Reflects both gross inhomogeneity and noise. Lower is better.

signal_hemispheric_asymmetry: Root mean square difference between left and right hemispheres, expressed as a fraction of the grand mean signal across time. Reflects gross inhomogeneity. Lower is better.

snr_rms_asymmetry: Voxel-wise left/right root mean square asymmetry in SNR, expressed as a fraction of the mean SNR. Reflects both gross inhomogeneity and noise. Lower is better.

snr_hemispheric_asymmetry: Root mean square difference between left and right hemispheres, expressed as a fraction of the mean SNR. Reflects gross inhomogeneity. Lower is better.

Examples

```
%SETUP
mydir{1} = '/Users/tor/Documents/Tor_Documents/Coursework_and_Teaching/psyc7215/Data/UM_Face_Hou
wcard = 'rarun*img';
epi_names = filenames(fullfile(mydir{1}, wcard), 'absolute');

maskdir = '/Users/tor/Documents/Tor_Documents/Coursework_and_Teaching/psyc7215/Data/UM_Face_Hous
mask = 'implicit_mask.img';
mask_name = fullfile(maskdir, maskname);

%RUN
```

```
    QC = canlab_qc_metrics1(epi_names, mask_name);

    QC = canlab_qc_metrics1(epi_names, mask_name, 'noplot', 'printfile', 'test_qc.txt');
    QC = canlab_qc_metrics1(epi_names, mask_name, 'noplot', 'printfile', 'test_qc.txt', 'noheader');
```

diagnostics.**check_cluster_data**(*cl*)

loads the first 5 images from the first voxel

Usage

```
        check_cluster_data(cl)
```

cl(1).imnames(1:5,:)

diagnostics.**compare_subjects**(*varargin*)

This function compares a set of images to one another and does some diagnostics on the similarity among images. - It returns multivariate distances and dissimilarities among images - It works on the GLOBAL signal after standardizing each image (case 1) or the REGIONAL values in each cluster (case 2) - You can also enter a reference image, in which case each image will be correlated with the ref.

Usage

```
    function [ds, g, mystd, d, d2, c, c2, mi, b, eigv, eigval] = compare_subjects([img files or clus
                                    [plot flag], [title on figure], [standardize flag], [text lab
```

Inputs a list of image names to compare

OR

a clusters structure, with data to compare in timeseries field

If a mask is entered, only voxels in the mask (e.g., with value of 1) will be used. You can use this option to specify brain-only or gray-matter only voxels

textlab: optional text labels for each image, can be empty []

If a ref image is entered, each image will be correlated with the ref, and values will be saved for the correlation (plot 2 will show these values) Useful for comparing anatomical imgs with template, etc.

Outputs from correls with ref image are in variable "c"

ds: multivariate distance (sim. to Mahalanobis) for each image ds is a matrix of squared distances, case numbers, and expected chi2 values (in columns in this order) rows are cases

g: global value for each image

d: global distance from mean image distance, or dissimilarity, is the average absolute deviation between images

d2: matrix of distances among all images

c: correlation between real valued voxels and mean image

c2: correlations among all images (treating voxels as cases)

mi: mutual information between images, with hist2.m

b: principal component scores on correlation matrix for eigenvalues > 1

eigv: eigenvectors

eigval: eigenvalues

Examples

```
% Compare normalized anatomcals with standard brain
P = get_filename2(['sub*\Anatomy\nscalped_ft1.img']);
[ds, g, mystd, d, d2, c, c2, mi] = compare_subjects(P, which('brain_avg152T1.img'), 1, 'intext_c
```

diagnostics.**compare_subjects256**(*varargin*)

This function compares a set of images to one another and does some diagnostics on the similarity among images. - It returns multivariate distances and dissimilarities among images - It works on the GLOBAL signal after standardizing each image (case 1) or the REGIONAL values in each cluster (case 2) - You can also enter a reference image, in which case each image will be correlated with the ref.

Usage

```
function [ds,g,mystd,d,d2,c,c2,mi,b,eigv,eigval] = compare_subjects256([img files or clusters],[
                                [plot flag],[title on figure],[standardize flag],[text labels],[r
```

Inputs a list of image names to compare

OR

a clusters structure, with data to compare in timeseries field

If a mask is entered, only voxels in the mask (e.g., with value of 1) will be used. You can use this option to specify brain-only or gray-matter only voxels

textlab: optional text labels for each image, can be empty []

If a ref image is entered, each image will be correlated with the ref, and values will be saved for the correlation (plot 2 will show these values) Useful for comparing anatomical imgs with template, etc.

Outputs from correls with ref image are in variable "c"

ds: multivariate distance (sim. to Mahalanobis) for each image ds is a matrix of squared distances, case numbers, and expected chi2 values (in columns in this order) rows are cases

g: global value for each image

d: global distance from mean image distance, or dissimilarity, is the average absolute deviation between images

d2: matrix of distances among all images

c: correlation between real valued voxels and mean image

c2: correlations among all images (treating voxels as cases)

mi: mutual information between images, with hist2.m

b: principal component scores on correlation matrix for eigenvalues > 1

eigv: eigenvectors

eigval: eigenvalues

Examples

```
% Compare normalized anatomcals with standard brain
P = get_filename2(['sub*\Anatomy\nscalped_ft1.img']);
[ds,g,mystd,d,d2,c,c2,mi] = compare_subjects256(P,which('brain_avg152T1.img'),1,'intext_countloc
```

diagnostics.**displayme**(*mm*, *txtlab*, *tlab2*)

Used in img_hist2 - included as internal function there. this function is for indepenent re-display after img_hist2 is finished.

Usage

```
function [subjM,Mtotalv] = displayme(mm,txtlab,tlab2)
```

Example

```
% TO run:
[O.subjM,O.Mtotalv] = displayme(O.m,txtlab,'MEANS');
[O.subjS,O.Stotalv] = displayme(O.s,txtlab,'STD');
[O.subjW,O.Wtotalv] = displayme(O.w,txtlab,'SKEWNESS');
[O.subjK,O.Ktotalv] = displayme(O.k,txtlab,'KURTOSIS');
```

diagnostics.**ellipse**(*x*, *v1*, *v2*, *c*, *varargin*)

Gives x and y coordinates for an ellipse, given x coordinates, at a distance of c

Usage

```
[x,y] = ellipse(x,v1,v2,c,[sorting method])
```

Inputs Based on the formula for an ellipse, x^2/v1^2 + y^2/v2^2 = c

c: is the distance from the origin

v1: is the x half-length

v2: is the y half-length

x: is a vector of points covering the x coordinates in the ellipse

Sorting methods

- sort by x, produces elliptical line in plot
- sort by y, produces horizontal lines in plot

Examples

```
[x,y]=ellipse((randn(1000,1)),1.5,2.5,1); figure; hh = plot(x(2:end-1),y(2:end-1),'r-')
rotate(hh,[0 90],45)  % rotate around z-axis by 45 degrees
x2 = get(hh,'XData'); y2 = get(hh,'YData'); hold on; plot(x2,y2,'bx');
rotate(hh,[0 90],-45)  % rotate original ellipse back

% fill
fill(x,y,'r','FaceAlpha',.2)
```

diagnostics.**fft_calc**(*dat*, *TR*)

Simple function to calculate the FFT power of a data vector (dat) as a function of frequency, given a sample-to-sample repetition time (TR)

Usage

```
[myfft, freq] = fft_calc(dat, TR)
```

diagnostics.**fmri_mask_thresh_canlab**(*fmri_file*, *outputname*, *implicit_masking_method*, *plot-figs*)

Implicit determination of which voxels are in-brain, based on the intensities of functional images. Assumes much (most) of the image has near-zero background noise values, and the in-brain values are substantially higher.

Usage

```
[mask_thresh, cl, inmaskvox, in_mask_logical_vector, maskfilename] = fmri_mask_thresh_canlab(fmr
```

Inputs

fmri_file: is either a list of file names or an fmri_data object

File names: a (preferably) 4-D file of imaging data, Analyze .img or .nii

fmri_data object: With multiple images loaded with *no* mask

outputname: is a mask file output name, e.g., 'mask.img', with .img extension. Empty [] means do not write output image.

Implicit_masking_method

mean:

take the top 95% of voxels above the mean value. used by

default if no value is entered

dip:

smooth the histogram and take the top 95% of values above the

first positive gradient

****plotfigs** [1/0]: enable or suppress mask display and orthviews

Outputs

mask_thresh: signal-value above which voxels are considered in brain

c1: clusters, from iimg_indx2clusters

inmaskvox: number of inmask voxels

dat: binary matrix of voxels that are in (1) or out (0) of mask

Note: we want to be more inclusive than not at this stage.

last edited Oct 2011 - add support for fmri_data/image_vector objects added figure suppression, SG 12/14/15 defaults

diagnostics.**get_filename**(*dwcard*, *wcard*, *varargin*)

Usage

```
function P = get_filename(dir_wcard,file_wcard,[verbose])
```

Start in directory above individual subject directories

Inputs

dwcard: Enter dwcard for wildcard of directories to look in; e.g. '02*'

wcard: Enter wcard for image or file to get - e.g., 'beta_0001.img' This can also include subdirectories (no *'s) 'anatomy/beta*img'

Output Returns list of all files in individual directories in string matrix

Missing files, or entries in directory that do not contain files, are removed from the list.

NOT entering a * in dwcard seems to produce an error.

Examples

```
    P = get_filename('02*','beta_0001*')
    P = get_filename('02*','beta_0001.img')
    P = get_filename('02*','anatomy/nnhe*_seg1.img')
    P = get_filename('020515sp*','anatomy/nnhe*_seg1.img')
```

one * is allowed in first string, multiple *'s in second, as long as they are in the filename, not directory names!

diagnostics.**get_filename2** (*dwcard*, *varargin*)

Usage

```
    function [P,P2,d] = get_filename2(search string (as with ls command),[verbose])
```

Start in directory above individual subject directories

Inputs

dwcard: Enter dwcard for wildcard of directories to look in; e.g. '02*'

wcard: Enter wcard for image or file to get - e.g., 'beta_0001.img' This can also include subdirectories (no *'s) 'anatomy/beta*img'

Outputs

Returns list of all files in individual directories in string matrix

P: file names with full paths

P2: file names only

d: list of directories searched for files

Missing files, or entries in directory that do not contain files, are removed from the list. NOT entering a * in dwcard seems to produce an error.

Examples

```
    P = get_filename2('02*/beta_0001*')
```

one * is allowed in the directory structure right now, multiple *'s in the filename.

diagnostics.**hist2** (*A*, *B*, *res*, *varargin*)
2-D histogram with res bins

Usage

```
    [H,mi,H2] = hist2(A,B,res,[plot])
```

A and B can be 3D, as in image volumes mi is mutual information, a la spm_mireg.m

diagnostics.**image_intensity_histograms** (*fout*, *imgs*, *varargin*)

Makes a sheet (fout.png) of intensity distribution histograms of imgs. Will put an even number of rows of subplots on each page saved. If more than one page is needed, will title outputs fout_1.png, etc.

Usage

```
    image_intensity_histograms(fout, imgs, [options])
```

Inputs

fout: imfilename to be saved ('.png' will be appended)

imgs: ima cell array of filenames of images to make histograms of

Optional Inputs

obj: im

'titles', cellarray: use strings in cellarray as plot titles (DEFAULT: use file names from imgs)

'bins', n: use n bins in histograms (DEFAULT: 100)

'ymax', y: use y-axis from 0 to y (DEFAULT: 10,000)

'xmax', x: use x-axis from -x to x (DEFAULT: 10)

'cols', c: use c columns of subplots (DEFAULT: 5)

'maxrows', r: use no more than r columns of subplots per page (DEFAULT: 7)

'includezeros': include zero intensities in histograms (DEFAULT: exclude zeros)

Tor Wager .. add path if necessary

diagnostics.**img_hist**(*imgname*, *subdir*)

A general function for plotting histograms of any image For each subject, comparing across subjects

Inputs

imgname: name of image file to make intensity histograms from

subdir: cell array of text strings containing names of individual subject directories (wherein are contained the file specified in imgname or each subject)

Performs the histogram plot twice, once for CSF space and once for gray matter

Locations of gray and CSF masks for each subject must be defined in the defaults section of the script. (hard-coded)

Start in directory above individual subject results

Examples

```
img_hist('beta_0010.img',subdir)
img_hist('con_0002.img',{'020827mk' '020829jh' '020903lb'}

% for batch
d = dir('020726ag/beta*img'); d = str2mat(d.name);
for i = 1:10:size(d,1)
    img_hist(deblank(d(i,:)),EXPT.subjects)
end

for i = 2:19,
   if i < 10, myz = '000';, else, myz = '00';, end,
   img_hist(['con_' myz num2str(i) '.img'],EXPT.subjects);,
end
```

Tor Wager .. ..

defaults

diagnostics.**img_hist2**(*subdir*)

A general function for plotting histograms of any image For each subject, comparing across subjects

Inputs

imgname: name of image file to make intensity histograms from

subdir: cell array of text strings containing names of individual subject directories (wherein are contained the file specified in imgname or each subject)

Performs the histogram plot a number of times, without plotting and reports the variance in pdf moments as a function of subject, run, and condition (beta img within run).

Start in directory above individual subject results

Examples

```
img_hist2(EXPT.subjects)
img_hist2({'020827mk' '020829jh' '020903lb'})
```

Tor Wager .. ..

defaults

diagnostics.**joint_hist**(*x*, *y*, *varargin*)

Create 2-D joint histogram from vectors x and y

Usage

```
[z, xbins, ybins] = joint_hist(x,y,[nbins],[noplot])
```

Inputs

x and y:** are vectors of paired observations on two variables

Outputs

z: is the matrix representing the joint histogram cols of z are bins of x, rows are bins of y in plot, X axis is y, Y axis is x

Optional: number of bins, suppress plotting

Examples

```
z = joint_hist(nnmfscores{i}{j}(:, 1),nnmfscores{i}{j}(:, 2), 50, 'noplot');
h = plot_joint_hist_contour(z, [0 0 1]);
```

diagnostics.**make_conv_mtx**(*sz*, *sampres*)

Constructs the matrix (H) for a linear convolution With the canonical SPM hrf such that Hx = conv(x,hrf)

Usage

```
function H = make_conv_mtx(sz,sampres)
```

Inputs

sz: size of output matrix (elements)

sampres: spm_hrf sampling resolution (~ TR), OR

if a vector, a custom HRF sampled at the appropriate frequency.

Tor Wager ..

diagnostics.**multivar_dist**(*X*)

multivariate normality checking and diagnostic plots

Usage

```
[ds, S, p] = multivar_dist(X)
```

Input given matrix X with cases = rows, cols = variables

Outputs

ds: is matrix of squared distances, case numbers, and expected chi2 values (in columns in this order) rows are cases

NOTE: Sorted in order of ascending distance!

S: estimated covariance matrix

mv_distance: squared distances in original order of rows

p: p-values in original order of rows

center

diagnostics.**orthogonalize**(*mX*, *X*, *varargin*)

orthogonalizes X with respect to mX, optionally scaling predictors of X For each nuisance covariate (column of X)

Usage

```
function X = orthogonalize(mX,X,[scale])
```

Regresses out model fits and saves residuals in X

diagnostics.**power_from_variance**(*con*, *N*, *sig2b*, *sig2wi*, *pthresh*)

Power and effect size measures, given contrast, N, and variance component estimates

Inputs

con: contrast/effect magnitude estimate; "mean difference"

N: sample size

sig2b: between-subjects variance estimate

sig2wi:

within-subjects variance estimate *note* this is not the "raw" within-subjects variance; it is the contribution to the group (2nd-level) variance, which is sig2within / number of images within-subjects

pthresh:

alpha (Type I error) rate; p-value threshold for power calculation

con, sig2b, and sig2wi can all be vectors, so you can run this function voxel-wise for a whole map at once

Outputs

power: Power from 0 to 1

t: effect size : expected t-value

d: effect size : Cohen's d

see effect_size_map.m for a whole-brain, image-based power mapping function

t-value threshold for significance at alpha level pthresh

diagnostics.**power_loss**(*y*, *ons*, *X*)

'true' model fit assume 'true' is FIR estimate

diagnostics.**publish_scn_session_spike_id**(*inputimgs*, *SUBJDATA*)

This function is a wrapper function to call scn_session_spike_id in 'multi-session' mode, using input data across the runs for a single subject. It runs the program, and generates both a yaml-format text file for uploading into the CANlab database, and an html file with all the results and images for that subject embedded.

Inputs

inputimgs: is a cell array of images (4-D) for each run in a separate cell.

SUBJDATA: Input fields of SUBJDATA define the experiment name, subject name, and directories for saving both QC images + yaml and HTML

Examples

```
SUBJDATA.study = 'NSF';
SUBJDATA.subject = subjects{i};
SUBJDATA.html_save_dir = fullfile(output_basedir, 'html_output');
SUBJDATA.subject_dir = fullfile(output_basedir, 'SubjectData', 'denoised_canlab', SUBJDATA.subje
```

Initialize yaml file for database integration

diagnostics.**qchist**(*images*, *Nbins*, *sparse*, *XLim*, *titles*)

This function generates a histogram of activations from a set of statistic images. Generally, you want the images to have a normal distribution. Highly skewed distributions may be indicative of bad data.

Usage

```
function: h = qchist(dat,Nbins,sparse,XLim)

  This function may generate multiple figuresa with 30 histograms
  each
```

Inputs

images: List of image file names OR fmri_data object.

Optional Inputs

Nbins: Number of bins in each histogram (default = 100)

sparse: flag for generating ONLY histograms (default = 0)

XLim: Xlim (default = [-1 1])

titles: a cell array of subplot titles. If omitted, titles are inferred from assuming images come from a directory structure that looks like the following: /.../subjname/contrastimage.nii

diagnostics.**reset_SPMcfg**()

resets columns in SPMcfg by removing all non-intercept nuisance covariates. runs on the SPMcfg.mat file in the current directory

diagnostics.**scale_imgs_by_csf**(*hP*)

Takes a string matrix of image file names finds the mean and std of the CSF space specified in a mask (hard-coded) and standardizes images by these values

Usage

```
Pout = scale_imgs_by_csf(hP)
```

Writes SC* images (SCaled)

assumes images are spatially normalized. uses a canonical CSF mask!

diagnostics.**scn_component_rsquare**(*V*, *nuisanceX*, *designX*)

Print a table of r-square values (variance explained) for each of V data vectors by nuisance (mvmt, physio) and task-related predictors

Designed to work with components

Examples

```
% Typical operation
scn_component_rsquare(compscore, movement_params(1:157, :), X(1:157, :));

% No design
scn_component_rsquare(compscore, movement_params(1:157, :));

% Neither design nor nuisance, uses linear drift
scn_component_rsquare(compscore, []);
```

diagnostics.**scn_session_spike_id**(*imgs*, *varargin*)

Gets global image values for a session, and uses trimts.m to find outliers. The optional input MADs allows one to lower or raise the threshold for identifying scans as spikes (default = 10).

Usage

```
[g, spikes, gtrim, nuisance_covs, spikesperimg, snr] = scn_session_spike_id(imgs,'mask',[mask na
```

Multi-session mode returns much more output and more images, and takes in a cell array with images (preferably 4-D) for each session (run).

Inputs

'mask',[pathtomaskfile]: mask images using the mask in pathtomaskfile, default: implicit mask

'MADs',[scalar]: change Mahalanobis distance, default: 10

'doplot',[0 / 1]: plot result figures, default: true

Returns:

g: global values

spikes: identified spikes

gtrim: trimmed/adjusted global values, can be used as covariate in GLM

nuisance_covs: a matrix of 1)gtrim and 2) dummy regressors that can be used to minimize spike influence in GLM

We may want to save norms on the number of outliers found.

Examples

```
% Get image names
for i = 1:6, sess_images{i} = filenames(sprintf('run%02d/vol0*img', i), 'char', 'absolute'); end

% Run
[g, spikes, gtrim, nuisance_covs, snr] = scn_session_spike_id(sess_images);
```

diagnostics.**scn_spm_choose_hpfilter**(*spm_results_dir*, *varargin*)

Plots and choice of optimal high-pass filter from an SPM first-level model directory (with statistics and contrasts estimated.)

Usage

```
scn_spm_choose_hpfilter(spm_results_dir, ['events_only'])
```

SPM5 compatible and SPM8.

Called by: scn_spm_design_check.m For all regressors or events only: see scn_spm_choose_hpfilter.m

diagnostics.**scn_spm_design_check**(*spm_results_dir*, *varargin*)

Run in a single-subject (first-level) SPM directory to check design matrix variance inflation factors and high-pass filtering. Prints out table of regressors and their above-threshold VIFs (see options). Saves .png images of the key figures.

Usage

```
scn_spm_design_check(spm_results_dir, varargin)
```

Optional Inputs

'events_only': Show plots and diagnostics for ONLY events, not nuisance covariates or other user-specified regressors. Useful when you have many nuisance covs.

'vif_thresh', t': Only regressors with a VIF > t will be printed in VIF table.

'sort_by_vif'': Sort regressors in VIF table by VIF (DEFAULT: order regressors as in model).

Calls: scn_spm_choose_hpfilter.m, scn_spm_get_events_of_interest.m

Examples

```
scn_spm_design_check(pwd, 'events_only');
```

diagnostics.**scn_spm_get_events_of_interest**(*SPM*, *varargin*)

Gets events of interest.

Usage

```
wh_cols = scn_spm_get_events_of_interest(SPM, varargin)
```

All regressors, or events only if 'events_only' is input as keyword 'from_multireg': followed by an integer, to include first n columns from the multireg R matrix as "of interest". only works with 'events_only' flag, of course.

diagnostics.**scnlab_norm_check**(*template*, *wanat_files*, *mean_func_files*, *subjects*)

Compares the similarity of one or two sets of images (wanat_files, mean_func_files) to a template image and to one another (via Malanobis distance) to determine whether some images are potential outliers. This is used to check the quality of spatial warping/normalization for a group of subjects, though it could be used for other purposes as well.

Usage

```
NORM_CHECK = scnlab_norm_check(template, wanat_files, mean_func_files, subjs)
```

Inputs

template: Char array with name of image of normalization template

wanat_files: Warped (to template) anatomical file names

mean_func_files: Names of mean functional images These images should all be in the same space/in register.

Subjs: Optional cell array of names for each subject, for display purposes

Outputs A structure with metrics (NORM_CHECK)

NORM_CHECK.global_t1: global values of first image series (wanat_files)

NORM_CHECK.std_t1: spatial standard deviation of first image series (wanat_files)

NORM_CHECK.names_t1: Names for columns of NORM_CHECK.norm_vs_template

NORM_CHECK.subjects: Cell array of names for each subject

NORM_CHECK.norm_vs_template:

Similarity data for subjects (rows) x metrics (cols) {'Dist. from group, actual chi2', 'Mutual info with template', 'Correlation with template'};

NB: Leave mean_func_files empty (e.g., []) to only check structural images

Computes metrics on the goodness of normalization based on multivariate distance, mutual information, and correlation with template. Automatically saves a .mat file of the results into the current directory.

the template file (i.e., avg152T1.nii) must be in the CURRENT working directory and have read/write permissions

USES the subfunction compare_subjects, which may be useful as a stand-alone function.

USED in canlab_preproc_norm_check.m

diagnostics.**scnlab_norm_check3**(*wt1*, *subjlabels*, *template*, *mask*, *varargin*)

WARNING: scnlab_norm_check3 is deprecated! All improvements are being placed in scnlab_norm_check.

Usage

```
EXPT = scnlab_norm_check3(wt1,subjlabels,template,mask,[print out MI])
```

Inputs

wt1: char array of wT1.img files, one per line

subjlabels: cell array of subject labels

template: template img that everything has been normalized to (usually the avg152T1.img file)

mask: image to mask with

Optional Input print out mutual information table - flag for whether or not to print out the MI table; defaults to 0

Examples

```
cd(studyroot); % wherever your study root is
wt1s = filenames('hr*/structural/wT1.img', 'char', 'absolute'); % assuming that hr is your study
subjlabels = filenames('hr*');
template = which('avg152T1.nii');
mask = filenames('scalped_avg152T1_graymatter.img', 'char', 'absolute'); % set to wherever your

EXPT = scnlab_norm_check3(wt1, subjlabels, template, mask);
```

THIS FUNCTION IS DEPRECATED; SCNLAB_NORM_CHECK IS PREFERRED

diagnostics.**scnlab_pca_check1**(*imgs*, *realign_files*, *X*, *spersess*)

Usage

```
function scnlab_pca_check1(imgs, realign_files or params (t x 6) across all runs, X, spersess)
```

Inputs

imgs: list of all image names in order

realign_files: movement param file for each session, names in a cell array, OR

a t x 6 matrix of realignment parameters across all sessions

X: design matrix; no intercept is needed

Examples

```
% setup code for auditory oddball data
cd('/Users/tor/Documents/Tor_Documents/Coursework_and_Teaching/Mind_Res_Net_fMRI_Course_2008/dat

imgs = filenames('*/sw*img','absolute','char')
realign_files = filenames('*/rp*txt')

% LOAD TASK ONSETS and CREATE DESIGN MATRIX
onsets{1} = load('novel_stimuli_run1.asc');
onsets{2} = load('target_stimuli_run1.asc');
onsets{3} = load('standard_stimuli_run1.asc');
onsets{4} = load('novel_stimuli_run2.asc');
onsets{5} = load('target_stimuli_run2.asc');
onsets{6} = load('standard_stimuli_run2.asc');

regs_per_sess = 3;
nsess = 2;
for i = 1:length(onsets), onsets{i} = onsets{i}'; end
X = cell(1, nsess);
X{1} = onsets2delta(onsets(1:3), 1, 249);
X{1} = X{1}(:, 1:end-1);
X{2} = onsets2delta(onsets(4:6), 1, 249);
X{2} = X{2}(:, 1:end-1);
X = blkdiag(X{:});
```

diagnostics.**spm_general_hist**(*hP*, *mP*, *textlab*, *varargin*)

Usage

```
function M = spm_general_hist(hP,mP,textlab,[suppress plot - enter anything])
```

Inputs

hP: list of file names to compute histograms from

mP: list of file names to compute masks from

textlab: text string, e.g. 'ventricles' to label output tiffs

Output histograms for all input images (usually contrast images from individual subjects) plotted against a normal curve.

The expected output is that each image will have roughly mean 0, with bumps or tails in the distribution of there are real activations in some parts of the brain.

Looking at these histograms may be helpful for detecting outliers or subjects with strange contrast values. These may be caused by bad scaling, multicolinearity in the design matrix, acquisition artifacts, task-correlated head movement, or ???

Histograms (blue) are overlaid on a Gaussian distribution (red) with a mean of 0 and a standard deviation equal to that of the observed data.

diagnostics.**spm_rfx_hist**(*cwd*)

Usage

```
function spm_rfx_hist(cwd)
```

Input a directory name where an spm or SnPM random effects analysis lives

Outputs histograms for all input images (usually contrast images from individual subjects) plotted against a normal curve.

The expected output is that each image will have roughly mean 0, with bumps or tails in the distribution of there are real activations in some parts of the brain.

Looking at these histograms may be helpful for detecting outliers or subjects with strange contrast values. These may be caused by bad scaling, multicolinearity in the design matrix, acquisition artifacts, task-correlated head movement, or ???

Histograms (blue) are overlaid on a Gaussian distribution (red) with a mean of 0 and a standard deviation equal to that of the observed data.

diagnostics.**struct2yaml**(*yamlfilename*, *DB*, *yamlfilemethod*, *dbmethod*)

Usage

```
struct2yaml(yamlfilename, DB, yamlfilemethod, dbmethod)
```

Inputs

yamlfilemethod: 'new' or 'add' (append)

dbmethod: how the canlab database will handle the record. 'add', 'replace', or 'keep_existing'

translate structure into YAML format text file this will be interpretable by the canlab database

Examples

```
yamlfilename = 'YAML_tmp.yaml';

DB.study = 'NSF';     % string; study code letters
DB.subject = '001';   % string; subject ID number
DB.occasion = '21';   % string; occasion ID; unique to subj*session
DB.unique_id = [DB.study '_' DB.subject '_' DB.occasion];
DB.mean_spikes_per_image = mean(cat(2, spikesperimg{:}));

struct2yaml(yamlfilename, DB, 'add', 'replace');
```

diagnostics.**tor_get_physio**(*varargin*)

Usage

```
[X,mP,spmP] = tor_get_physio([mP],[spmP],[nvoxels],[doortho])
```

arguments are optional, but you must enter them in this order.

Tor Wager 10/21/02

Get nuisance covariates likely to be related to physiological noise and head motion The algorithm:

The program extracts raw/preprocessed image data from the ventricles (CSF space), as defined by a mask denoting which voxels are CSF for that subject. Either all voxels or a randomly selected subset [nvoxels] is subjected to principal components analysis, to determine regular patters of drift over time and across voxels. Those patterns are expected to be related to global signal drift, head movement, and physiological noise, and are assumed to be UNrelated to the task of interest, by virtue of the fact that they occur in the ventricles.

PCA is done twice on the timeseries' of CSF voxels. The first time, PCA is done on the sums of squared values (not the correlations) of voxel timeserieses across the entire experiment, mean-centered based on the whole experiment. Most of the coherent variation in this case is expected to be due to head movement and changes in shims/gradients/etc. from run to run. The SS values are used because we want to weight the voxels with the highest variation most heavily, as they are presumably picking up most of this signal. The first 3 eigenvariates (canonical timeseries) are saved.

Following, a separate, second PCA is done on the correlation matrix of data within each session. Session data for each voxel are mean-centered and scaled relative to the session (variance of each voxel = 1). We do this because physiological noise-related signals may produce periodic signals of different magnitudes in different voxels, and we want to extract the most coherent signals we can within each session. So these eigenvariates are expected to reflect primarily noise related to physiology (heart rate, respiration). Up to 5 eigenvariates for each session are saved (nothing with eigenvalue < 1 is saved).

Next, the CSF-related nuisance covariates (eigenvariates from PCA) are combined with existing nuisance covariates and intercept columns from the existing design matrix (SPMcfg xX). The proportion of variance in each predictor of interest explained by this nuisance basis set is calculated using regression, and the nuisance covariates are orthogonalized with respect to each predictor of interest. There are good and bad results of this step. The bad is that any signal that tracks the predictors is attributed to the task, not to noise, even if it's actually caused by physiological artifact. So the orthogonalized basis set does not protect you from physiology or movement-related false positives. However, the nuisance covariates are also unlikely to reduce power in estimating you effects of interest. More importantly, it avoids false positives created when one predictor (A) is more highly correlated with the nuisance covariates than another (B). In practice, betas for A will tend to be smaller than B, given the same actual response to both, and a random effects analysis on A-B will produce false positive activations. Orthogonalization of the nuisance set precludes this.

Inputs

mP: CSF mask image file. *_seg3.img output from SPM is appropriate should be in same space and have same dims as functionals but automatic reslicing is done if necessary.

spmP: name (full path name preferred) of SPMcfg.mat file to use This contains the design matrix and raw/preproc image file names to use.

nvoxels: Number of CSF voxels to use in PCA analysis More than 100 can be very slow and memory intensive. Fewer than 100 voxels loads a different way, and may be slower. Best is probably between 100 - 1000. 800 runs pretty fast.

doortho: Orthogonalize nuisance covariates with respect to regs of interest This assumes that any signal that covaries with the task is, in fact, due to the task, so it gives you some bias towards finding positive results. However, the alternative is that nuisance covariates may soak up variance related to the task, and you'll miss activations. In addition, if some regressors are more colinear with the nuisance set, you can create false "activations" when comparing these regressors to other ones. This problem exists whether or not we choose to model nuisance covariates. One solution is to use the ortho when doing random effects analyses, as the sign and magnitude of nuisance-related activations would not be expected to be the same across subjects unless the variance was really task-related. Default is 1, or "yes, do orthogonalization."

for functions called, see this .m file.

Examples

```
% get filenames for SPMcfg files and CSF mask for each subject
cd C:\Tor_Documents\CurrentExperiments\intext2\RESULTS\model1
spmP = get_filename('sub*','SPMcfg.mat');
cd C:\Tor_Documents\CurrentExperiments\intext2\
mP = get_filename('sub*','anatomy/nscalped_f*seg3.img');
% Now run:
for i = 1:size(mP,1)
    tor_get_physio(mP(i,:),spmP(i,:),300);  % 300 voxels
    pause(10); close all
end
```

Functions called

- spm functions: spm_get, etc.
- timeseries2.m (for < 100 voxels)
- read_hdr.m (big-little endian dependent; validate for your data)
- timeseries3.m (for > 100 voxels; uses SPM's image reading)
- reslice_imgs.m
- mask2voxel.m (only if ind2sub.m from Matlab is not found)

2.3 GLM_Batch_tools

GLM_Batch_tools.**canlab_glm_getinfo**(*modeldir*, *varargin*)

SUBJECT LEVEL input

```
INFO = canlab_glm_getinfo(spm_subject_level_directory, option, [n])
```

Get information out of an spm subject level analysis.

Options (each option can be called by the listed letter or word + a number, when noted)

i' 'input': number of volumes and (first) volume name for each run

'b' 'betas' [n]: beta names (for nth session)

'B' 'taskbetas' [n]: beta names (that didn't come from multiple regressors) (for nth session)

'c' 'cons' [n]: contrast names (for nth contrast)

'C' 'conw' [n]: beta names and weights for contrasts (for nth con)

'v' 'image' [n]:

create figure of design matrix (for nth session) (design matrix is multiplied by 100 for visibility) (works well for multiple runs)

'V' 'taskimage' [n]: same as 'image', but only for task betas

'imagesc':

same as 'image', but uses imagesc (works well for single runs)

GROUP LEVEL input

```
INFO = canlab_glm_getinfo(robfit_group_level_directory, option, [n])
```

Get information out of a robfit group level analysis.

Options (each option can be called by the listed word or letter + a number, when noted)

Any of the subject level options can be used on a group level robfit analysis by prefixing '1i' (output is generated based on the first input to the first robust analysis).

Ex:

```
canlab_glm_getinfo('second_level/model3','1iconw')
```

'i' 'input' [n]: input contrasts by number and name (for nth analysis)

'I' 'allinput' [n]: input images (for nth analysis)

'm' 'model': weights by subject (i.e., directory containing input contrast images)

'M' 'allmodels' [n]: weights and input images (for nth analysis)

Assumptions In some options, the first contrasts and group level analysis directories are assumed to represent the rest, which may not be the case.

Note group level options do not yet return a usable INFO struct.

GLM_Batch_tools.**canlab_glm_group_levels**(*varargin*)

Performs group level robust GLM analysis with robfit

1. sets up analysis
2. runs robfit
3. (optionally) make inverse p maps (for FSL viewing)
4. (optionally) estimate significant cluster sizes
5. publishes analysis with robfit_results_batch

Usage

```
canlab_glm_group_levels([options])
```

Optional Inputs

's', subjects: cell array of filenames of subject-level analysis directories IN modeldir (note: modeldir won't be prepended to absolute paths)

'm', modeldir: filename of directory containing subject level analyses

'o', grpmodeldir: output directory name

'c', cov: a matrix describing group level model (do not include intercept, it is automatically included as first regressor)

see help robfit

note: requires specifying an output directory name

note: ordering of inputs (rows) must match subjects ordering

'n', covname: a cell array of names for the covariates in cov

'f', covfile: a csv file will specify the group level model: first column, header 'subject', contains names of subject directories (to be found in modeldir) subsequent columns have covariates, headers are names of covariates name of covfile will be name of group analysis directory (placed in grpmodeldir)

'mask', maskimage: filename of mask image

DSGN: will use the following fields of the DSGN structure: modeldir = DSGN.modeldir

subjects = DSGN.subjects

maskimage = DSGN.mask

Note A covfile will cause other specifications of subject, cov, and covnames to be ignored.

If parameters are defined more than once (e.g., modeldir or subjects), only the last entered option will count.

Defaults

subjects: all SPM.mat-containing directories in modeldir

modeldir: pwd

grpmodeldir: modeldir/one_sample_t_test

cov: {} (run 1 sample t-test, see help robfit)

covname: 'groupmean'

mask: 'brainmask.nii'

Options

'README': prints canlab_glm_README, an overview of canlab_glm_{subject,group}_levels

'overwrite': overwrite existing output directories

'noresults': don't run/publish robfit_results_batch

'onlyresults': just run/publish robfit_results_batch, don't run robfit (assumes existing analyses)

****'whichcons', [which cons]** vector of contrasts to analyze (DEFAULT: aall subject level contrasts) see [which cons] in help robfit

'invp' [, target_space_image]: generate inverse p maps and resample to the voxel and image dimensions of target_space_image (viewable on dream with ~ruzicl/scripts/invpview)

'nolinks': do not make directory of named links to robust directories (using contrast names)

'dream':

if you're running on the dream cluster, this option will cause

all analyses (e.g., lower level contrasts) to be run in parallel (submitted with matlab DCS and the Sun Grid Engine) Note: currently only works with MATLAB R2009a

'email', address: send notification email to address when done running

GLM_Batch_tools.**canlab_glm_group_levels_run1input**(*wd*, *c*)

child process of canlab_glm_group_levels (see canlab_glm_README.txt for an overview)

GLM_Batch_tools.**canlab_glm_maskstats**(*DIRS*, *MASK*, *varargin*)

Returns a MASKSTATS structure containing data from robfitdir's input subject level SPM analyses.

Usage

```
MASKSTATS = canlab_glm_maskstats(robfitdir, mask, [options])
```

output structure: MASKSTATS

COV: subject x covariate matrix (EXPT.cov from robfit design)

COVNAME: names of covariates in COV

MASK: array of structs (1 per mask)

MASKFILE: the filename of the mask used

SUB: struct containing data from subject level images

CON: struct contains data from contrast images

NAME: cell array (1 cell per contrast) of contrast names

IMGFILES: cell array (1 cell per contrast) of character arrays of contrast image filenames

(MEASURE): subject X contrast matrix of measures (see canlab_maskstats)

COVxCON.(MEASURE): arrays of covariate matrix X contrast means correlation results (RHO and P, see help corr())

GRP: struct containing data from group level images

BETA: struct array (1 struct per group level regressor) of data from beta images

NAME: name of group level regressor

IMGFILES: cell array (1 cell per robust directory) of beta image files

(MEASURE): row vector (1 value per robust directory) of measures (see canlab_maskstats)

The following plots are saved in each mask's directory in the plots directory:

- contrast means by contrast (means are lines across subjects on x axis)
- contrast means by subject (means are dots, lined up along the x axis by contrast)
- group level betas (bar plot with group level regressors grouped by subject level contrasts)

If there's more than one regressor in the group level model, for each regressor:

- scatter plot of subject level contrast means against group level regressor

Arguments

robfitdir: a directory in which robfit was run contains robfit directories (e.g., robust0001) preferably contains EXPT.mat or EXPTm.mat

mask: a filename or cell array of filenames of masks to apply to data

Options

MEASURE OPTIONS: (see canlab_maskstats) (DEFAULT: mean (within mask's non-zero voxels))

'cons', connums: (vector of contrast numbers) only include data from contrasts as specified by numbers

'cons', conname(s): (string or cell array of strings) only include data from contrasts as specified by name

'plots': make plots

'od', dir: will save plots in dir (DEFAULT: robfitdir/stats_scatterplots)

GLM_Batch_tools.**canlab_glm_publish**(*varargin*)

Usage

```
canlab_glm_publish(directory_specifications [options])
```

Directory Specification

's', dirs: Generates HTML reports of the output from scn_spm_design_check for directories in cell array dires (string allowable for single dir). If a directory is a subject level analysis, the HTML will be generated for that subject in the analysis directory. Ex:

canlab_glm_publish('s',{'model1/1011' 'model1/1014'})

If a directory contains subject level analyses, an HTML will be generated with output for each subject level analysis. Ex:

canlab_glm_publish('s','model1')

ASSUMPTION: lower level analyses contain an SPM.mat file.

'g', dirs: For each "robfit directory" in cell array dirs, will run robust_results_batch on all contrast directories (e.g., robust0001/) (string allowable for single dir). ("robfit directories" contain robfit contrast directories (like robust0001)) EITHER directories contain EXPT.mat files (see help robfit) OR an EXPT struct is loaded in the workspace and a single directory is specified OR will do best to figure out info normally contained in EXPT.mat Ex:

canlab_glm_publish('g', {'group_n35' 'group_anxiety_n35' 'group_sadness_n35'})

Note directory paths may be absolute or relative (to working directory)

Options

't', {[pthresh clustersize] ...}: Use the paired voxelwise_pthresh and minimum_cluster_size thresholds with which to produce robfit results maps. This option must follow immediately after a 'g' option (see above) and will only apply to the analyses specified in that option. ONLY applies to robfit directories (no bearing on lower level design checks)

DEFAULT: {[.001 5] [.005 1] [.05 1]} Ex:

canlab_glm_publish('g', pwd, 't', {[.001 1] [.005 10] [.05 10] [.01 25]})

'email', address: send notification email to address when done running Ex:

canlab_glm_publish('g', pwd, 'email', 'ruzic@colorado.edu')

GLM_Batch_tools.**canlab_glm_roistats**(*DIRS*, *ROI*, *varargin*)

Use canlab_glm_maskstats instead

GLM_Batch_tools.**canlab_glm_subject_levels**(*dsgnarg*, *varargin*)

Performs lower level GLM analysis with SPM:

1. specifies model
2. estimates model
3. generates contrast images for model
4. creates directory with named links to spmT and con maps
5. publishes analyses with scn_spm_design_check

Usage

```
canlab_glm_subject_levels(DSGN [options])
```

DSGN struct - defines the model and analysis parameters

canlab_glm_subject_levels('README') to see description

Options

'README': prints canlab_glm_README, an overview of canlab_glm_{subject,group}_levels

'dsgninfo': prints description of DSGN structure

'subjects', subject_list: ignore DSGN.subjects, use cell array subject_list

'overwrite': turn on overwriting of existing analyses (DEFAULT: skip existing)

'onlycons': only run contrast job (no model specification or estimation) note: will overwrite existing contrasts note: to not run contrasts, simply do not include a contrasts field in DSGN

'addcons': only run contrasts that aren't already in SPM.mat option to canlab_spm_contrast_job

'nodelete': do not delete existing contrasts (consider using addcons, above) option to canlab_spm_contrast_job

'nolinks': will not make directory with named links to contrast images

'noreview': will not run scn_spm_design_check

'dream': if you're running on the dream cluster, this option will cause all subjects to be run in parallel (submitted with matlab DCS and the Sun Grid Engine) Note: currently only works with MATLAB R2009a

'email', address: send notification email to address when done running

Model specification and estimation done by canlab_spm_fmri_model_job

Contrasts are specified by canlab_spm_contrast_job_luka see that function for more info.

GLM_Batch_tools.**canlab_glm_subject_levels_run1subject**(*wd*, *s*)
child process of canlab_glm_subject_levels (see canlab_glm_README.txt for an overview)

2.4 HRF_Est_Toolbox2

HRF_Est_Toolbox2.**Anneal_Logit**(*theta0*, *t*, *tc*, *Run*)
Estimate inverse logit (IL) HRF model using Simulated Annealing Creates fitted curve - 3 logistic functions to be summed together - from parameter estimates

Usage

```
[theta,HH,C,P] = Anneal_Logit(theta0,t,tc,Run)
```

Inputs

Run: stick function

tc: time course

t: vector of time points

theta0: initial value for the parameter vector

HRF_Est_Toolbox2.**Det_Logit**(*V0*, *t*, *tc*, *Run*)

Estimate inverse logit (IL) HRF model Creates fitted curve - 3 logistic functions to be summed together - from parameter estimates

Usage

```
[VM, h, fit, e, param] = Det_Logit_allstim(V0,t,tc,Run)
```

Inputs

Run: stick function

tc: time course

t: vector of time points

V0: initial value for the parameter vector

HRF_Est_Toolbox2.**Fit_Canonical_HRF**(*tc*, *TR*, *Run*, *T*, *p*)

Fits GLM using canonical hrf (with option of using time and dispersion derivatives)';

Usage

```
function [hrf, fit, e, param, info] = Fit_Canonical_HRF(tc,TR,Runs,T,p)
```

Inputs

tc: time course

TR: time resolution

Runs: expermental design

T: length of estimated HRF

p: Model type

Options

- p=1 - only canonical HRF
- p=2 - canonical + temporal derivative
- p=3 - canonical + time and dispersion derivative

Outputs

hrf: estimated hemodynamic response function

fit: estimated time course

e: residual time course

param: estimated amplitude, height and width

info: struct containing design matrices, beta values etc

HRF_Est_Toolbox2.**Fit_Logit2**(*tc*, *TR*, *Run*, *T*, *mode*)

Fits FIR and smooth FIR model

Usage

```
function [hrf, fit, e, param] = Fit_Logit(tc,Run,t,mode)
```

Inputs

tc: time course

TR: time resolution

Runs: expermental design

T: length of estimated HRF

mode:

deterministic or stochastic

Options: 0 - deterministic aproach

1 - simulated annealing approach

Please note that when using simulated annealing approach you may need to perform some tuning before use.

Outputs

hrf: estimated hemodynamic response function

fit: estimated time course

e: residual time course

param: estimated amplitude, height and width

HRF_Est_Toolbox2.**Fit_sFIR**(*tc*, *TR*, *Run*, *T*, *mode*)

Fits FIR and smooth FIR model

Usage

```
function [hrf, fit, e, param] = Fit_sFIR(tc,TR,Runs,T,mode)
```

Inputs

tc: time course

TR: time resolution

Runs: expermental design

T: length of estimated HRF

mode: FIR or smooth FIR

Options: 0 - standard FIR

1 - smooth FIR

Outputs

hrf: estimated hemodynamic response function

fit: estimated time course

e: residual time course

param: estimated amplitude, height and width

HRF_Est_Toolbox2.**Get_Logit**(*V*, *t*)

Calculate inverse logit (IL) HRF model Creates fitted curve - 3 logistic functions to be summed together - from parameter estimates

Usage

```
[h] = get_logit(V,t)
```

Inputs

t: vector of time points

V: parameters

HRF_Est_Toolbox2.**HMHRFest**(*y*, *Runs*, *TR*, *nbasis*, *norder*)

HRF estimation algorithm

Inputs

y: Data matrix (#time points) by (#subjects) by (#voxels)

Runs: Stick functions for each subject (#time points) by (#conditions) by (#subjects)

TR: Time resolution

nbasis: Number of b-spline basis

norder: Order of b-spline basis

HRF_Est_Toolbox2.**PowerLoss**(*modres*, *modfit*, *moddf*, *tc*, *TR*, *Run*, *alpha*)

Estimates Power-loss due to mis-modeling.

Usage

```
function [PowLoss] = PowerLoss(modres, modfit, moddf, tc, TR, Run, alpha)
```

Inputs

modres: residuals

modfit: model fit

moddf: model degrees of freedom

tc: time course

TR: time resolution

Runs: expermental design

alpha: alpha value

Output

PowLoss: Estimated power loss

HRF_Est_Toolbox2.**ResidScan**(*res*, *FWHM*)

Calculates P(M>=t) where M is the max value of the smoothed residuals. In this implementation the residuals are smoothed using a Gaussian kernel.

Usage

```
function [p sres sres_ns] = ResidScan(res, FWHM)
```

Inputs

res: residual time course

FWHM: Full Width Half Maximum (in time units)

Outputs

p: pvalues

sres: smoothed residuals

sres_ns: smoothed residuals (non standardized)

HRF_Est_Toolbox2.**get_parameters2**(*hdrf*, *t*)

Find model parameters

Height - h

Time to peak - p (in time units of TR seconds)

Width (at half peak) - w

Calculate Heights and Time to peak:

delta = 1/(t(2)-t(1));

HRF_Est_Toolbox2.**hrf_fit_one_voxel**(*tc*, *TR*, *Runc*, *T*, *method*, *mode*)

HRF estimation function for a single voxel;

Usage

```
function [h, fit, e, param] =  hrf_fit_one_voxel(tc,TR,Runc,T,type,mode)
```

Implemented methods include: IL-model (Deterministic/Stochastic), FIR (Regular/Smooth), and HRF (Canonical/+ temporal/+ temporal & dispersion)

Inputs

tc: time course

TR: time resolution

Runs: expermental design

T: length of estimated HRF

type: Model type

mode: Mode

Options

- p=1 - only canonical HRF
- p=2 - canonical + temporal derivative
- p=3 - canonical + time and dispersion derivative

Outputs

hrf: estimated hemodynamic response function

fit: estimated time course

e: residual time course

param: estimated amplitude, height and width

HRF_Est_Toolbox2.**ilogit**(*t*)

Calculate the inverse logit function corresponding to the value t

Usage

```
function [L] = ilogit(t)
```

Output

L: exp(t)./(1+exp(t));

2.5 OptimizeDesign11

OptimizeDesign11.**optimizeGA**(*GA*)

Usage

```
M = optimizeGA(GA)
```

outputs a pseudo-random list of condition codes that optimizes multiple fitness measures for fMRI task designs

more help can be found in ga_example_script.m and in Genetic_Algorithm_readme.rtf

OptimizeDesign11.**optimizeGA_epochs**(*GA*)
outputs a random-ordered list of condition #s that optimizes 3 fMRI considerations

Ways to avoid block designs counterbalancing factor power lower limit cutoff pushes power higher

Why using avg power is better than efficiency efficiency doesn't account for 1/f model (altho here we just use a cutoff, the same thing) avoid transformation errors in using high pass filter efficiency is based on the sample size, determined by TR, of the model - but so is fft power...

OptimizeDesign11.**optimize_rand_search**(*GA*)
Just like the GA, but generates random designs each time!

outputs a random-ordered list of condition #s that optimizes 3 fMRI considerations%

Ways to avoid block designs counterbalancing factor power lower limit cutoff pushes power higher

Why using avg power is better than efficiency efficiency doesn't account for 1/f model (altho here we just use a cutoff, the same thing) avoid transformation errors in using high pass filter efficiency is based on the sample size, determined by TR, of the model - but so is fft power...

CHAPTER 3

Miscellaneous Tools

3.1 Data_extraction

Data_extraction.**canlab_maskstats**(*msks*, *imgs*, *varargin*)

Produces comparison of pattern mask and images e.g., look at NPS pattern expression in set of beta images

Usage

```
MASKSTATS = canlab_maskstats(maskfiles,imgfiles,[options])
```

Inputs

maskfiles: string or cellstring of mask filenames or fmri_data object

imgfiles: string or cellstring of image filenames or fmri_data object

Optional Inputs

ts: timeseries treatment: each string in imgfiles is assumed to be a 4D file. Data will be returned with one column per time series and one volume per row. If not all timeseries are same length, all will be NaN-padded to the length of the longest timeseries.

Note: does not work with imgfiles input as fmri_data object

dir2cell: will sort stats into cells based on directory containing the imgfile they belong to such that each cell contains one directory's worth of stats which is a vector with a value for each imgfile. :Examples:

```
# Input: a list of single trial betas for a set of subjects
b = filenames('sub*/*heat_trials*.img');
ms = canlab_maskstats('nps',b,'dot_product','dir2cell');
```

Output: includes the set of cells that are input to a mediation analysis.

keepzeros: don't remove zeros from imgfiles before taking measurements

keepzerosmask: don't remove zeros from maskfiles before taking measurements

single: leave data as single (DEFAULT: convert to double)

trinarize: trinarize maskfile (set values larger than 0 to 1 and values less than zero to -1)

noreshape: don't attempt to reshape results according to imgfiles array

nobin: don't binarize mask before extracting mean or std

Note ts, dir2cell, and noreshape are mutually exclusive options

Built-In Masks The following strings can be given as the maskfile argument to call up built-in mask files:

nps: weights_NSF_grouppred_cvpcr.img

nps_thresh: weights_NSF_grouppred_cvpcr_FDR05.img

nps_thresh_smooth: weights_NSF_grouppred_cvpcr_FDR05_smoothed_fwhm05.img

Measure Options

all:

add: mean, dot_product, centered_dot_product, cosine_similarity, and correlation

mean: (DEFAULT) apply binarized mask to images and return means mean(img .* abs(bin(mask)))

std: apply binarized mask to images and return standard deviations std(img .* abs(bin(mask)))

dot_product: dot(mask, img)

cosine_similarity: dot(mask, img) / (norm(mask) * norm(img))

correlation: corr(mask, img)

centered_dot_product: dot(mask-mean(mask), img-mean(img))

Details

- imgfiles are spatially resampled to maskfiles
- voxels with zeros in maskfile are removed
- in-mask voxels with zeros in imgfiles will generate warnings

`Data_extraction.`**`cluster_tmask`**(*cl*, *tm*, *si*, *varargin*)

Given clusters and a string name of a t-image, finds voxels that exceed a specified t-threshold

Usage

```
[cl, varargout] = cluster_tmask(cl, tm, si, varargin)
```

Inputs

cl: clusters

tm: t-image

si: subject index integer

[dat]: cluster_barplot data structure

creates new XYZ in the space of t-image using cl.XYZmm coordinates in mm. Required fields of cl: XYZmm

Calculates and saves single-subject data avgd over voxels if:

1. **cl.all_data field is present** THIS WORKS if all_data has individual subject contrast estimates in it, with rows as subjects and columns as voxels indiv data saved in cl(region).timeseries(subject)
2. **cl.raw_data is present** raw_data should be time x voxels x subjects, a 3D matrix see output of extract_raw_data. indiv data saved in cl(region).indiv_timeseries(:, subject)

Note Retains upper 50% of voxels; highest t-values

Data_extraction.**extract_contrast_data**(*P*, *clusters*, *varargin*)

This function does not plot, but separates subclusters using pca / cluster_princomp.m based on pattern across all conditions, covariance (not correlation),

Usage

```
function [clusters, subcl] = extract_contrast_data(P, clusters)
```

Inputs

P: cell array of strings containing image file names (data extracted from these)

clusters:

Optional Inputs

subclusters: to get sub-clustering based on pca and clustering of voxels add the string 'subclusters' as an input argument

cell array: of strings with condition names

split: value is 1: to split into 2 plots (first half and last half of P)

value is 2: to plot individual subjects over bars

center: center parmeter values in plot (subtract row means) this gives closer to correct "within subjects" error bars and may be used when the overall parameter values have no meaning

covs: followed by between-subject covariates (e.g., behavioral regressors) plots remove these before plotting means and std. errors

max: to make plots based on max z-values within region for each dataset P not compatible with 'split' and 'center' (ignores these commands) right now, special for inhib - see also inhib2_cluster_barplot (good function)

indiv: to threshold based on individual t-statistics or contrast values

FOLLOW with cell array of t-images or con images – usually, there will be one cell, with images for each subject in rows, to define voxels for each ss. BUT Tnames can be the same length as contrast images, one t-img per subject per contrast, if desired.

Outputs clusters struture, with CONTRAST substructure added substructure contains data extracted and image file names

This program uses XYZmm millimeter coordinates in clusters to find voxels So clusters and data files may have different dimensions.

Examples

```
cl = extract_contrast_data(EXPT.SNPM.P, cl, 'indiv', EXPT.FILES.Timgs{1});
cluster_barplot(EXPT.SNPM.P(7:12), clusters(2:3), {'ObjE' 'AttE' 'InteractE' 'ObjI' 'AttI' 'Inte
[clusters, subclusters] = cluster_barplot(EXPT.SNPM.P(17:24), clusters, 'subclusters', 'split')
RS2_8vs2_placeboCP = cluster_barplot(EXPT.SNPM.P([8 10 12 14 16]), RS2meta, 'indiv', T);
```

also see mask2clusters.m, a simpler version that just extracts clusters from a mask file.

Data_extraction.**extract_from_rois**(*imgs_to_extract_from*, *mask_image*, *varargin*)

Generic function for extracting image data from a mask or atlas image, and returning the data and averages within regions specified by the user.

Usage

```
    [cl, imgdat] = extract_from_rois(imgs_to_extract_from, mask_image, varargin)

Regions to average over can be either regions of contiguous voxels
bounded by voxels with values of 0 or NaN, which are considered non-data
values, or regions defined by unique integer codes in the mask image
(i.e., for atlas images with unique codes for each defined region.)

Mask/Atlas image does NOT have to be in the same space as the images to
extract from.  It will be remapped/resliced.

extracted data is returned in single data format.
```

Inputs **char array** of strings containing 4D image file names (data extracted from these)

mask_image to extract from

Optional Inputs

average_over: Default = 'unique_mask_values' to average over unique integer codes in the mask image (i.e., for atlas images with unique codes for each defined region)

OPT = 'contiguous_regions' to average over contiguous voxels bounded by voxels of 0 or NaN (non-data values)

Examples

```
imgs_to_extract_from = filenames('w*.nii','char');
mask_image = which('anat_lbpa_thal.img');
[cl, imgdat] = extract_from_rois(imgs_to_extract_from, mask_image);
```

Data_extraction.**extract_image_data**(*imgs_to_extract_from*, *mask_image*, *varargin*)

Generic function for extracting image data from a mask or atlas image, and returning the data and averages within regions specified by the user.

Usage

```
[imgdat, volInfo, cl] = extract_image_data(imgs_to_extract_from, mask_image, varargin)
```

Regions to average over can be either regions of contiguous voxels bounded by voxels with values of 0 or NaN, which are considered non-data values, or regions defined by unique integer codes in the mask image (i.e., for atlas images with unique codes for each defined region.)

Mask/Atlas image does NOT have to be in the same space as the images to extract from. It will be remapped/resliced.

extracted data is returned in single data format.

Inputs **char array** of strings containing 4D image file names (data extracted from these)

mask_image to extract from

Optional inputs

average_over: Default = 'contiguous_regions' to average over contiguous voxels bounded by voxels of 0 or NaN (non-data values)

Alt. option = 'unique_mask_values' to average over unique integer codes in the mask image (i.e., for atlas images with unique codes for each defined region)

Examples

```
imgs_to_extract_from = filenames('w*.nii','char');
mask_image = which('anat_lbpa_thal.img');
[imgdat, volInfo, cl] = extract_image_data(imgs_to_extract_from, mask_image, 'unique_mask_values
```

Related functions

For an object-oriented alternative, see the fmri_data class and extract_roi_averages method

Data_extraction.**extract_indiv_peak_data**(*cl*, *imgs*)

Purpose: to find individually significant regions within each subject and save average timecourses for each individual ROI for each subject

Usage

```
cl = extract_indiv_peak_data(cl,imgs)
```

Inputs

cl: is a clusters structure with one element per region. Each element (cluster) is a structure containing coordinates and data.

imgs: is a string matrix with one row per subject, with names of images used to define thresholds. These may be contrast or t-images from individual subjects

The method of extraction is defined in cluster_tmask, which is currently to use 50% of voxels with the highest values in imgs(subject) for each subject, or 100% if 50% returns less than 5 voxels. It's easy in cluster_tmask to use absolute t-thresholds instead.

Required fields of cl

XYZmm: 3 x k list of k coordinates in mm space

raw_data: time x voxels x subjects matrix of data for this region

Outputs appended to cl structure: indiv_timeseries, time x subjects averaged individual ROI timecourses

:.INDIV:, a structure with the following information:

- tname, the name of the t-mask (or contrast mask) entered
- XYZ, voxel coords for significant voxels for each subject
- XYZmm, mm coords for sig voxels for each subject
- sigt, logical matrix subjects x voxels for sig voxels
- maxt, max map value for each subject
- center, average coordinate for sig voxels for each subject
- mm_center, the same in mm

The spatial information may be used to correlate peak voxel location with behavior, for example.

Data_extraction.**read_hdr**(*name*, *varargin*)

Loads the analyze format header file from a file 'name'

Usage

```
function hdr = read_hdr(name,[opt] datatype)
```

The function returns a structure defined as

```
hdr = struct(... 'sizeof_hdr', fread(pFile, 1,'int32'),...
        'pad1', setstr(fread(pFile, 28, 'char')),...
        'extents', fread(pFile, 1,'int32'),...
        'pad2', setstr(fread(pFile, 2, 'char')),...
        'regular',setstr(fread(pFile, 1,'char')), ...
        'pad3', setstr(fread(pFile,1, 'char')),...
        'dims', fread(pFile, 1,'int16'),...
        'xdim', fread(pFile, 1,'int16'),...
        'ydim', fread(pFile, 1,'int16'),...
        'zdim', fread(pFile, 1,'int16'),...
        'tdim', fread(pFile, 1,'int16'),...
        'pad4', setstr(fread(pFile,20, 'char')),...
        'datatype', fread(pFile, 1,'int16'),...
        'bits', fread(pFile, 1,'int16'),...
        'pad5', setstr(fread(pFile, 6, 'char')),...
        'xsize', fread(pFile, 1,'float'),...
        'ysize', fread(pFile, 1,'float'),...
        'zsize', fread(pFile, 1,'float'),...
        'pad6', setstr(fread(pFile, 48, 'char'))...
        'glmax', fread(pFile, 1,'int32'),...
        'glmin', fread(pFile, 1,'int32'),... 'descrip', setstr(fread(pFile, 80,'char')),...
        'aux_file' , setstr(fread(pFile,24,'char'))',...
        'orient' , fread(pFile,1,'char'),...
        'origin' , fread(pFile,5,'int16'),...
        'generated' , setstr(fread(pFile,10,'char'))',...
        'scannum' , setstr(fread(pFile,10,'char'))',...
        'patient_id' , setstr(fread(pFile,10,'char'))',...
        'exp_date' , setstr(fread(pFile,10,'char'))',...
        'exp_time' , setstr(fread(pFile,10,'char'))',...
        'hist_un0' , setstr(fread(pFile,3,'char'))',...
        'views' , fread(pFile,1,'int32'),...
        'vols_added' , fread(pFile,1,'int32'),...
        'start_field' , fread(pFile,1,'int32'),...
        'field_skip' , fread(pFile,1,'int32'),...
        'omax' , fread(pFile,1,'int32'),...
        'omin' , fread(pFile,1,'int32'),...
```

'smax' , fread(pFile,1,'int32'),...

'smin' , fread(pFile,1,'int32'));

Data_extraction.**readim2**(*varargin*)

Usage

```
[array,hdr,h,whichslices,rows,cols,figh] = readim2(basename or array [opt],'p' [opt], 'sagg' or
```

Inputs

- basename of file, without image extension ,OR
- 3-D array in the workspace to plot, OR
- nothing, to browse for file

p: to plot montage of slices to the screen

sagg: to rotate to saggital view

cor: to rotate to coronal view

t: to save array as double instead of int16 - to save negative t values.

Outputs

- 3d array of image
- hdr of image
- handles for axes of montage if plotting

Special Features

range:

specified in mm, must be LAST and FIFTH input argument.

- OR range can specify slices, e.g. 1:4:28

clim: color limits for axis plot, must be 1st or 2nd argument. form: [-1 1]

Data_extraction.**timeseries_extract_slice**(*V*, *sliceno*, *orientation*)

For a given set of image names or memory mapped volumes (V) extracts data from slice # sliceno and returns an X x Y x time matrix of data.

Usage

```
function sl = timeseries_extract_slice(V,sliceno)
```

Data_extraction.**tor_extract_rois**(*imnames*, *varargin*)

This function gets timeseries data from all clusters in an SPM results output.

Usage

```
function [clusters, SPM, xX, xCon] = tor_extract_rois(imnames [can be empty],[opt] SPM, [opt] VC
```

Inputs

imnames: a matrix of image names, in spm_list_files output format if empty, no timeseries data will be extracted.

clusters: if only 2 arguments, clusters structure is 2nd arg, and we extract data using existing clusters structure

If 3 arguments, enter SPM and VOL to extract data from VOXEL coordinates in these structures - SPM: SPM variable from loaded results - VOL: VOL variable from loaded results

Optional 4th argument fits a design matrix and returns betas - xX: xX design matrix to fit to timeseries OR 4th argument can be 0 (or any non-structure arg), suppressing verbose output.

[Last 2 arguments are optional. Use if results are already loaded into workspace]

Automatic fitting of model to cluster timeseries average using analyze_cluster_rois with High-Pass filter length of your choice. This only works if you input only the file names or input all optional arguments, including xX

NOTE (WARNING): WORKS ON XYZ VOXEL COORDINATES - TRANSFORMATION TO DIFFERENT SPACES ONLY IF ENTERING 2 ARGS, 1st one names, 2nd one clusters

see transform_coordinates.m for transformation, or check_spm_mat, or cluster_interp.

3.2 Data_processing_tools

Data_processing_tools.**center_of_mass**(*XYZ*, *Z*)

This function returns the center of mass of a cluster of voxels or mm coordinates defined as the nearest in-list coordinate to the average of the coordinate values weighted by the Z-score

Usage

```
com = center_of_mass(XYZ,Z)
```

assigns a rank to each coordinate based on Z scores and includes

enter a 3 x n list of XYZ coordinates returns 1 x 3 center of mass

Data_processing_tools.**detransition**(*y*, *varargin*)

For fMRI timeseries that contains large 'jump' artifacts due to motion correction or other problems.

Usage

```
y = detransition(y,[doplot])
```

Removes these large spikes. Updated version built into spikecorrect in trimts

Data_processing_tools.**downsample_scnlab**(*y*, *orig_samprate*, *new_samprate*, *varargin*)

Uses linear interpolation to resample a vector from one sampling rate to another

Usage

```
[yi, xi] = downsample_scnlab(y, orig_samprate, new_samprate, [doplot])
```

Inputs

orig_samprate: sampling rate in Hz

new_samprate: desired sampling rate in Hz

Example

```
    % Downsample a 100 Hz signal to a scanning TR of 2 sec
    % signal at 100 Hz, sample to low-freq TR of 0.5 hz (2 sec TR)
    % every 100 / TR = 100/.5 = 200 samples

    [yi, xi] = downsample_scnlab(y, 100, .5)
```

Data_processing_tools.**fft_plot_scnlab**(*dat*, *TR*, *varargin*)

Usage

```
    [myfft, freq, handle] = fft_plot_scnlab(dat, TR, varargin)
```

Inputs

dat: is a data vector (column)

TR: is the sampling rate of the data you put in in seconds / sample, or 1/Hz

Optional inputs

- 'samefig'
- 'color, ['b'] or other color
- 'bar'
- 'linebar': both line and bar

Examples

```
    % plot effects of filtering on a difference
    % between two regressors
    spm_hplength = SPM.xX.K.HParam;
    d = SPM.xX.X * SPM.xCon(mycon).c(:, 1);
    create_figure('Contrast'); plot(d) % contrast we care about
    px = pinv(SPM.xX.K.X0);           % pinv of the filtering matrix
    y = d;
    y = y - SPM.xX.K.X0 * px * y;     % residuals after filtering

    [myfft, freq, handle] = fft_plot_scnlab(d, 2);
    hold on;
    [myfft2, freq2, handle] = fft_plot_scnlab(y, 2); set(handle,'Color','r')
    plot_vertical_line(1/spm_hplength)
    set(ans, 'Color', 'b', 'LineWidth', 3)
```

Data_processing_tools.**filterAdjust**(*O*)

Usage

```
    [y,O,X,S] = filterAdjust(OPTIONS)
```

O

O.y: signal

O.HP: high pass freq. cutoff

O.TR: sampling rate in s

O.doHP: [0 or 1] - do HP filter (spm), default is 0

O.doLP:

[0, 1, 2] - do LP filter (spm), default is 0 2 = Gaussian filter with TR*2 s length

O.firstimg: sets values for first image in each run to mean of the remaining values. Good for removing first-image artifacts present in some scanner sequences. Default is 1.

O.cyclecorrection: checks for unimodal (normally distributed) data within each session, because some bimodal data that cycles between two mean scanner values has been observed in some data. if a high proportion of outliers are found in non-normal data, subtracts mean of higher mode to adjust data. Sorry-not clear. Check the code. Default is 0

O.cyclecorrection2: removes large transitions from data, as in detransition.m. Default is 0. Artifacts may be acquisition or motion-correction/resampling related.

We do this after filtering, but if there's trouble, we re-do the scanadjust and filtering, because 'cycling' can affect these.

O.scanadjust:

[0 or 1] - adjust to scan means, default is 0

- If O.X is entered, assumes this is the session mean matrix,

instead of recomputing.

O.percent: [0 or 1] - adjust to percent change from baseline, default is 0

O.filtertype:

filter style, default is 'none'

- 'spm', use spm's filtering
- if O.S is entered, uses this instead of recomputing
- 'fourier', Doug's fourier filter
- 'fouriernotch', Omit frequencies between HP(1) and HP(2)
- 'cheby', chebyshev
- 'Luis', Luis' custom filter
- 'none', no filtering (or leave field out).

O.HP: for SPM, the filter cutoff in s for fourier, the HP value or the [HP LP] values notches out everything slower than HP and faster than LP, in s (1/s = Hz).

O.nruns: number of runs (scanadjust), default is 1

O.adjustmatrix: custom adjustment matrix to regress out (e.g., movement params)

O.plot [0 or 1]: plots intermediate products, default is 0

O.verbose [0 or 1]: verbose output

O.trimts [0 or std]: trim overall timseries values to std, 0 for no trimming

O.lindetrend: specify linear detrending of timeseries. occurs after adjustment and filtering and windsorizing

- detrending option -> what to enter in this field:
- no detrending -> empty, missing field, or 0
- detrending every n elements -> single number (n)
- piecewise linear detrend -> ROW vector of breakpoints (do not specify 1 as the start of the 1st segment.)

Data_processing_tools.**fir2htw2**(*b*, *varargin*)

Estimates height, time to peak, and width of FIR response

Usage

```
[h,t,w,w_times,halfh, auc] = fir2htw2(b,[hconstraint],[doplot],[colors cell])
```

Inputs

b: beta/estimate series for hemodynamic response curve

hconstraint:** max time in samples that can be considered the peak (default = last sample)

doplot:** flag for plot, 1/0

colors:** cell vector of colors for plot

minh = min height

This version uses turning points (zero gradient) to find the largest "hump" in the data and the time it occurs.

Example

```
hrf = spm_hrf(.5); hrf = hrf ./ max(hrf); hrf = hrf + .1 * randn(length(hrf), 1);
create_figure('hrf'); plot(hrf);
[h,t,w,w_times,halfh, auc] = fir2htw2(hrf, [], 1);
```

Data_processing_tools.**get_snr**(*data*)

Data is a matrix whos columns index voxels, and rows index subjects (or trials, etc.)

Usage

```
snr = get_snr(data)
```

Data_processing_tools.**htw_from_fit**(*hrf*, *b*, *dt*, *varargin*)

Estimates height, time to peak, width, and area under the curve of a fitted response

Usage

```
h, t, w, auc, w_times, halfh] = htw_from_fit(hrf, b, dt, [optional arguments])
```

Inputs

hrf:

a matrix of columns that form a linear basis set for an event type in an fMRI design, t time points x k basis functions

b: betas associated with the columns of hrf, k x 1

dt: the sampling resolution of hrf (in seconds)

Optional Inputs

plot: make plot

verbose: verbose output

startval: followed by the starting value in sec within which to calculate peak

endval: followed by the ending value in sec within which to calculate peak

colors: followed by a cell array, for example, {'r'} or {[1 0 0]}

This function is essentially the same as fir2htw2.m, but the main differences are:

1. It imposes a time constraint on the peak amplitude automatically, which is constrained to be between 4 seconds and 12 seconds (endval, which was hconstraint) by default. YOU MAY WANT TO CHANGE hconstraint depending on whether you're expecting delayed hemodynamic responses. This requires input of the sampling resolution (e.g., dt)

2. This function will automatically create fitted responses, given a basis set and betas (parameters). This is different from fir2htw2.m, which takes the fitted response as input.

3. The method for getting width (w) has been changed to work better for multi-modal (multi-peak) responses.

Otherwise, the algorithm is the same. See Lindquist and Wager, 2007, for simulations that use a version of this method to estimate HRFs using different kinds of models.

Notes on scaling

The scaling of the amplitude depends on the scaling of the hrf basis set, which (in SPM) depends on the time resolution. You should at least use an hrf basis set with the same scaling for all subjects in a group analysis. The amplitude of the fitted response is interpreted as the amplitude of the unit "impulse response," assuming that the hrf you enter here is the same as the impulse response function you used to create the design matrix. In SPM5, higher-res impulse response functions are normalized by their positive sum, and the higher the time resolution, the lower the amplitude of the unit HRF. (The scaling of regressors in the SPM design matrix isn't affected, because the hrf basis functions are convolved with a boxcar that also depends on the time resolution. The bottom line is that if all your subjects have the same scaling, you should be fine. And, secondly, the amplitudes that come out of this function reflect the scaling of the HRF you put in and are for an impulse response, NOT for an "event," and so the scaling here would not be expected to match up with the amplitudes on a plot that you'd get from a selective average time-course plot, unless you adjust by multiplying by the number of elements in SPMs "hi-res" onset boxcar to adjust.

For example

With "zero-duration" events, an hrf input scaled to reflect "event response" amplitudes might look something like this: (******may not be exactly right because i think dt is in sec) figure; plot(conv(SPM.xBF.bf(:, 1), my_ons)) my_ons = ones(1, TR ./ SPM.xBF.dt . SPM.Sess.U(1).dur(1));

If you have epochs and want "epoch response" amplitude, you have to consider that as well. If your durations are specified in TRs, and all durations are the same: TR = SPM.xY.RT; my_ons = ones(1, TR ./ SPM.xBF.dt .* SPM.Sess.U(1).dur(1));

minh = min height

This version uses turning points (zero gradient) to find the largest "hump" in the data and the time it occurs.

Examples

```
% Load and SPM mat file and use the basis set stored in that, and use
% that as the hrf.  Generate some arbitrary combos to test different shapes:
% cd('my spm directory')
% load SPM
[h, t, w, auc] = htw_from_fit(SPM.xBF.bf, [1 .4 .4]', SPM.xBF.dt, 'plot', 'verbose');

for i = 1:20
    [h, t, w, auc] = htw_from_fit(SPM.xBF.bf, randn(3, 1), SPM.xBF.dt, 'plot'); pause(1.5);
end

% Generate an SPM basis set at a lower resolution, and try that:
bf = spm_get_bf(struct('name', 'hrf (with time and dispersion derivatives)', 'length', 30, 'dt',
for i = 1:20
   [h, t, w, auc] = htw_from_fit(bf.bf, randn(3, 1), bf.dt, 'plot'); h, t, w, auc, pause(1.5)
end
```

Data_processing_tools.**luisFilter**(*data*, *TR*, *cutoff*, *verbose*)

This is a low pass FIR filter using the Parks Mclellan design algorithm The phase introduced by the filter is linear and is un-done by filtering the data again, backwards if you want to see what it does to the data, use verbose=1 if just want to filter the data, don't use the argument at all.

Usage

```
function outdata = luisFilter(data,  TR, cutoff [, verbose])
```

close all

These are the important lines of the code %%%%%%%%%%%%

Data_processing_tools.**nuisance_cov_estimates**(*X*, *images*, *SETUP*, *varargin*)

Estimates F-map for model parameters of interest Writes to disk: 'F_cols_of_interest.img' 'p_cols_of_interest.img' (3-D images) 'resid_full_model.img' (a 4-D image)

Locates voxels whose activity is unrelated to the model

Extracts principal components from these voxels for use as covariates in subsequent models

Note: invalidates statistical inference in subsequent models based on these data, but may improve predictive accuracy and single-trial model.

Usage

```
nuisance_cov_estimates(X, images, SETUP, varargin)
```

Defining the SETUP structure with inputs

SETUP.(fields)

.wh_of_interest:

vector of which columns of X matrix are of interest % columns of interest in X matrix; tests var explained by these with F-test

.mask: name of mask image

.TR: repetition time of volume (image) acquisition

.HPlength: high-pass filter length, in s

.scans_per_session: vector of # volumes in each run, e.g., [128 128 128 128 128]

.dummyscans: indices of images in each run that will be modeled with separate dummy variables

.startslice: start at slice #...

SETUP Optional Inputs

nopreproc: to skip preprocessing (i.e., for trial-level inputs)

Data_processing_tools.**resample_scnlab**(*data*, *p*, *q*, *varargin*)

Resample : Uses matlab's resample.m, but pads ends to avoid edge artifacts

Usage

```
[y, x] = resample_scnlab(data, p, q)
% OR
[y, x] = resample_scnlab(data, p, q, origHz, targetHz)
```

Y = RESAMPLE(X,P,Q) resamples the sequence in vector X at P/Q times the original sample rate using a polyphase implementation. Y is P/Q times the length of X (or the ceiling of this if P/Q is not an integer). P and Q must be positive integers.

Other features:

Returns x values for resampled data in original index scale

IF two additional args are entered (origHz and targetHz), p and q are determined automatically, based on your desired sampling rate (targetHz)

Example

```
create_figure('test'); plot(y);
[y2, x] = resample_scnlab(y, 1, 5);
plot(x, y2, 'r');

%Use target Hz...take 100 Hz vector and resample at 20 Hz
create_figure('test');
plot(y); [y2, x] = resample_scnlab(y, [], [], 100, 20);
plot(x, y2, 'r');
```

Data_processing_tools.**scale**(*x*, *varargin*)

Centers and scales column vectors to mean 0 and st. deviation 1

Usage

```
x = scale(x,[just center])
```

Data_processing_tools.**scnlab_filter_fmri_data**(*imgs*, *mvmt*, *mask*, *tr*, *spersess*, *hp*)

Outlier and artifact removal for one subject Writes new output images for timeseries

Usage

```
names = scnlab_filter_fmri_data(imgs, mvmt, mask, tr, spersess, hp)
```

Examples

```
OUT = mean_image(imgs, 'mean_ravol.img',ones(size(imgs,1),1));
spm_imcalc_ui('mean_ravol.img', 'graymatter.img', 'i1 > 0');
spm_image('init', 'graymatter.img');
names = scnlab_filter_fmri_data(imgs, mvmt, 'graymatter.img', 2, repmat(184, 1, 6), 80);
```

Data_processing_tools.**scnlab_outlier_id**(*varargin*)

Methods (modes of operation)

Setup Run this method first to generate an options structure OPT that can be passed in along with any data vector for speedy processing

Usage

```
OPT = scnlab_outlier_id('setup', 'tr', 2, 'spersess',
                        [184 184 184 184 184 184], 'dummy', 1:3,
                        'hp', 100, 'mad', 4, 'niter', 3, 'mvmt', mvmt);
```

Data Run this method second with an already-created OPT and a data vector from one time series

```
[y2, outliers, num_outliers, mvmt_rsquare] = scnlab_outlier_id('data', y, 'options', OPT);
```

all outputs:

```
    [y2, out, nout, mvmtrsq, mvmt_baseline_rsquare, yperc, rawvarp, rawvarF, percvarp, percvarF, ...
    ybase] = scnlab_outlier_id('data', y, 'options', OPT);
```

Example

```
    [dat, volInfo] = iimg_get_data('graymask.img', imgs);
    y = dat(:,1);
    % SETUP:
    OPT = scnlab_outlier_id('setup', 'tr', 2, 'spersess', [184 184 184 184 184 184], 'dummy', 1:3, '
    % RUN:
    [y2, outliers, num_outliers] = scnlab_outlier_id('data', y, 'options', OPT);

    % run on whole brain
    [dat, volInfo] = iimg_get_data('graymask.img', imgs);
    OPT = scnlab_outlier_id('setup', 'tr', 2, 'spersess', [184 184 184 184 184 184], 'dummy', 1:2, '
    OPT.doplot = 0;
    OPT.verbose = 0;
    fhandle = @(y) scnlab_outlier_id('data', y, 'options', OPT);
    y2 = matrix_eval_function(dat, fhandle)';
```

Data_processing_tools.**selective_average**(*y*, *onsets*, *varargin*)

Purpose: Get a selective average of values of data vector y, given onsets specified in onsets. Onsets can be fractional; in this case, linear interpolation is used.

Usage

```
    [averages, stderrs, data, indices] = selective_average(y, onsets, varargin)
```

Inputs

y: is a data vector to get selective averages from. It should be a column vector.

onsets: should be a cell array, with one cell per condition each cell should contain a column vector of onset times in SAMPLES (same resolution as y; e.g., in TRs, if y is an fMRI time series.

Optional Inputs

t: followed by number of time points following onset to use; default is 20

plot: plot results.

baseline: followed by vector of which time points are baseline values; will subtract from each

Outputs

data: indices, averages, stderrs: Cell vectors, one cell per condition

data, indices: time points (observations) x trials (onsets)

indices: Cell vector, one cell per condition; time points (observations) x trials (onsets)

Examples

```
    onsets = {[1 10 30 80]'  [20 60 90]'}; y = (1:120)';
    [averages, stderrs, data, indices] = selective_average(y, onsets, 't', 20)

    V = spm_vol(EXPT.FILES.im_files{1});
    y = spm_get_data(V, [10 10 10 1]');
    [averages, stderrs, data, indices] = selective_average(y, onsets2(1), 't', 20, 'plot');
```

Data_processing_tools.**selective_average_group**(*V*, *onsets*, *xyz_mm_pos*, *varargin*)

uses selective_average.m used in selective_average_interactive_view_init.m

Examples

```
[group_avgs, group_stes, subject_avgs, subject_stes] = selective_average_group(V, onsets, vox, '

% Format onsets from onsets2 (NSF study format) into correct format for this function
N = length(imgs); n_conditions = size(eventdesign{1}, 2);
onsets = cell(1, N);
for i = 1:N
   for j = 1:n_conditions
       onsets{i}{j} = onsets2{i}(find(eventdesign{i}(:, j)));
   end
end
```

Data_processing_tools.**smooth_timeseries**(*x*, *perc*)

Make exponential smoothing function with perc proportion of data points (0 < perc < 1) OR specify length directly, as % of points to 0 weight

Usage

```
[x,V] = smooth_timeseries(x,perc)
```

apply smoothing filter V to data (x), V * x (works just as well for a matrix of column vectors) You could also apply it to a model matrix X

Data_processing_tools.**splineDetrend**(*v*, *varargin*)

What it does: A spline detrend with knot points every 2 s (hard coded number) I made this up too - it's not the FDA approved method.

Usage

```
[fv,bp,yy,myfft] = splineDetrend(v,'p' [opt])
```

Input

v: a vector to be detrended

Optional Inputs

p: means plot

any other opt argument sets the knots and is treated as an integer, with detrending every n elements.

Outputs

fv: the detrended vector, bp, the knot points, and myfft, the abs(fft) of the detrended vector.

Examples

```
tmp = cl(1).raw_data(:,1,6); tmp2 = trimts(tmp,3,[],1); [fv,bp,yy]=splineDetrend(tmp2);
[fv,bp] = splineDetrend(tmp2,'p');
```

Data_processing_tools.**splinetrim**(*y*, *varargin*)

Usage

```
function [y,ntrimmed,spikes, yfit] = splinetrim(y,[iqrmult],[knotrate],[X],['p'])
```

Uses a robust measure of deviations in a timeseries gradient to find high-velocity 'spikes', presumed to be artifacts

Uses spline interpolation to replace spikes with reasonable values.

Input

y: a timeseries

Optional Inputs

iqrmult:

how many times the interquartile range above which velocities are outliers, default is 1.5

knotrate: sets knot points every k observations, default is 3

X: matrix of session means or other linear regressors to remove

p: plot the results

X and 'p' can be entered in any order, but after iqrmult and knotrate

Examples

```
[y2,nt] = splinetrim(trialdat,3,5,'p'); nt
```

•setup

Data_processing_tools.**trimts**(*y*, *sd*, *X*, *varargin*)

1.Adjusts for scan effects (unless X is empty)

2.**Windsorizes timeseries to sd standard deviations**

- Recursive: 3 steps

3.Adds scan effects back into timeseries

Usage

```
function [y,ntrimmed,allw] = trimts(y,sd,X,[do spike correct],[trimming iterations],[MADs])
```

Spike correct: Some attempt at automatic adjustment for abrupt level shifts in data; default is 0, enter 1 to do this "Spikes" are IDd as values more than 10 MADs (by default) from moving average with 20 image FWHM Replaces "spike" data values with moving average

iterations: number of cycles through trimming; default is 3

MADs: allows you to change the number of MADs above which values are IDd as "spikes"

filter y using X matrix; yf is residuals

Data_processing_tools.**use_spm_filter**(*TR*, *dims*, *LChoice*, *HChoice*, *HParam*, *varargin*)

Usage

```
function [S,KL,KH] = use_spm_filter(TR,dim of filter,LChoice,HChoice,HP filter in s,[LP Gauss le
```

Inputs

K{s}.LChoice: Low-pass filtering {'hrf' 'Gaussian' 'none'}

K{s}.LParam: Gaussian parameter in seconds

K{s}.HChoice: High-pass filtering {'specify' 'none'}

3.3 Filename_tools

Filename_tools.**check_valid_imagename**(*P*, *varargin*)

Usage

```
P = check_valid_imagename(P, [return error])
```

Optional Input 0 to return empty output, 1 to break with an error message,

2 to return list of bad/missing images, or

nothing to select missing filenames graphically

Filename_tools.**copy_image_files**(*image_list*, *to_dir*, *varargin*)

Copies a set of image files from one directory to another, creating the directory if needed.

Usage

```
function output_names = copy_image_files(image_list, to_dir, [method='copy' or 'move'], varargin
```

MAC OSX only!!

For .img files, will look for a paired .hdr and copy it automatically. Works for non-image files (e.g., .txt) as well.

Other variable args:

Will take flags:

'append image number' -> will append a number to each file corresponding to the order listed, e.g., 1 for the first, 2 for the 2nd, etc.

'append string' -> followed by string to append

with both flags and 'run' for append string, a matrix of [run1/vols.img; run2/vols.img] would become [vols_run0001.img vols_run0002.img]

Filename_tools.**delete_ana_imgs**(*imgs*)

Function to delete a list of .img files

- automatically removes associated .hdr files as well.
- accepts cellstr or char inputs

Usage

```
delete_ana_imgs(imgs)
```

Filename_tools.**dicom_tarzip**(*varargin*)

tars all .dcm files in subdirectories you specify, and deletes the original .dcm files if successful.

Inputs none (default): all directories under the current one directory wildcard, e.g., 'run*' to search only those subdirectories. absolute path names should also be ok.

Examples

```
dicom_tarzip

dicom_tarzip('18*')
```

Or, let's say you have runs within subjects, and dicom files within run dirs. Subject dirs all start with NSF*. run dirs all start with 18* Starting from the directory above subjects, you could do this to zip ALL subjects:

```
    subjdirs = dir('NSF*');
    for s = 1:length(subjdirs)
        dicom_tarzip([subjdirs(s).name filesep '18*']);
    end
```

Needless to say, use this with extreme caution, as it deletes your original files. And test it thoroughly with a backup dataset first!!

Filename_tools.**escapeForShell**(*string*)

Returns converted string for use with the current OS's shell. Hence, it's operation will be different, depending on where it's executed.

Usage

```
    [escapedString] = escapeForShell(string)
```

Filename_tools.**expand_4d_filenames**(*imgs*, *nvols*)

Expand a list of image names to 4-D name format as used by SPM2/5 Also compatible with SPM8.

Usage

```
    [spm_imgs] = expand_4d_filenames(imgs, [nvols])
```

This function runs with either imgs, nvols, or both variables defined If the images exist, spm_imgs is returned as a char matrix with each row in the following format: 'path/filename.ext,[N]' where [N] is the number of the given image. The space after [N] is padded with blanks to ensure that all the image names fit properly.

Example

```
    spm_imgs = expand_4d_filenames('run03.img', 10)
```

Filename_tools.**filename_get_new_root_dir**(*P*, *newroot*, *nlevels*)

Usage

```
    Pout = filename_get_new_root_dir(P,newroot,nlevels)
```

Inputs

P: is files (string matrix)

newroot: Append a new root directory to filenames if newroot is empty, prompt

nlevels:

Keeps nlevels dirs deep from old directory:

- keeps only filename
- keeps filename plus last subdir
- keeps filename plus last 2 subdirs, etc.

Examples

```
    Pout = filename_get_new_root_dir(EXPT.SNPM.P{3},pwd,2)

    % Change root dir to pwd for all images in EXPT
    for i=1:length(EXPT.SNPM.P), EXPT.SNPM.P{i} = filename_get_new_root_dir(EXPT.SNPM.P{i},pwd,2); e
```

Note: see regexprep.m for a simpler way!!! This function could be improved by using it.

Filename_tools.**getfullpath**(*Pspm*)

Searches for file in curr dir, then in specified dir

Usage

```
Pso = getfullpath(Pspm)
```

Input

Pspm: is file name with no path or relative path

Output Returns absolute path name (full path)

Filename_tools.**nums_from_text**(*mytext*)

returns numeric values of numbers embedded in text strings given text string input

Usage

```
function [nums,whnums] = nums_from_text(mytext)
```

Filename_tools.**read_excel**(*excelfilename*, *has_header_row*)

Reads each sheet of an Excel input file into a data structure DAT

Usage

```
   DAT = read_excel(excelfilename, has_header_row [1 or 0])

- reads multiple sheets
- adds fields named with variable names for easy access
- uses importdata <matlab internal> to do most of the work
```

Input

excelfilename: Char array with full path of input filename

Outputs

DAT: Structure with DAT.(sheetname).(varname) data fields DAT.(sheetname).varnames is cell array of var names

behdat: Data structure as read in by importdata

Examples

```
excelfilename = fullfile(basedir, 'data', 'SchulzNewcorn_regressor_demograph.xlsx');
has_header_row = 1;
[DAT, behdat] = read_excel(excelfilename, has_header_row)
```

See also

importdata, read_database, read_database2, read_edat_output_2008, read_physio_data

Filename_tools.**remove_disdaq_vols**(*img_files*, *num_vols_per_run*, *num_disdaq_vols*, *varargin*)

Usage

```
new_files = remove_disdaq_vols(img_files, num_vols_per_run, num_disdaq_vols, ['overwrite', 0|1],
```

Inputs

img_files: cellstr of files to remove disdaqs from - each cell represents a run

num_vols_per_run: vector of volume counts per run, *not* including disdaq vols

num_disdaq_vols: constant describing how many data points to remove from the beginning of each run

'overwrite': if set, will overwriting images in place - defaults to 0

'FSLOUTPUTTYPE': outputfile type - defaults to 'NIFTI'

'strict': if set, will error out unless data given to it is exactly proper length - defaults to 1 - turn off ONLY with good reason

Output

new_files: input files, but with a 'd' prepended, unless 'overwriting' was specified

NB: Not set up for 3d files yet!!!

Examples

```
% for an experiment
num_vols_per_run = [124 140 109];  % NOT including disdaqs
num_disdaq_vols = 4;
```

Filename_tools.**rename_lowercase**()
renames all files in dir with lower case versions if all characters are capitalized.

Example

```
d = dir('0*'); for i=1:length(d),cd(d(i).name),rename_lowercase, cd .., end
```

Filename_tools.**rename_uppercase**()
renames all files in dir with upper case versions if all characters are capitalized.

Example

```
d = dir('0*'); for i=1:length(d),cd(d(i).name),rename_lowercase, cd .., end
```

Filename_tools.**scan_get_files**(*n*, *filt*, *mesg*, *wd*)
Drop-in replacement for spm_get. Main advantage is that it can use spm_select for SPM5+ to mimic older spm_get behavior.

Usage

```
files = scan_get_files(n, filt, mesg, wd)
```

Filename_tools.**sort_image_filenames**(*P*)

Usage

```
[P, indices] = sort_image_filenames(P)
```

Not all image name listing functions return imgs in the correct numbered order!

This function resorts a string matrix of image file names by image number, in ascending order At most, filename can have one number in it, or error is returned

P can be a string matrix of a cell of string matrices

3.4 hewma_utility

hewma_utility.**Gaussian_mix**(*x*, *niter*, *basepts*, *verbose*, *doplot*, *doplot2*)

Two-Gaussian mixture model

Usage

```
[ind,ind2,stats] = Gaussian_mix(x,niter,basepts,verbose,doplot, doplot2)
```

Inputs

x: data

iter: number of iterations

basepts: number of baseline pts at start of run. The modal class in the baseline period is defined as the 0-class

Outputs

ind: indicator function of class belonging

ind2: indicator function of class belonging where 3 consecutive points are needed to switch states

mu: mean vector

sigma: standard deviation

p: probability that latent class random variable (delta) is equal to [0,1]

Examples

```
[ind,ind2,stats] = Gaussian_mix(linear_detrending(dat),20);
% dat is n subjects by t time points
% plotting is on

% Simulated data
r = normrnd(1, 1, 200, 1); r(1:50) = r(1:50) + normrnd(3, 1, 50, 1);
[ind, ind2, stats] = Gaussian_mix(r, 50, [], 0, 1);
```

We recommend 50 iterations ..

Set up inputs

hewma_utility.**change_point**(*t*, *tt*, *Mode*, *tthresh*)

Usage

```
[cp,tm,b,t,wh,ooc_vector] = change_point(t,tt,Mode,tthresh)

[cp,baseline mean,max t value,time of max t] = change_point(Zm,sterr,tt,tthresh)
```

Inputs

tt: is number of baseline timepoints

t: is group t-value timeseries

Mode: 'thresh' or 'time': if 'thresh': tthresh = threshold for significant t-values.

change_point finds the first significant supra-threshold t-value and looks back in time to estimate the change point.

if Mode = 'time'

change_point finds the estimated change-point for process determined to be out of control at time tthresh

Outputs

wh: indices of out of control points

ooc_vector: indicator vector for ooc points

hewma_utility.**cluster_kmeans_parcel**(*x*, *CLU*, *doplot*, *varargin*)

K-means clustering of voxels

Usage

```
[cl2,nclasses,colors] = function cluster_kmeans_parcel(x,CLU,doplot,[overlay img])
```

Inputs

x: a voxels x data matrix

CLU: a structure containing a list of XYZ coordinates and z-scores for voxels (see clusters2CLU.m)

a plot flag

Outputs

cl2: a k-length cell array of clusters structures each cell contains a clusters structure for one data class

hewma_utility.**cnt_runs**(*tc*)

Counts number of contiguous 1s in a timeseries

Usage

```
function [cnt, tot, lenmat] = cnt_runs(tc)
```

hewma_utility.**ewma5**(*X*, *lambda*, *L*, *type*, *tt*, *doplot*)

EWMA - Exponentially Weighted Moving Average for fMRI slice

Usage

```
SMap, SMapTh, CPMap, CNTMap, TWidMap, W1Map, W2Map, W3Map, Tsq,stats] = ewma3(X, lambda , L, typ
```

Inputs

X: Data over one slice (N x N x T data matrix) or (N x T) data matrix

lambda: smoothing factor

L: width of control limits, in standard errors

type: Type of noise model - White Noise: 'WN', AR(1): 'AR', AR+WN: 'ARMA'

Outputs

SMap: Significance Map

SMap: Significance Map Thresholded at L standard errors

CPMap: Changepoint Map

CNTMap: Map that depicts the number of times the process is out-of-control

TWidMap: map of total number of points out-of-control (all runs)

W1Map: map of number of points out-of-control in first run

W2Map: map of number of points out-of-control in second run

W3Map: map of number of points out-of-control in third run

Tsq: Squared, normalized data set for multi-subject analysis

stats.model: noise type

stats.lam: value of lambda

stats.Z: EWMA statistic (smoothed timeseries)

stats.var: variance of EWMA statistic

stats.doasym: 1 for asymptotic variance 0 otherwise

stats.dim: data dimension

stats.Z3: EWMA statistic on the format NxVxT

stats.UCL: upper control limit (at L standard errors, depends on lambda)

stats.LCL: lower control limit

Examples

```
% to plot voxel i,j from a slice
i=8;
j=17;
ts=squeeze(fsl(i,j,:));
tor_fig;
plot(ts);
hold on;
plot(squeeze(stats.Z(i,j,:)),'r');
plot(squeeze(stats.UCL(i,j,:)),'r:');
plot(squeeze(stats.LCL(i,j,:)),'r:');
plot([chptmap(i,j)],stats.Z(i,j,chptmap(i,j)),'yo','MarkerFaceColor','r')
```

hewma_utility.**get_ax_slice**(*imgs*, *slice_num*)

Get an axial slice

Usage

```
function slice_data = get_ax_slice(imgs, slice_num)
```

Inputs

imgs: img filenames or spm_vols eg. 'spmT_0004.img';

slice_num: slice number eg. 31

Output

slice_data: unprocessed slice data

hewma_utility.**get_max_t**(*Zpop*, *sterr*, *tt*)

Usage

```
[t,tm,b] = get_max_t(Zpop,sterr,tt)
```

Outputs

t: t-value timeseries

tm: max t-value (abs)

b: time (index) of max t-value

hewma_utility.**hewma2** (*Zm*, *varZm*, *lambda*, *varargin*)

Usage

```
function [p,tm,Zcor,sterr,Zpop,tvals,sb,stats] = hewma2(Zm, varZm, lambda,[doplot],[dodetrend],[
```

Hierarchical (Group) Ewma - HEWMA

Inputs

Zm: data matrix (nsubjects x Time)

varZm: variance for each time point and subject

lambda: smoothing parameter

Optional Plot flag, 1 or 0

Outputs

p: p-value

tm: maximumum (over time) group t-statistic

Zcor: temporal-search corrected Z-score (max t - nullmaxt) / std(nullmax t)

Note The speed of this function is largely determined by nrep, which is

the number of repititions in the Monte carlo integration. A large value of nrep gives greater accuracy, but is slower.

Examples

```
% Generate random data where there should be a true effect and test
n = noisevector(100,[1 .9 .8 .6 .4 .1],1);
n = [zeros(1,60) n];
for i = 1:20, Zm(i,:) = n + randn(1,160);,end
varZm = ones(size(Zm));
[p,tm,Zcor,sterr,Zpop,tvals,sb,stats] = hewma2(Zm, varZm,.2,1);

% Now try with random, non-identical weights:
varZm = rand(size(Zm));

% Now try with random autocorrelated data
for i = 1:20, Zm(i,:) = noisevector(160,[1 .9 .8 .5 .2],1)';,end
[p,tm,Zcor,sterr,Zpop,tvals,sb,stats] = hewma2(Zm, varZm, .2,1);

% Add a group difference:
n = noisevector(100,[1 .9 .8 .6 .4 .1],1);
n = [zeros(1,60) n];
for i = 1:10, Zm(i,:) = n + randn(1,160);,end
n = noisevector(100,[1 .9 .8 .6 .4 .1],1);
n = [zeros(1,60) n];
for i = 11:20, Zm(i,:) = n + randn(1,160);,end
varZm = ones(size(Zm));
[p,tm,Zcor,sterr,Zpop,tvals,sb,stats] = hewma2(Zm, varZm,.2,1,0,[ones(10,1); -1*ones(10,1)]);
```

```
    % Add linear drift, to test detrending:
    n = noisevector(100,[1 .9 .8 .6 .4 .1],1);
    n = [zeros(1,60) n];
    for i = 1:10, Zm(i,:) = n + randn(1,160) + (rand(1)-.5) * (1:160);,end
    n = noisevector(100,[1 .9 .8 .6 .4 .1],1);
    n = [zeros(1,60) n];
    for i = 11:20, Zm(i,:) = n + randn(1,160) + (rand(1)-.5) * (1:160);,end
    varZm = ones(size(Zm));
    [p,tm,Zcor,sterr,Zpop,tvals,sb,stats] = hewma2(Zm, varZm,.2,1,0,[ones(10,1); -1*ones(10,1)]);
```

hewma_utility.**hewma2_plot** (*Zpop*, *mu*, *cp*, *cp_ind*, *ooc_indices*, *maxtime*, *tthresh*, *tt*, *tm*, *q*, *Wm*, *Zm*, *sterr*, *varargin*)

Usage

```
    hewma2_plot(Zpop,mu,cp,cp_ind,ooc_indices,maxtime,tthresh,tt,tm,q,Wm,Zm,sterr,varargin)
```

see hewma2.m

hewma_utility.**hewma_extract_voxel** (*EXPT*, *coords*)

Extracts data for one voxel and runs hewma2 on it.

Usage

```
    function [dat,stats] = hewma_extract_voxel(EXPT,coords)
```

Uses plot option in hewma2 to create plots.

hewma_utility.**hewma_from_raw_timeseries** (*raw_data*, *varargin*)

Usage

```
    STATS = hewma_from_raw_timeseries(raw_data, varargin)
```

Inputs

obj: imraw_data, a N subjects x T time points matrix of data

Optional Inputs For all optional inputs, enter a keyword followed by an input value/argument (this is a standard Matlab format for entering optional arguments) e.g., 'lam', .3 to enter a lambda value of .3

Here are the optional inputs and their defaults:

lam = .2; ewma smoothing param

L = 2; ewma control limit

noisemodel = 'AR(2)'; noise structure type

base_timepts = 60; baseline time points to use

doplot = 1; plot toggle (1/0)

dodetrend = 1; hewma linear detrending toggle (1/0)

grpcontrast = []; vector of 1 or -1 values for group assignment for each subject

samprate = .5; sampling rate in Hz, 1/TR; TR = 2 by default

Note see ewma5.m and hewma2.m for more information on these inputs

hewma_utility.**hewma_gui** (*Action*, *varargin*)

Usage

```
    varargout=hewma_gui(Action,varargin)
```

To run, type "hewma" at the Matlab prompt

hewma_utility.**hewma_plot_bivariate**(*varargin*)

Visualize a change_point map stored in hewma_cp.img and a run-length map stored in hewma_runlen.img (output from hewma2)

Usage

```
    [cl,cl2] = hewma_plot_bivariate([Method],[optional overlay image])
```

and classify voxels into groupings based on locations in the bivariate space

hewma_utility.**hewma_plot_coord_btnupfcn**()

function for loading and plotting hewma data from a voxel. must have declared globals f,f2,VOL,EXPT

Usage

```
    [dat,stats,mycov] = hewma_plot_coord_btnupfcn
```

see hewma_timeseries_plot, the shell function.

hewma_utility.**hewma_plot_cpmap**(*varargin*)

Visualize a change_point map stored in hewma_cp.img (output from hewma2) and classify voxels into groupings based on CP

Usage

```
    [cl2,classes] = hewma_plot_cpmap
```

hewma_utility.**hewma_plot_runlen**()

Visualize a change_point map stored in hewma_cp.img (output from hewma2) and classify voxels into groupings based on CP

Usage

```
    [cl2,classes] = hewma_plot_runlen
```

hewma_utility.**hewma_save_timeseries**(*varargin*)

Usage

```
    cl = hewma_save_timeseries([mask to extract from or cl],[k],[[set of:grpmean,grpste,xdim,ydim]])
```

load hewma_timeseries run this to save data in clusters format

Function: extract timeseries data from a mask (and extent thr k) Gets group avg data for clusters, not individual. For indiv, use hewma_extract_voxel.m

Last args contain the data for the group for each voxel. If last args are not entered, attempts to load from hewma_timeseries.mat defaults

hewma_utility.**linear_detrending**(*Zm*)

Usage

```
    Zdt = linear_detrending(Zm)
```

Input

Zm: nsub x T matrix;

Output

Zdt: linearly detrended version of Zm (baseline retained)

hewma_utility.**timeseries_btwngroups_plot**(*dat*, *cov*, *varargin*)

Usage

```
h = timeseries_btwngroups_plot(dat,cov,[baseperiod],[dedetrend],[x],[legend text])
```

Inputs

imdat: subjects x time matrix of estimates

baseperiod: integer; removes mean of 1:baseperiod for each subject

cov: subjects x 1 contrast vector (individual/group differences)

dodetrend: linear detrending of each subject's estimates

legend text: {'name1' 'name2' ...}

Output

h: line handles

hewma_utility.**timeseries_mc_pvalue**(*varZm*, *lambda*, *Wm*, *sesq*, *df*, *tm*)

Uses Monte Carlo simulation on a timeeseries to get the null hypothesis max t-value over time and establish an appropriate statistical threshold

Usage

```
[Zcor,p,tthresh,q,C] = timeseries_mc_pvalue(varZm,lambda,Wm,sesq,df,tm)
```

Inputs

varZm: within subjects variances, subj x time

lambda: smoothing parameter in EWMA

Wm: individual case (subject) weights

sesq: squared standard error between subjects (variance of group error estimate)

df: degrees of freedom between subjects (est.)

tm: max t stat for group over time

hewma_utility.**wb_hewma_shell**(*EXPT*, *varargin*)

Usage

```
EXPT = wb_hewma_shell(EXPT,[start slice or vector of slices],[dools],[mask])
```

Run EWMA first: wb_multisubject_ewma.

Fields in EXPT needed:

- subdir
- im_files
- FILES.ewma_z
- FILES.ewma_var
- [cov]

- [mask]

hewma_utility.**weighted_reg**(*Y*, *varargin*)

Calculate weighted average using weighted linear least squares See examples below for usage

Model Y_i = 1*Ypop + noise

Inputs

Y: data matrix (nsub x T)

w: weights

varY: variance of data at each time point (nsub x T) + var between

Outputs

Ymean: weighted mean of each column of Y

dfe: error degrees of freedom, adjusted for inequality of variance (Sattherwaite) and pooled across data columns

Extended output in stats structure:

stats.t: t-values for weighted t-test

stats.p: 2-tailed p-values for weighted t-test

r: weighted correlation coeff across columns of Y

xy: weighted covariance matrix

v:

weighted variance estimates for each column of Y

- sqrt(v) is the standard error of the mean (or grp difference)

stats.fits: fits for each group (Ymean by group), low contrast weight group then high Fastest if no stats are asked for.

Computation time: For FULL stats report

•Triples from 500 -> 1000 columns of Y, continues to increase

For mean/dfe only, fast for full dataset (many columns of Y)

Examples

```
% Basic multivariate stats for 1000 columns of dat, no weighting
% Multivariate covariances are meaningful if cols of Y are organized, e.g., timeseries
[means,stats] = weighted_reg(dat(:,1:1000));

% The same, but return univariate stats only (good for large Y)
[means,stats] = weighted_reg(dat,'uni');
```

hewma_utility.**weighted_reg_old2**(*y*, *w*, *varz*)

Calculate weighted average using weighted linear least squares

Model z_i = 1*zpop + noise

Inputs

Y: data matrix (nsub x T)

w: weights

varz: variance of data at each time point (nsub x T)

Outputs

r: weighted correlation coeff across columns of y

xy: weighted covariance matrix

v: weighted variance estimates for each column of y

zp: weighted mean of each column of y

hewma_utility.**weighted_reg_oldglmfit_old**(*Y*, *varargin*)

Calculate weighted average using weighted linear least squares See examples below for usage

Model Y_i = 1*Ypop + noise

Inputs

Y: data matrix (nsub x T)

w: weights

varY: variance of data at each time point (nsub x T) + var between

Outputs

Ymean: weighted mean of each column of Y

dfe: error degrees of freedom, adjusted for inequality of variance (Sattherwaite) and pooled across data columns

Extended output in stats structure:

stats.t: t-values for weighted t-test

stats.p: 2-tailed p-values for weighted t-test

r: weighted correlation coeff across columns of Y

xy: weighted covariance matrix

v:

weighted variance estimates for each column of Y

- sqrt(v) is the standard error of the mean (or grp difference)

stats.fits: fits for each group (Ymean by group), low contrast weight group then high Fastest if no stats are asked for.

Computation time: For FULL stats report

•Triples from 500 -> 1000 columns of Y, continues to increase

For mean/dfe only, fast for full dataset (many columns of Y)

Examples

```
% Basic multivariate stats for 1000 columns of dat, no weighting
% Multivariate covariances are meaningful if cols of Y are organized, e.g., timeseries
[means,stats] = weighted_reg(dat(:,1:1000));

% The same, but return univariate stats only (good for large Y)
[means,stats] = weighted_reg(dat,'uni');
```

hewma_utility.**whole_brain_ewma**(*P*, *DX*, *TR*, *HP*, *nruns*, *numframes*, *varargin*)

Single-subject timeseries extraction and model fitting saves filtered images f* and full-model betas (model fits)

Usage

```
function Pw = whole_brain_ewma(P,DX,TR,HP,nruns,numframes,[doplot],[mask],[smoothlen],[startslic
```

Inputs

P:

list of image names, raw functionals or beta images (see "type"

below)

DX: model matrix of effects to REMOVE before EWMA, see, e.g., for FIR tor_make_deconv_mtx3.m

TR: repetition time of your study (sampling rate)

HP: high-pass filter cutoff in seconds to apply (SPM style)

nruns:

number of sessions (intercepts) OR num. of images in each sess, e.g., [169 169 172]

numframes:

number of beta images per event type, e.g., [20 20 20] for FIR model

[doplot]: optional graphics

[mask]: optional name of mask image to limit processing to brain, or []

[smoothlen]:

exponential smoothing filter length for timeseries and betas; influence is 0 after smoothlen seconds; default:**

6

[startslice]: slice to start at, default = 1

[type]: optional type of analysis to run. Options: 'full' Everything, from filtering to height/time/width, enter

raw image names and model matrix DX

'data' Filter data only and save images in f*img Enter raw image names and empty model matrix []

'htw' Smooths beta series and computes height/time/width only

Enter beta images from FIR model within condition, e.g., beta1_event1 beta2_event1 ... beta1_event2 b2_e2 ...

One possibility is to use images from SPM2 at this stage.

For a shell to run this function, see whole_brain_fir.m

The analysis sequence

Run whole_brain_fir, which mainly just calls whole_brain_filter whole_brain_filter gives you:

1. **trimmed, filtered images, saved in f*img** these can be passed to SPM2, e.g., if desired

2. **extraction of beta images for FIR model, saved in dx_beta*img** in individual subject folders; also smooths betas if desired

3. **calculation of height, delay, and width for each FIR extracted** saved on images in individual subject folders.

After running this, you'll need to collect image names for use in random effects analyses, and most likely create contrasts across conditions for differences in height, delay, and width.

to do this, first go to main model directory, with subject subfolders to save a list of height/delay/width images:

```
EXPT = get_htw_image_names(EXPT)
```

to create contrasts across those images and save in EXPT.SNPM for rfx analysis:

```
EXPT = make_htw_contrast_images(EXPT);
```

Now you can run robust regression across the images in EXPT.SNPM. The robust reg uses image names in EXPT.SNPM.P, so you may have to save images from another field (e.g., EXPT.SNPM.heightP) in the .P field. you may also want to test individual contrasts against zero, in which case you can use EXPT.NLCON.height images, for example (and others in NLCON) instead.

```
EXPT = robfit(EXPT);
```

Explanations of some more variables

vb: [optional] verbose output level: 0 none, 1 some, 2 lots

mask: [optional] mask 3-D volume to apply before extracting

P: image file names (str matrix)

S: filtering matrix (e.g., high-pass)

DX: full model to fit, unfiltered

vb: [optional] verbose output level: 0 none, 1 some, 2 lots

mask: [optional] mask 3-D volume to apply before extracting

nsess: number of sessions (intercepts): assumes last nsess columns of DX are run intercepts to be removed before trimming

dims: dimensions of image files in data

cols: no. of columns of DX, also no. of beta images to write

SDX: smoothed (filtered) full model to fit

PSDX: pseudoinverse of filtered full model

PSDXS: PSDX * S, ready to multiply with data y for pinv(SX) * Sy

betas: 4-D array of x,y,z,column

ntrimmed: number of outliers trimmed from timeseries at each voxel

Pw: string matrix of output file names

Example

```
Pw = whole_brain_filter(d,c.model(1:166,:),2,120,5,c.numframes,1,10);
```

hewma_utility.**zero_crossing**(*M*, *T*, *XX*, *ZIU*, *ZIL*, *Z*, *mu*)

Estimate out of control points and change-point using zero-crossing method.

Usage

```
[CP2,CNTmat,TOTmat,LENmat] = zero_crossing(M,T,XX,ZIU,ZIL,Z,mu)
```

Used in ewma5.m See ewma5 for description of variables.

3.5 Image_computation_tools

Image_computation_tools.**apply_derivative_boost**(*varargin*)

Allows for recalculation of amplitude images from the fitted responses for each trial type. Necessary for calculating group statistics when using multiple basis functions

Usage

```
function apply_derivative_boost(varargin)
```

NOTES DB estimation only works for 2 specific basis sets: timeonly, with 2 parameters (canonical + time derivative) timedispersion,' with 3 parameters (canonical hrf + temporal and spatial dispersion)

This function loads the SPM.mat file in the current directory and uses the basis set specified in the loaded SPM structure.

Optional Inputs

'amplitudes' will only create amping images (combination of betas across basis functions)

'contrasts' assumes that amping images are already created; will only create contrast images

'all' will run both the amplitudes and contrasts sections

In addition, 'amplitudes' now has two separate parts:

- **The first uses Vince Calhoun's derivative boost (Calhoun, 2004) to** estimate amplitudes. NOTE: *We have not worked out the scaling yet, so I'm not sure this is working right* To turn this OFF, enter 'nodb' as an optional argument
- The second way uses our HTW code to estimate height, time to peak, width, and area under the curve (see Lindquist & Wager 2007 for simulations using a version of this code). It requires SCANlab specific functions, in SCN_Core_Support (unlike the deriv. boost). To turn this OFF, enter 'nohtw' as an optional argument

'startend' followed by starting and ending values in seconds for amplitude estimation window (for HTW estimation only). If you do not enter this, it will show you a plot and ask you to pick these values. If you enter them here as inputs, you can skip the interactive step and loop over subjects.

'condition_numbers' followed by which index numbers in your list should be used to calculate h, t, w from. You should use this if you are entering regressors of no interest, besides the intercepts.

Important for Contrasts disp('Using contrasts entered as F-contrasts. Assuming the first contrast vector in each F-contrast '

disp('is a contrast of interest across the CANONICAL basis function regressors.')

Examples

```
% RUN THIS IN COMMAND WINDOW TO BATCH
subj = dir('06*')
for i = 1:length(subj), cd(subj(i).name), apply_derivative_boost, cd('..'); end

% ANOTHER BATCH EXAMPLE:
d = dir('remi*'); d = d(cat(2, d.isdir)); [mydirs{1:length(d)}] = deal(d.name)
for i = 1:length(mydirs), cd(mydirs{i}), apply_derivative_boost('all', 'nodb', 'startend', [4 15

%An example for an event-related design, specifying condition numbers to get HTW from:
apply_derivative_boost('all', 'nodb', 'contrasts', 'condition_numbers', 1:14, 'startend', [4 10]

% CALCULATE CONTRASTS ONLY ON ALREADY-ESTIMATED HTW IMAGES
apply_derivative_boost('contrasts','condition_numbers',1:14);
```

Image_computation_tools.**canlab_create_wm_ventricle_masks**(*wm_mask*, *gm_mask*, *varargin*)

This function saves white matter and ventricle masks.

Usage

```
function canlab_create_wm_ventricle_masks(wm_mask, gm_mask)
```

Inputs

wm_mask: white matter structural image file

```
wm_mask = filenames('Structural/SPGR/wc2*.nii', 'char', 'absolute');
```

gm_mask: gray matter structural image file

```
gm_mask = filenames('Structural/SPGR/wc1*.nii', 'char', 'absolute');
```

Optional You can specify how liberal or conservative to be in estimating white matter and ventricles. 1 is most conservative and will yield no ventricles, and 0 is very liberal. The default of wm_thr is .9, the default of vent_thr is .9. e.g)

- 'wm_thr', .99
- 'vent_thr', .95

Output

"white_matter.img" and "ventricles.img": in the same folder of the input structural files

Image_computation_tools.**fisherp**(*p*, *varargin*)

Usage

```
function [z,p,sig,pt] = fisherp(p,[alph])
```

Inputs

p: values in 4-D array

1st 3 dims are within images, dim4 = image

Optional alpha value for thresholding

Outputs

z: Fisher's combined test statistic, compare to normal

p: p-values for combined test

sig: signficance 1 / 0 binary mask, p < .05 (or alph) FDR-corr

pt: p-value threshold for FDR corrected significance at alph

Described in

Lazar, N. A., Luna, B., Sweeney, J. A., & Eddy, W. F. (2002). Combining brains: a survey of methods for statistical pooling of information. Neuroimage, 16(2), 538-550.

Stouffer, S. A., Suchman, E. A., DeVinney, L. C., Star, S. A., and Williams, R. M. 1949. The American Soldier: Vol. I. Adjustment During Army Life. Princeton University Press, Princeton.

Threshold is determined with False Discovery Rate (Benjamini & Hochberg, 1995)

Image_computation_tools.**get_mask_vol**(*Pf*, *Pm*)

Returns 3-D volume of binary mask values, in the space of the functional image Pf.

Usage

```
function mask = get_mask_vol(Pf,Pm)
```

Inputs

Pf: filename of functional image

Pm: filename of mask image (should be 1's and 0's)

Image_computation_tools.**image_eval_function**(*imageNames*, *fhandle*, *varargin*)

Evaluate any function, defined by the function handle fhandle, on each in-mask voxel for a set of images.

Usage

```
varargout = image_eval_function(imgnames, fhandle, ['mask', mask], ['preprochandle', preprochand
```

Other Optional args 'outimagelabels' , 'connames'

```
varargout = image_eval_function(Y, fhandle, varargin)
```

evaluate fhandle on paired columns of X and Y

Note You must call image_eval_function with outputs, one output for

each output you're requesting from the voxel-level function. Eg:

```
[t, df, betaorcontrast, Phi, sigma, stebeta, F, pvals] = ...
image_eval_function(imgs, fhandle, 'mask', maskimg, 'outimagelabels' , names);
```

Inputs

fhandle: is a function handle:

```
fhandle = @(variable inputs) fit_gls(variable_input, fixed_inputs);

fhandle = @(y) fit_gls(y, X, c, p, PX);
```

'outimagelabels': should be followed by a set of image names, one name per output of fhandle per element. e.g., outnames{j}{i} is the output image for output j and element (input image) i. elements may be images for each subject, if a set of one image per subject is entered, or something else depending on the nature of input imgs.

Note: do not include suffixes: no .img

Examples

Generalized least squares fitting on 100 Y-variables, same X

```
% Get image list
imgs = filenames('trial_height*img', 'char')
imgs = sort_image_filenames(imgs)

% Get pre-stored design matrix
X = eventdesign{3};

preprochandle = @(y) trimts(y, 3, []);
```

Generate an image with the number of in-analysis (valid) subjects in each voxel

EXPT.SNPM.P{2} is a list of subject-level contrast images.

```
fhan = @(y) sum(abs(y) > 0 & ~isnan(y));
y = image_eval_function(EXPT.SNPM.P{2}, fhan, 'mask', EXPT.mask, 'outimagelabels', {{'sum_valid_

y = rand(100, 100); X = y + rand(100, 1); X(:,end+1) = 1; c = [1 0]'; p = 2; PX = pinv(X);
fhandle = @(y) fit_gls(y, X, c, p, PX);
[t, df, beta, Phi, sigma, stebeta, F] = fhandle(y);
```

setup inputs

Image_computation_tools.**image_eval_function_multisubj**(*imageNames*, *fhandle*, *varargin*)

Evaluate any function, defined by the function handle fhandle, on each in-mask voxel for a set of images. imageNames is a cell array of N cells, each containing images for one replication (i.e., subject)

Usage

```
image_eval_function_multisubj(imgnames,fhandle,['mask',mask],['preprochandle',preprochandle],['c
```

other optional args: 'outimagelabels' , 'connames'

At each voxel, a cell array is formed, one cell per subject. This would correspond to a matrix is formed of t x N, where t is time and N is replication (subject) but cells can deal with unequal data vector lengths for each subject. The anonymous function in fhandle should operate on data from each cell (subject).

```
varargout = image_eval_function(Y,fhandle,varargin)
```

evaluate fhandle on paired columns of X and Y

fhandle is a function handle:

```
fhandle = @(variable inputs) fit_gls(variable_input,fixed_inputs);
fhandle = @(y) fit_gls(y,X,c,p,PX);
```

specify the outputs by adding them as output image names. The number of outputs returned is determined by the number of named images in the list entered following the 'outnames' keyword.

Note on output images You specify the names of the output images

One image will be written per output of fhandle. The images will have one volume per element of the output variable. If you are returning an output with one value per subject, for example, then a single image will be written with one volume in it per subject.

preprochandle is a function handle. it encapsulates the preprocessing function. the function should work on each cell (subject) of a t x N cell array of time courses for each subject, where each cell contains a t x v matrix

of data from a slice. The preproc function should thus be able to handle a whole slice as input. the function can itself be a cell array with multiple handles in different cells

Examples Generalized least squares fitting on 100 Y-variables, same X

```
% Get image list
imgs = filenames('trial_height*img','char')
imgs = sort_image_filenames(imgs)

% Get pre-stored design matrix
X = eventdesign{3};

preprochandle = @(y) trimts(y,3,[]);

y = rand(100,100); X = y + rand(100,1); X(:,end+1) = 1; c = [1 0]'; p = 2; PX = pinv(X);
fhandle = @(y) fit_gls(y,X,c,p,PX);
[t, df, beta, Phi, sigma,stebeta, F] = fhandle(y);
```

Image_computation_tools.**image_histogram1d**(*varargin*)

Visualize a change_point map stored in hewma_cp.img (output from hewma2) and classify voxels into groupings based on CP

Usage

```
[cl2,classes] = image_histogram1d([image name],[overlay])
```

Image_computation_tools.**mask_create_from_image_set**(*imgs*, *outname*, *atleastN*, *varargin*)

Take a set of images and create a mask of voxels in which at least N subjects have valid (not exactly zero, non NaN) data.

Usage

```
mask = mask_create_from_image_set(imgs, outname, atleastN, ['sum'])
```

This makes a useful results mask for a set of images, i.e., in a group analysis.

Optional 'sum' input writes the sum image instead of the mask image,

so that the values in the image reflect the number of input images with valid values.

compatible with SPM5 and above only!

Examples

```
mask_create_from_image_set(EXPT.SNPM.P{1}, 'mask_all_valid.img');

imgs = filenames('vascular_mask_*img');
mask_create_from_image_set(imgs, 'vascular_group_sum.img', 6, 'sum');
```

Image_computation_tools.**mask_create_results_mask**(*mask*, *study_img*, *kern_size*, *varargin*)

Takes a mask image name (mask) smooths it by smooth_mm (optional; enter 0 for kern_size to avoid this) Resamples it to study image dimensions (study_img) writes output image

Usage

```
[cl_out, out_name] = mask_create_results_mask(mask, study_img, kern_size, [opt: expression to ev
```

Optional Applies an expression to be evaluated to the image,

in spm_imcalc format, e.g., 'i1 < .05' (uses spm_imcalc_ui.m)

Outputs Returns mask_clusters (mask_cl) and mask name (out_name)

Examples

```
mask = which('spm2_amy.img')
study_img = '/Users/tor/Documents/Tor_Documents/PublishedProjects/inpress_2007_Emotion_handbook_
kern_size = 3;
cl_out = mask_create_results_mask(mask, study_img, kern_size);
cluster_orthviews(cl_out, {[0 1 0]}, 'add', 'handle', 1);

[cl_out, out_name] = mask_create_results_mask('X-Y_total_pvals.img', 'X-Y_total_pvals.img', 0, '
spm_image('init', 'X-Y_total_pvals.img');
cluster_orthviews(cl_out, {[1 0 0]}, 'trans', 'add');
```

Image_computation_tools.**mask_fisher**(*clsize*, *outname*, *k*, *Pimg*, *Timg*)

This function performs combination of p-values across images using the Fisher method (see refs below). Voxels are thresholded with FDR, and can be positive and negative at once if both positive and negative sig effects exist in input images. FDR correction is based on 2-tailed p-values - so this function assumes 2-tailed p-values!! (robfit does this, glmfit and robustfit both return 2-tailed p-values as well.)

Usage

```
function [clpos,clneg] = mask_fisher(clsize,outname,k,Pimg,Timg)
```

Assumes input p-values are 1-tailed, and divides by two to get 2-tailed p-value inputs!!

Inputs

Pimg: string mtx of p-value image names

```
Pimg=get_filename2('rob*\*_p_*0002.img'); Timg = get_filename2('rob*\*_tmap_0002.img
```

Timg: t-value img names, used to ensure signs are same across all tests

k: num voxels/num comparisons **NOT USED NOW - FDR USED INSTEAD if empty, uses # of non-zero, non-NaN values in images

empty i1 prompts for graphic selection of filenames extra arguments are more file names for 3 - n-way intersection empty outname prompts for entry of output img file name

Described in

Lazar, N. A., Luna, B., Sweeney, J. A., & Eddy, W. F. (2002). Combining brains: a survey of methods for statistical pooling of information. Neuroimage, 16(2), 538-550.

Stouffer, S. A., Suchman, E. A., DeVinney, L. C., Star, S. A., and Williams, R. M. 1949. The American Soldier: Vol. I. Adjustment During Army Life. Princeton University Press, Princeton.

Image_computation_tools.**mask_image**(*img*, *mask*, *outname*, *varargin*)

Usage

```
masked_dat = mask_image(img, mask, outname, ['reverse'])
```

Mask an image file (img) with a mask (mask), and save in outname. zero and NaN values are considered invalid values, and so voxels with these values are considered "excluded"

Optional Inputs

'minmask': mask values less than next argument are excluded

'maxmask': mask values greater than next argument are excluded

'minimg': img values less than next argument are excluded

'maximg': img values greater than next argument are excluded

'abs': impose min/max thresholds based on absolute values

'reverse':

make "reverse mask," including only previously excluded areas (values of zero or NaN)

Note: applies to mask, not img values So values with 0 in img will always be 0, whether

standard or "reverse" mask is used.

'binary': make the image values binary (i.e., create a new mask)

This function can handle images of different dimensions. The output image will use the dimensions of img.

Examples

```
% Create an image with non-zero numbers only where p-values in an image are greater than .05
img = 'X-M_pvals.img'
mask = 'X-M_pvals.img';
maxmask = .05
outname = 'notX_p05.img';
mask_image(img, mask, outname, 'reverse', 'maxmask', maxmask);
spm_image('init', outname);

mask_image(my_mean_image, 'functional_mask.img', ...
               'functional_mask.img', 'minimg', cutoff, 'abs');


mask_image('n15_avgpet.img',EXPT.mask,'n15_avgpet_brain.img');
```

Image_computation_tools.**mask_intersection**(*clsize*, *outname*, *i1*, *i2*, *varargin*)

empty i1 prompts for graphic selection of filenames extra arguments are more file names for 3 - n-way intersection empty outname prompts for entry of output img file name

Usage

```
function [vol,V,XYZ,clusters,Q] = mask_intersection(clsize,outname,i1,i2,varargin)
```

Image_computation_tools.**mask_intersection2**(*clsize*, *outname*, *P*, *calcstr*)

empty P prompts for graphic selection of filenames extra arguments are more file names for 3 - n-way intersection empty outname prompts for entry of output img file name

Usage

```
function [clusters,vol,V,XYZ,Q] = mask_intersection2(clsize,outname,P,calcstr)
```

this version allows contrasts last argument, calcstr, is the string to evaluate in imcalc

e.g., intersection of 4 images:

```
'i1 & i2 & i3 & i4 & ~(isnan(i1) | isnan(i2) | isnan(i3) | isnan(i4))'
```

e.g.,

```
'i1 & i2 & ~i3 & ~i4 & ~(isnan(i1) | isnan(i2)) = intersection of i1 and i2 and not i3 and not i
```

or

```
      'i1 & i2 & isnan(i3) & isnan(i4) & ~(isnan(i1) | isnan(i2))'
    ... for nan masked images

       cl = mask_intersection2(5,'intersect001.img',P,'i1 & i2 & i3 & i4 & ~(isnan(i1) | isnan(i2) |
```

Image_computation_tools.**mask_stouffer**(*clsize*, *outname*, *k*, *Pimg*, *Timg*)

This function performs combination of p-values across images using the Stouffer method (see refs below).

Usage

```
function [vol,V,XYZ,clusters,Q] = mask_stouffer(clsize,outname,k,Pimg,Timg)
```

Inputs

Pimg: string mtx of p-value image names

Timg: t-value img names, used to ensure signs are same across all tests

k: num voxels/num comparisons if empty, uses # of non-zero, non-NaN values in images

empty i1 prompts for graphic selection of filenames extra arguments are more file names for 3 - n-way intersection empty outname prompts for entry of output img file name

Described in

Lazar, N. A., Luna, B., Sweeney, J. A., & Eddy, W. F. (2002). Combining brains: a survey of methods for statistical pooling of information. Neuroimage, 16(2), 538-550.

Stouffer, S. A., Suchman, E. A., DeVinney, L. C., Star, S. A., and Williams, R. M. 1949. The American Soldier: Vol. I. Adjustment During Army Life. Princeton University Press, Princeton.

Examples

```
Pimg=get_filename2('rob*\*_p_*0002.img');
Timg = get_filename2('rob*\*_tmap_0002.img')
```

Image_computation_tools.**mask_union**(*clsize*, *outname*, *i1*, *i2*, *varargin*)

empty i1 prompts for graphic selection of filenames extra arguments are more file names for 3 - n-way intersection empty outname prompts for entry of output img file name

Usage

```
function [vol,V,XYZ,clusters,Q] = mask_union(clsize,outname,i1,i2,varargin)
```

Image_computation_tools.**percent_sig_image**(*imgs*, *baseimg*, *outname*)

Creates a percent signal change image saved in outname by dividing each image by baseimg

Usage

```
function Vo = percent_sig_image(imgs, baseimg, outname)
```

Image_computation_tools.**reslice_imgs**(*sampleTo*, *resliceThis*, *varargin*)

arguments are file names of .img files if empty, select from GUI% :Usage:

```
function [P, reslicedImgs] = reslice_imgs(sampleTo, resliceThis, [domask], [overwrite])
```

if domask, recalculates a 1 or 0 mask for each image file in resliceThis if overwrite, overwrite the original instead of prepending an 'r'

Examples

```
    % Reslice a mask image into the space of some functional images, and move to the current directo

    [tmp, maskname] = reslice_imgs(image_names(1, :), maskname, 1);
    eval(['!mv ' maskname ' ./'])
    eval(['!mv ' maskname(1:end-4) '.hdr ./'])
```

Image_computation_tools.**reverse_mask**(*inname*, *outname*)

Changes 1 and greater's to 0's in an .img file, and vice versa NaNs are still NaNs.

Usage

```
    reverse_mask(inname,outname)
```

Image_computation_tools.**scn_write_plane**(*filenames_or_V*, *dat*, *wh_slice*, *varargin*)

Images can be 3D or 4D (4D is possible with SPM2+ using ,xx indexing, or SPM5+).

Usage

```
    V = scn_write_plane(filenames_or_V, dat, wh_slice, [exampleV])
```

Inputs

ofilenames_or_V: is string matrix or cell array of names, or structures created with spm_vol(names)

dat: is 3-D array of data, vox x vox x slices, with data on 3rd dim being written to separate image files

wh_slice: slice number (voxel space)

exampleV: is optional, but required if filenames are passed in.

Write a plane, given filenames or spm_vol structures and 4-D data with the slice to write.

SPM2/5 compatible, and creates image if necessary from file name and example V structure.

```
    P = char({'one.img', 'two.img', 'three.img'})
    dat = randn(64, 64, 3);
    Vout = scn_write_plane(P, dat, wh_slice, V)
    spm_image('init', Vout(1).fname);
```

Notes on SPM5 and why we need to do what we do the way we do it: Write data...in SPM5, by accessing file_array object in V.private directly (with spm_write_plane. spm_write_vol first creates NIFTI obj/file array and then uses spm_write_plane.)

The file_array object points to data in the actual file, so when values are assigned to the array object, they are written directly in the file. These values depend on the offset and slope values (spm scaling factors!) in V.private.dat as well, so care must be taken to assign data to a file_array object with the correct scaling factors. This is why it is better to load the structure one wants to write to with spm_vol first, so that the name in V.private.dat.fname will be correct, and V.private.dat.scl_slope and ...inter will be correct as well. Vout = spm_vol(V(i)); % loads correct .private info from V.fname

in SPM5, this simply assigns data in Vout.private.dat in SPM2, it does something different, but should be compatible, since spm_vol was used above...

```
    spm_write_plane(Vout, fsl(:, :, i), slicei);
    if isfield(V, 'dt'), Vout.dt = V.dt; end      % SPM5 only
    if isfield(V, 'n'), Vout.n = V.n;  end        % SPM5 only
        Vout = spm_create_vol(Vout);
    else
        Vout = spm_vol(V(i).fname);
```

```
    end
    spm_write_plane(Vout, fsl(:, :, i), slicei);
```

Image_computation_tools.**tor_spm_mean_ui**(*P*, *varargin*)

Promts for a series of images and averages them

Usage

```
    tor_spm_mean_ui(Pinputnames,[outputname])
```

3.6 Image_space_tools

Image_space_tools.**check_spm_mat**(*mat1*, *mat2*, *clusters*)

mat1 is from clusters, mat2 is functional (imgs to extract)

Usage

```
    check_spm_mat(mat1,mat2,clusters)
```

Image_space_tools.**check_spm_matfiles**(*P*)

Check a list of SPM-style images to see if they all have the same space, dims.

Usage

```
    [anybad] = check_spm_matfiles(P)
```

Image_space_tools.**img2voxel**(*P*, *varargin*)

Given a mask or filtered image file name, returns XYZ coordinates in voxels and mm of nonzero, non-NaN voxels

and img values at these coordinates in val

Image_space_tools.**mask2voxel**(*mask*)

convert from 3-D mask to voxel list in canonical orientation

Usage

```
    function voxels = mask2voxel(mask)
```

[i j k] = row, column, slice

[x y z] in brain if brain is in analyze format

(x is rows, y is columns, z is slices)

Image_space_tools.**mni2tal**(*inpoints*)

Converts coordinates from MNI brain to best guess for equivalent Talairach coordinates

Usage

```
    outpoints = mni2tal(inpoints)
```

Where inpoints is N by 3 or 3 by N matrix of coordinates (N being the number of points)

Output

outpoints: is the coordinate matrix with Talairach points

Image_space_tools.**scn_map_image**(*loadImg*, *sampleTo*, *varargin*)

This function takes an image name in loadImg and loads the data, resampling to the space defined in the image sampleTo. The resampled image will retain the data type of the input image.

Usage

```
[imgData, volInfo_mapto] = scn_map_image(loadImg, sampleTo, varargin)
```

Optional Inputs

'write': followed by name of resampled image to write

Compatible with SPM5/8.

Input images can have the following formats:

1. String with name of image file (.img or .nii)
2. spm_vol-style V struct (see spm_vol)
3. volInfo struct (see iimg_read_img)
4. fmri_mask_image object (see fmri_mask_image)

Examples

```
img = scn_map_image(EXPT.mask, EXPT.SNPM.P{1}(1,:), 'write', 'resliced_mask.img');
```

Image_space_tools.**scn_resample_voxel_size**(*loadImg*, *voxsize*, *varargin*)

This function takes an image name in loadImg and loads the data, resampling to the space defined in the image sampleTo.

Usage

```
[imgData, volInfo_mapto] = scn_resample_voxel_size(loadImg, voxsize, varargin)
```

take volInfo and fmri_mask_image inputs as well as image file names

Optional Inputs

'write': followed by name of resampled image to write

Compatible with SPM5/8.

Input images can have the following formats:

1. String with name of image file (.img or .nii)
2. spm_vol-style V struct (see spm_vol)
3. volInfo struct (see iimg_read_img)
4. fmri_mask_image object (see fmri_mask_image)

Examples

```
% Reslice standard brain mask to 3 x 3 x 3 voxels.
img = which('brainmask.nii');
[dat, Vto] = scn_resample_voxel_size(img, [3 3 3], 'write', 'test.img');
spm_image('init', 'test.img');
spm_check_registration(char(img, 'test.img'));
```

Image_space_tools.**tal2mni**(*inpoints*)

Converts coordinates to MNI brain best guess from Talairach coordinates

Usage

```
outpoints = tal2mni(inpoints)
```

Where inpoints is N by 3 or 3 by N matrix of coordinates (N being the number of points)

Output

outpoints: is the coordinate matrix with MNI points

Image_space_tools.**tal2vox**(*tal*, *VOL*)

converts from talairach coordinate to voxel coordinate based on variables from SPM.M (passed here for faster operation)

Example

```
foo = tal2vox([-30 28 -30], VOL)
```

Image_space_tools.**transform_coordinates**(*CLU*, *mat*)

Transforms XYZ voxel coordinates in CLU (clusters or CLU) to new voxel coordinates, given mm coordinates in CLU and a mat file describing the transformation, as in SPM99

Usage

```
CLU = transform_coordinates(CLU,mat)
```

This preserves the order of the voxels, but is slower and gives UNIQUE XYZ voxels given XYZmm. see mm2voxel.m

Image_space_tools.**voxel2mask**(*voxels*, *maskdims*)

Usage

```
function mask = voxel2mask(voxels, x y z mask dimensions)
```

Voxels

- 3 column vectors
- [i j k] = row, column, slice
- [x y z] in brain if brain is in analyze format

  (x is rows, y is columns, z is slices)

Image_space_tools.**voxel2mm**(*XYZ*, *m*)

Usage

```
function XYZmm = voxel2mm(XYZ,m)
```

Inputs

XYZ: is 3 vector point list (3 rows, n columns)

m: is SPM mat - 4 x 4 affine transform (what's stored in the .mat file)

Example

```
XYZmm = voxel2mm([x y z]',V.mat);
```

3.7 Image_thresholding

Image_thresholding.**FDR**(*p*, *q*)

Usage

```
pt = FDR(p,q)
```

Inputs

p: vector of p-values

q: False Discovery Rate level

Outputs

pID: p-value threshold based on independence or positive dependence

pN: Nonparametric p-value threshold

Image_thresholding.**cl_ext_3dClustSim**(*corrected_p*, *prim_p*, *residual_images*, *mask*, *voxelsize_mm*, *ClustSim_dir*, *varargin*)

Usage

```
[cl_ext_ClustSim, fwhm] = cl_ext_3dClustSim(corrected_p, prim_p, residual_images, mask, voxelsiz
```

Inputs

corrected_p: cluster-extent corrected p value

e.g.) if cluster-extent corrected p < .05: corrected_p = .05

prim_p: primary threshold for height (i.e., cluster-defining threshold)

e.g.) prim_p = [0.01 0.005 0.001];

residual_images: residual image names; if you used

cl_ext_make_resid.m, this should be 'Res4d.nii'.

e.g.) residual_images = filenames('Res4d.hdr', 'char', 'absolute');

residual_images = filenames('Res4d.nii', 'char', 'absolute');

mask: mask image name (should have header)

e.g.) mask = filenames('mask.hdr', 'char', 'absolute');

mask = filenames('mask.nii', 'char', 'absolute');

voxelsize_mm: voxel sizes in milimeter. e.g) voxelsize_mm = [2 2 2];

3dClustSim_dir: directory where alphasim is installed.

e.g.) 3dClustSim_dir = '/Users/clinpsywoo/abin/macosx_10.6_Intel_64';

If you don't have 3dClustSim, see http://afni.nimh.nih.gov/pub/dist/HOWTO/howto/ht00_inst/html/index.shtml

Optional Inputs

'iter': you can set up the iteration number for Monte Carlo simulation. default is doing 1000 iterations.

'twotail': default is one-tail - with this option, primary_p/2 will be used for all clsuter extent estimations.

'fwhm': you can add fwhm manually

Outputs

cl_ext_ClustSim: cl_ext_ClustSim is the cluster size that makes a corrected p value under corrected_p (e.g., 0.05).

fwhm (x, y, z in voxel): intrinsic smoothness level estimated by AFNI(3dFWHMx). If you want to convert this into mm, you need to multiply these values by voxel sizes in mm.

Image_thresholding.**cl_ext_make_resid**(*con_files*, *varargin*)

This function will create residual images (4d) and mask image in a current or assigned directory in order to use them in estimating smoothness (relevant functions: spm_est_smoothness (SPM), 3dFWHMx (AFNI), smoothest (FSL).

Usage

```
function cl_ext_make_resid(conimgs, varargin)
```

Inputs

con_files: contrast image file names; This could be a cell array or strings. This could be 4d images.

Best: Input a cell string. e.g., for a string matrix:

Use cl_ext_make_resid(cellstr(imgs)); % save residual images

- If you are not providing the absolute paths of the images, you need to

be in the directory that has the image files.

Outputs

Res4d.nii: residual images saved by SPM.

mask.nii: the mask image that was used.

Options for varargin

'mask' This option can be used to estimate a cluster size for the correction for multiple comparisons "within the mask". You can put in a ROI mask or gray matter, whatever. If you don't specify a mask image, brainmask.nii (default) will be used, but the image has to be in your path.

e.g.)

```
mask = fullfile(basedir, 'ROI_image.img');
mask = which('scalped_avg152T1_graymatter_smoothed.img'); % limited to gray matte
```

'outputdir' With this option, this will save residual and mask images and in the outputdir directory. If you don't give outputdir, the current directory will be used (default).

This function calls cl_ext_spm_spm.m, which is a modified spm_spm not to delete residual images.

Image_thresholding.**cl_ext_spm_grf**(*corrected_p*, *prim_p*, *residual_images*, *mask*, *varargin*)

This function is designed to estimate a cluster extent size for the correction for multiple comparisons based on a Gaussian Random Field Theory using SPM toolboxes.

Usage

```
[cl_ext, fwhm] = cl_ext_spm_grf(corrected_p, prim_p, residual_images, mask, varargin)
```

Inputs

corrected_p: corrected p value

e.g.) cluster-extent corrected p < .05: corrected_p = .05

prim_p: primary threshold for height (i.e., cluster-defining threshold)

e.g.) prim_p = [0.01 0.005 0.001];

residual_images: residual image names; if you used

cl_ext_make_resid.m, this should be 'Res4d.nii'

mask: mask image name

Optional Inputs 'doplot':

'twotail': default is one-tail - with this option, primary_p/2 will be used for all clsuter extent estimations.

Output:

cl_ext_spm:

cl_ext_spm is the cluster size that makes a corrected p value under

corrected_p (e.g., 0.05).

fwhm (x, y, z in voxels):

intrinsic smoothness level estimated by SPM (spm_est_smoothness.m)

If you want to convert this into mm, you need to multiply these values by voxel sizes in mm.

Image_thresholding.**cl_ext_spm_spm**(*SPM*)

[Re]ML Estimation of a General Linear Model

Usage

```
FORMAT [SPM] = spm_spm(SPM)
```

Required fields of SPM

xY.VY - nScan x 1 struct array of image handles (see spm_vol)

Images must have the same orientation, voxel size and data type

- Any scaling should have already been applied via the image handle scalefactors.

xX - Structure containing design matrix information

- Required fields are: xX.X - Design matrix (raw, not temporally smoothed) xX.name - cellstr of parameter names corresponding to columns

  of design matrix

- Optional fields are: xX.K - cell of session-specific structures (see spm_filter)
  - Design & data are pre-multiplied by K (K*Y = K*X*beta + K*e)
  - Note that K should not smooth across block boundaries

  - defaults to speye(size(xX.X,1))

xX.W - Optional whitening/weighting matrix used to give weighted least squares estimates (WLS). If not specified spm_spm will set this to whiten the data and render the OLS estimates maximum likelihood i.e. W*W' = inv(xVi.V).

xVi - Structure describing intrinsic temporal non-sphericity

- Required fields are: xVi.Vi - array of non-sphericity components
  - defaults to {speye(size(xX.X,1))} - i.i.d.
  - specifying a cell array of constraints (Qi) These constraints invoke spm_reml to estimate hyperparameters assuming V is constant over voxels. that provide a high precise estimate of xX.V
- Optional fields are: xX.V - Optional non-sphericity matrix. Cov(e) = sigma^2*V

  If not specified spm_spm will compute this using a 1st pass to identify significant voxels over which to estimate V. A 2nd pass is then used to re-estimate the parameters with WLS and save the ML estimates (unless xX.W is already specified).

xM - Structure containing masking information, or a simple column vector

of thresholds corresponding to the images in VY [default: -Inf]

- If a structure, the required fields are: xM.TH - nVar x nScan matrix of analysis thresholds, one per image xM.I - Implicit masking (0=>none, 1 => implicit zero/NaN mask) xM.VM - struct array of explicit mask image handles
- **(empty if no explicit masks)**
  - Explicit mask images are >0 for valid voxels to assess.
  - Mask images can have any orientation, voxel size or data type. They are interpolated using nearest neighbour interpolation to the voxel locations of the data Y.
- Note that voxels with constant data (i.e. the same value across scans) are also automatically masked out.

swd - Directory where the output files will be saved [default: pwd] If exists, it becomes the current working directory.

In addition, global SPM "defaults" variable is used (see spm_defaults):

stats.<modality>.UFp - critical F-threshold for selecting voxels over which the non-sphericity is estimated (if required) [default: 0.001]

stats.maxres - maximum number of residual images for smoothness estimation

stats.maxmem - maximum amount of data processed at a time (in bytes)

modality - SPM modality {'PET','FMRI','EEG'}

spm_spm is the heart of the SPM package. Given image files and a General Linear Model, it estimates the model parameters, variance hyperparameters, and smoothness of standardised residual fields, writing these out to disk in the current working directory for later interrogation in the results section. (NB: Existing analyses in the current working directory are overwritten). This directory now becomes the working directory for this analysis and all saved images are relative to this directory.

The model is expressed via the design matrix (xX.X). The basic model at each voxel is of the form is Y = X*B + e, for data Y, design matrix X, (unknown) parameters B and residual errors e. The errors are assumed to have a normal distribution.

Sometimes confounds (e.g. drift terms in fMRI) are necessary. These can be specified directly in the design matrix or implicitly, in terms of a residual forming matrix K to give a generalised linear model K*Y = K*X*B + K*e. In fact K can be any matrix (e.g. a convolution matrix).

In some instances i.i.d. assumptions about errors do not hold. For example, with serially correlated (fMRI) data or correlations among the levels of a factor in repeated measures designs. This non-sphericity can be specified in terms of components (SPM.xVi.Vi{i}). If specified these covariance components will then be estimated with ReML (restricted maximum likelihood) hyperparameters. This estimation assumes the same non-sphericity for voxels that exceed the global F-threshold. The ReML estimates can then be used to whiten the data giving maximum likelihood (ML) or Gauss-Markov estimators. This entails a second pass of the data with an augmented model K*W*Y = K*W*X*B + K*W*e where W*W' = inv(xVi.V). xVi.V is the non-sphericity based on the hyperparameter estimates. W is stored in xX.W and cov(K*W*e) in xX.V. The covariance of the parameter estimates is then xX.Bcov = pinv(K*W*X)*xX.V*pinv(K*W*X)'.

If you do not want ML estimates but want to use ordinary least squares (OLS) then simply set SPM.xX.W to the identity matrix. Any non-sphericity V will still be estimated but will be used to adjust the degrees of freedom of the ensuing statistics using the Satterthwaite approximation (c.f. the Greenhouse-Geisser corrections).

If [non-spherical] variance components Vi are not specified xVi.Vi and xVi.V default to the identity matrix (i.e. i.i.d). The parameters are then estimated by OLS. In this instance the OLS and ML estimates are the same.

Note that only a single voxel-specific hyperparameter (i.e. variance component) is estimated, even if V is not i.i.d. This means spm_spm always implements a fixed-effects model. Random effects models can be emulated using a multi-stage procedure: This entails summarising the data with contrasts such that the fixed effects in a second model on the summary data are those effects of interest (i.e. the population effects). This means contrasts are re-entered into spm_spm to make an inference (SPM) at the next level. At this higher hierarchical level the residual variance for the model contains the appropriate variance components from lower levels. See spm_RandFX.man for further details and below.

Under the additional assumption that the standardised error fields are non-stationary standard Gaussian random fields, results from Random field theory can be applied to estimate the significance statistic images (SPM's) adjusting p values for the multiple tests at all voxels in the search volume. The parameters required for this random field correction are the volume, and Lambda, the covariance matrix of partial derivatives of the standardised error fields, estimated by spm_est_smoothness.

The volume analysed is the intersection of the threshold masks, explicit masks and implicit masks. See spm_spm_ui for further details on masking options.

The output of spm_spm takes the form of an SPM.mat file of the analysis parameters, and 'float' flat-file images of the parameter and variance [hyperparameter] estimates. An 8bit zero-one mask image indicating the voxels assessed is also written out, with zero indicating voxels outside tha analysed volume.

The following SPM.fields are set by spm_spm (unless specified)

xVi.V - estimated non-sphericity trace(V) = rank(V) xVi.h - hyperparameters xVi.V =

xVi.h(1)*xVi.Vi{1} + ... xVi.Cy - spatially whitened <Y*Y'> (used by ReML to estimate h) xVi.CY - <(Y - <Y>)*(Y - <Y>)'> (used by spm_spm_Bayes)

Vbeta - struct array of beta image handles (relative) VResMS - file struct of ResMS image handle (relative) VM - file struct of Mask image handle (relative)

xX.W - if not specified W*W' = inv(x.Vi.V) xX.V - V matrix (K*W*Vi*W'*K') = correlations after K*W is applied xX.xKXs - space structure for K*W*X, the 'filtered and whitened'

design matrix

•given as spm_sp('Set',xX.K*xX.W*xX.X) - see spm_sp

xX.pKX - pseudoinverse of K*W*X, computed by spm_sp xX.Bcov - xX.pKX*xX.V*xX.pKX - variance-covariance matrix of

parameter estimates (when multiplied by the voxel-specific hyperparameter ResMS of the parameter estimates (ResSS/xX.trRV = ResMS))

xX.trRV - trace of R*V xX.trRVRV - trace of RVRV xX.erdf - effective residual degrees of freedom (trRV^2/trRVRV) xX.nKX - design matrix (xX.xKXs.X) scaled for display

(see spm_DesMtx('sca',... for details)

xVol.M - 4x4 voxel->mm transformation matrix xVol.iM - 4x4 mm->voxel transformation matrix xVol.DIM - image dimensions - column vector (in voxels) xVol.XYZ - 3 x S vector of in-mask voxel coordinates xVol.S - Lebesgue measure or volume (in voxels) xVol.R - vector of resel counts (in resels) xVol.FWHM - Smoothness of components - FWHM, (in voxels)

xCon - Contrast structure (created by spm_FcUtil.m) xCon.name - Name of contrast xCon.STAT - 'F', 'T' or 'P' - for F/T-contrast ('P' for PPMs) xCon.c - (F) Contrast weights xCon.X0 - Reduced design matrix (spans design space under Ho)

It is in the form of a matrix (spm99b) or the coordinates of this matrix in the orthogonal basis of xX.X defined in spm_sp.

xCon.iX0 - Indicates how contrast was specified: If by columns for reduced design matrix then iX0 contains the column indices. Otherwise, it's a string containing the spm_FcUtil 'Set' action: Usually one of {'c','c+','X0'} (Usually this is the input argument F_iX0.)

xCon.X1o - Remaining design space (orthogonal to X0). It is in the form of a matrix (spm99b) or the coordinates of this matrix in the orthogonal basis of xX.X defined in spm_sp.

xCon.eidf - Effective interest degrees of freedom (numerator df) xCon.Vcon - ...for handle of contrast/ESS image (empty at this stage) xCon.Vspm - ...for handle of SPM image (empty at this stage)

The following images are written to file

mask.{img,hdr} - analysis mask image 8-bit (uint8) image of zero-s & one's indicating which voxels were included in the analysis. This mask image is the intersection of the explicit, implicit and threshold masks specified in the xM argument. The XYZ matrix contains the voxel coordinates of all voxels in the analysis mask. The mask image is included for reference, but is not explicitly used by the results section.

beta_????.{img,hdr} - parameter images These are 32-bit (float32) images of the parameter estimates. The image files are numbered according to the corresponding column of the design matrix. Voxels outside the analysis mask (mask.img) are given value NaN.

ResMS.{img,hdr} - estimated residual variance image This is a 64-bit (float64) image of the residual variance estimate. Voxels outside the analysis mask are given value NaN.

RPV.{img,hdr} - estimated resels per voxel image This is a 64-bit (float64) image of the RESELs per voxel estimate. Voxels outside the analysis mask are given value 0. These images reflect the nonstationary aspects the spatial autocorrelations.

ResI_????.{img,hdr} - standardised residual (temporary) images These are 64-bit (float64) images of standardised residuals. At most maxres images will be saved and used by spm_est_smoothness, after which they will be deleted.

References:

Christensen R (1996) Plane Answers to Complex Questions Springer Verlag

Friston KJ, Holmes AP, Worsley KJ, Poline JB, Frith CD, Frackowiak RSJ (1995) ''Statistical Parametric Maps in Functional Imaging:

A General Linear Approach'' Human Brain Mapping 2:189-210

Worsley KJ, Friston KJ (1995) ''Analysis of fMRI Time-Series Revisited - Again''

NeuroImage 2:173-181

Image_thresholding.**clusterSizeMask**(*sizeThresh*, *height_mask*)

Usage

```
function [mask,numClusters,XYZ] = clusterSizeMask(sizeThresh,height_mask)
```

Image_thresholding.**threshold_imgs**(*dd*, *u*, *varargin*)

Usage

```
function [P2,P,sigmat,sigmatneg] = threshold_imgs(dd,u,[k],['pos' 'neg' 'both'])
```

Inputs

dd: list of filenames (str matrix)

u: height threshold for images

k: extent threshold for contiguous voxels

[str]:

'pos' 'neg' or 'both', to values above, below,

- and + threshold

Output - threshold t - generic function

also do: plot rob vs ols benefit by tissue class and ols-irls average

Examples

```
[P2,P,s,sn] = threshold_imgs(p([5 8],:),tinv(1-.001,10),0,'both');
compare_filtered_t([],P2(1,:),P2(2,:))
[p2,p1] = threshold_imgs('irls-ols_z_0001.img',norminv(.9),0,'both');

% compare_filtered_t([],P2(1,:),P2(2,:),p2)
h = image_scatterplot(str2mat(P,p1),'avgvs3');
xlabel('Average OLS and Robust t-value'), ylabel('Z-score of Robust - OLS difference')

s = str2mat('rob_tmap_0002.img','rob_tmap_0003.img');
P = threshold_imgs(s,tinv(1-.05,36),[],'pos');P = threshold_imgs(s,tinv(1-.05,36),[],'neg');
```

3.8 Index_image_manip_tools

Index_image_manip_tools.**flip_endianness**(*imgs*)

Flips fMRI images (swap R/L)

Usage

```
[Vimgs] = flip_endianness(imgs)
```

Inputs

imgs: Give this a list of fMRI images (paths, char str)

Outputs

Vimgs: Your images, flipped

Examples

:: load EXPT %study specific spm design file imgs = EXPT.SNPM.P{1}; [Vimgs] = flip_endianness(imgs)

References N/A

See also

- spm_vol

Index_image_manip_tools.**iimg_check_volinfo**(*maskInfo*, *imgInfo*)

Checks a series of image .mat files and dims against a reference (maskInfo)

Usage

```
anybad = iimg_check_volinfo(maskInfo,imgInfo)
```

Inputs

maskInfo and volInfo : are spm-style volume info structures

see spm_vol.m

Index_image_manip_tools.**iimg_cluster_extent**(*dat*, *volInfo*, *k*)

Apply a cluster size threshold to a series of index image data vectors (each img is one column) given volInfo (must be created with iimg_read_img, with extended output)

Usage

```
[dat,nvox] = iimg_cluster_extent(dat,volInfo,k)
```

Inputs

dat: may be an indexed image of all image values or only those in mask defined by volInfo

Index_image_manip_tools.**iimg_cluster_index**(*dat*, *xyz*, *k*)

Get voxel cluster indices and sizes :Usage:

```
[clindx,nvox] = iimg_cluster_index(dat,xyz,[k])
```

Inputs

xyz: is 3 x n list of voxel coords, volInfo.xyzlist'

dat: is index vector of image values

Returns: cluster index and cluster sizes for non-zero, non-nan voxels

initialize

Index_image_manip_tools.**iimg_cluster_intersect**(*dat1*, *dat2*, *volInfo*)

Prunes two image data vectors (dat1 and dat2) assumed to contain suprathreshold contiguous clusters ("blobs") by saving only those blobs that have one or more significant (nonzero) elements in both images

Usage

```
[cl1,cl2,dat1,dat2] = iimg_cluster_intersect(dat1,dat2,volInfo)
```

Inputs

volInfo.xyzlist: must contain xyz coordinates corresponding to dat1 and dat2

dat1 and dat2: can be either image-length or in-mask length vectorized images

Try iimg_threshold to create dat vectors from Analyze images (e.g., statistic images)

The outputs cl1 and cl2 are the overlap (intersection) clusters, in the space of dat1 (cl1) and dat2 (cl2). The outputs dat1 and dat2 are the 'pruned' intersection data vectors.

prune dat1 with sig values in dat2

Index_image_manip_tools.**iimg_cluster_prune**(*dat*, *datsig*, *volInfo*)

Prunes an image data vector (dat) assumed to contain suprathreshold contiguous clusters ("blobs") by saving only those blobs that have one or more significant (nonzero) elements in another image, datsig

Usage

```
[dat_out,clindx,keepit] = iimg_cluster_prune(dat,datsig,volInfo)
```

One intended use is to define an FWE-corrected significance map in datsig, and report blobs at some lower threshold (in dat) that have at least one corrected voxel.

Try iimg_threshold to create dat vectors from Analyze images (e.g., statistic images)

check data type and reduce to in-mask only if necessary

Index_image_manip_tools.**iimg_clusters2indx**(*cl*, *volInfo*)

Take a clusters structure and turn it into an indexed image of dims volInfo.dim

Usage

```
[imgvec,maskvec] = iimg_clusters2indx(cl,volInfo)
[imgvec,maskvec] = iimg_clusters2indx(cl,my_image_name)
```

Outputs

imgvec: vector of all image voxels

maskvec: vector of in-mask voxels

Uses: volInfo.nvox, .wh_inmask, .dim convert image to volinfo struct, if necessary

Index_image_manip_tools.**iimg_indx2contiguousxyz**(*dat*, *volInfo*, *remove_mean_flag*)

Take in index image data vector (in-mask values only) and a volume info structure with xyzlist and return a cl structure whose XYZ values list contiguous sets of voxels ("blobs")

Usage

```
cl = iimg_indx2contiguousxyz(dat,volInfo,[remove_mean_flag])
```

If a 3rd arg is entered, means of each blob are subtracted This is to facilitate randomizing blob centers in meta_stochastic_activation_blobs.m

Index_image_manip_tools.**iimg_intersection**(*varargin*)

Make a mask of the intersection of n image files (pos or neg values) If 'name' followed by an image file name is entered, writes output to that image

Usage

```
[int_dat,mask_vol,outname] = iimg_intersection(name1, name2, etc.)
[int_dat,mask_vol,outname] = iimg_intersection(dat1, dat2, etc.)
```

If special string 'posneg' is entered, separates first two images into combinations of pos/neg values in each image. The order returned in columns of int_dat is pospos, posneg, negpos, and negneg

fastest if one output requested.

Index_image_manip_tools.**iimg_make_sure_indx**(*inputarg*)

Usage

```
dat = iimg_make_sure_indx(inputarg)
```

Make sure input is an index list, and convert if not

Index_image_manip_tools.**iimg_mask**(*maskindx*, *dat*, *varargin*)

Masks index image vector or filenames in dat With img vector or filenames in maskindx * Checks dimensions to make sure origins and vox. sizes are same

Usage

```
[masked_indx, volInfo, masked_vol] = iimg_mask(maskindx,dat,[volInfo], [outname])
```

Fastest way to mask an image:

```
[masked_indx_data, volInfo] = iimg_mask(maskindx,dat)

% or

[masked_indx, xyz, masked_vol] = iimg_mask(mask filename,image filenames)
```

even slower: reconstruct a 3-D volume and return in masked_vol

Index_image_manip_tools.**iimg_multi_threshold**(*inname*, *varargin*)

Usage

```
[cl, dat, cl_extent, dat_extent] = iimg_multi_threshold(inname, varargin)
```

Multi-threshold application and orthview display of blobs

Inputs

inname: Either the name of an image file or a data vector

if data, enter volInfo structure separately as below.

Command strings

'prune': consider only contiguous blobs for which at least 1 voxel meets the most stringent threshold

'pruneseed': followed by a vectorized thresholded activation image

Only those blobs overlapping with at least one 'seed' data voxel are saved

Also: the first color (highest threshold) in the output images is assigned to the seed

'add': add new blobs to existing orthviews

'p': if image is a p-value image (and thresholds are p-values)

'thresh': followed by a vector of n thresholds, most to least stringent

'size': followed by a vector of size thresholds

**'volInfo': followed by volInfo struct; needed if inname is data rather

than filenames

'overlay': followed by overlay image (anatomical) filename

'transseed': transparent seed

'hideseed': do not show seed regions on plot

'pos': positive results only

'neg': negative results only

'add': add to existing multi-threshold plot

Needs (development): Legend, nice handling of colors, input colors, color maps specified with command words (like 'red')

Examples

```
inname = 'Activation_proportion.img';
[cl, dat] = iimg_multi_threshold(inname, 'prune', 'thresh', [.1 .5 .3], 'size', [1 5 10]);
```

```
    cl2 = iimg_multi_threshold(Pimg(1, :), 'thresh', [.001 .01 .05], 'size', [3 5 10], 'p');
    cl2 = iimg_multi_threshold(Pimg(1, :), 'thresh', [.001 .01 .05], 'size', [3 3 3], 'p', 'prune');
```

from act + corr results (see robust_results_act_plus_corr) First prepare 'seed' regions that overlap with correlated regions, then use multi_threshold to see full extent of clusters

```
    [dattype, seeddat] = iimg_check_indx(res.act.overlapdat, res.volInfo, 'full');
    [cl, dat] = iimg_multi_threshold('rob_p_0001.img', 'p', 'thresh', [.001 .01 .05], 'size', [1 1 1
```

Display an F-map from robust regression on a customized mean anatomical, with pruning.

```
    cl = iimg_multi_threshold('rob_pmap_full.img', 'thresh', [.001 .01 .05], 'size', [1 1 1], 'p', '
```

Display regions at 3 thresholds with an input data 'seed' vector

```
    [cl, dat] = iimg_multi_threshold(pvals, 'p', 'thresh', [.001 .01 .05], 'size', [1 1 1], 'prunese
```

As above, but ONLY POSITIVE RESULTS

```
    [cl, datout] = iimg_multi_threshold(pimg1, 'thresh', [.001 .01 .05], 'size', [1 1 1], 'p', 'prun
```

Complete example for showing positive and negative blobs:

```
    [clpos] = iimg_multi_threshold('slope_p.img', 'p', 'thresh', [.005 .01 .05], 'size', [1 1 1], 'p
    [clneg] = iimg_multi_threshold('slope_p.img', 'p', 'thresh', [.005 .01 .05], 'size', [1 1 1], 'p
```

Index_image_manip_tools.**iimg_princomp**(*maskname*, *image_names*)

Usage

```
    [volInfo,clusters] = iimg_princomp(maskname,image_names)
```

Index_image_manip_tools.**iimg_princomp_display**(*volInfo*, *k*, *overlay*, *dofigs*)

Usage

```
    cl = iimg_princomp_display(volInfo,k,overlay,dofigs)
```

Show output from iimg_princomp

Example

```
    cl2 = iimg_princomp_display(volInfo2,1,EXPT.overlay);
```

Index_image_manip_tools.**iimg_read_img**(*inputimgs*, *extended_output_flag*, *reading_data*, *first_vol*)

Usage

```
    [volInfo, dat] = iimg_read_img(inputimgs, [extended_output_flag (1|2)], [reading_data (0|1)], [f
```

- reads 3-D or 4-D img/nii files, with extended volInfo output
- flag to read first volume only
- can read iimg_data vectorized form as well as file names (just passes dat to output)
- uses SPM

volInfo fields:

- all fields returned by spm_vol, plus:
- .nvox - number of voxels in 3d image

- .image_indx - logical index (e.g., [0 1 1 0 1 ...]) of all nonzero voxels
- .wh_inmask (extended_output_flag > 0) - linear index (e.g., [2 3 5 ...]) - same as image_indx, but as a result of find
- .n_inmask (extended_output_flag > 0) - length of .wh_inmask
- .xyzlist (extended_output_flag > 0) - voxel coords of .wh_inmask voxels
- .cluster (extended_output_flag > 1) - cluster structure of in-mask voxels

Extended output flag values: 1: Add xyz coord list and linear index of which voxels nonzero & non-nan in mask 2: Add clusters from spm_clusters

Enforces that dat is a vector of data values, regardless of input.

Examples

```
imname = 'rob_tmap_0001_filt_t_3-05_k5_pos.img';
volInfo = iimg_read_img(imname);
```

WARNING: The behavior of this function is SPM version dependent. volInfo only contains the spm_vol structure of the FIRST image!!! Data in SPM5/8 should be 4-D, but in SPM2 should be 3-D

1. **Get volume info from first volume of a 3-D or 4-D image** volInfo structure has necessary info for converting to/from "vectorized" format dat returns 4-D data from entire image (all volumes)

```
iimg_name = 'test_run1_pca.img';
```

[maskInfo, dat] = iimg_read_img(img_name, 2);

2. Get data in "vectorized" image format for each volume in the image. Works for a list of images too. data is in-mask voxels x volumes this can be useful when you want to return whole-brain data for many images, but in a search volume only (i.e., no extra-brain voxels)

```
idata = iimg_get_data(maskInfo, img_name);
data = data';  % make sure columns are volumes
```

% 3) Write out a 4-D image with the same data, called test_run1_pca2.img

```
ivoldat3D = iimg_reconstruct_vols(data, maskInfo, 'outname',
                                  'test_run1_pca2.img');
```

Index_image_manip_tools.**iimg_read_vols**(*V*, *mask*)

Drop-in replacement for spm_read_vols Primary difference is that if the images have the same voxel size but are NOT resliced (e.g., have differing affine matrices), this will handle reading each individually and putting them together.

Index_image_manip_tools.**iimg_reconstruct_3dvol**(*dat*, *volInfo*, *varargin*)

Reconstruct a 3-D volume from a dat index list

Usage

```
voldata = iimg_reconstruct_3dvol(dat, volInfo, [optional args])
```

Optional Inputs if entered, will write .img file to disk

'outname': followed by output name

'descrip': followed by description for img file

'slice': followed by slice number of single-slice data in image

THIS FUNCTION IS DEPRECATED. USE IIMG_RECONSTRUCT_VOLS.M' WHICH CAN DEAL WITH 4-D VOLUMES AS WELL

Index_image_manip_tools.**iimg_reconstruct_vols**(*dat*, *volInfo*, *varargin*)

Reconstruct a 3-D or 4-D volume from a "dat" matrix of vectorized images

Usage

```
voldata = iimg_reconstruct_vols(dat, volInfo, [optional args])
```

Optional Inputs If entered, will write .img file to disk

'outname': followed by output name

'descrip': followed by description for img file

'slice': followed by slice number of single-slice data in image

Example of image reading

1. **Get volume info from first volume of a 3-D or 4-D image** volInfo structure has necessary info for converting to/from "vectorized" format dat returns 4-D data from entire image (all volumes)

```
img_name = 'test_run1_pca.img';
[maskInfo, dat] = iimg_read_img(img_name, 2);
```

2. Get data in "vectorized" image format for each volume in the image. Works for a list of images too. data is in-mask voxels x volumes this can be useful when you want to return whole-brain data for many images, but in a search volume only (i.e., no extra-brain voxels)

```
data = iimg_get_data(maskInfo, img_name);
```

data = data'; % make sure columns are volumes

3.Write out a 4-D image with the same data, called test_run1_pca2.img

```
voldat3D = iimg_reconstruct_vols(data, maskInfo, 'outname', 'test_run1_pca2.img');
```

Index_image_manip_tools.**iimg_reslice**(*matchto*, *reslicethis*, *varargin*)

Usage

```
out = iimg_reslice(matchto, reslicethis, varargin)

out = iimg_reslice(matchto, reslicethis, 'write', 'outname', 'myimg.img')
```

default flags interp = 2 is trilinear

Index_image_manip_tools.**iimg_smooth_3d**(*w*, *volInfo*, *sfwhm*, *varargin*)

Smooth 3-D images stored in columns with FWHM in mm of sfwhm

Usage

```
function w = smooth_3d(w, volInfo, sfwhm, [badvox])
```

NOTE: SMOOTHING KERNEL MAY BE IN VOX, AS VOL INFO IS NOT PASSED IN

Inputs

w: Take v x n matrix w, and smooth columns (images), returning v x n matrix again

volInfo: is volume info structure from iimg_read_img.m

Optional If v is smaller than the original image because some voxels

were removed, enter logical vector badvox, and the missing voxels will be filled with zeros.

NOTE: 4-D version is horribly slow and memory intensive:

Example

```
wvol = iimg_reconstruct_vols(w', fmri_data_obj.mask.volInfo);
```

Index_image_manip_tools.**iimg_sphere_timeseries**(*images*, *XYZmmCenter*, *radius*)

Usage

```
function [data, XYZvoxSphere, XYZmmSphere] = iimg_sphere_timeseries(images, XYZmm, radius)
```

Inputs

images: list of image files

XYZmm: [3 x n] array of mm coords

radiu: radius in mm of sphere to generate

Outputs

data: voxel data

XYZvoxSphere: voxel

Index_image_manip_tools.**iimg_stouffer**(*Pimg*, *maskname*, *thr*, *clsize*, *outname*)

This function performs combination of p-values across images using the Stouffer method (see refs below).

Usage

```
function cl = iimg_stouffer(Pimg,maskname,thr,clsize,outname)
```

Inputs

Pimg: string mtx of p-value image names

k: num voxels/num comparisons if empty, uses # of non-zero, non-NaN values in images

empty Pimg prompts for graphic selection of filenames empty outname prompts for entry of output img file name

Outputs

cl: clusters, cl{1} is stouffer, cl{2} is image 1, cl{3} is image 2

Described in

Lazar, N. A., Luna, B., Sweeney, J. A., & Eddy, W. F. (2002). Combining brains: a survey of methods for statistical pooling of information. Neuroimage, 16(2), 538-550.

Stouffer, S. A., Suchman, E. A., DeVinney, L. C., Star, S. A., and Williams, R. M. 1949. The American Soldier: Vol. I. Adjustment During Army Life. Princeton University Press, Princeton.

...

Index_image_manip_tools.**iimg_threshold**(*image_names*, *varargin*)

Thresholds images to be at least thresh(1) and at most thresh(2)

Usage

```
function [dat, volInfo, cl] = iimg_threshold(image_names, varargin)
```

Input types for image names

1. String matrix of filenames
2. 4-D array of 3-D data volumes
3. voxels x images 2-D index array
4. image_vector object

Outputs

imdat: index vector of thresholded data

volInfo: structure of info about volume

cl: optional (slower) clusters structure from dat

Command strings

'imgtype':

followed by 't', 'p', or 'data' (default)

specify type of values in image

'threshtype': followed by 't' or 'p'

'df': followed by degrees of freedom, for p- and t-image threshold calc

'k': followed by extent threshold in voxels (slower)

'abs': absolute value must be > threshold. Use to get both pos. and neg. results in two-tailed test.

'intersect: Create intersection of images; uses abs, so + and - values count as yesses for intersection

'contrast': followed by contrasts across images

'volInfo: followed by volInfo structure. Necessary for extended output

Examples

```
% Threshold an image set (p) to be at least zero
image_names =
/Users/tor/Documents/Tor_Documents/CurrentExperiments/Lab/Pre-appraisal/Results/rscalped_avg152T
/Users/tor/Documents/Tor_Documents/CurrentExperiments/Lab/Pre-appraisal/Results/t_intercept.img

[dat, volInfo] = iimg_threshold(image_names, 'thr', 0);

% Do the same, but take the intersection and write an output image
[dat, volInfo] = iimg_threshold(image_names, 'thr', 3, 'outnames', 'intersct', 'masked_t.img');


% Threshold a t-image based on a p-value and a df
image_names = '/Users/tor/Documents/Tor_Documents/CurrentExperiments/Lab/Pre-appraisal/Results/t
[dat, volInfo] = iimg_threshold(image_names, 'thr', .005, 'imgtype', 't', 'threshtype', 'p', 'df

cl = mask2clusters('masked_t.img');
cluster_orthviews(cl);
```

```
    % The same, but threshold based on absolute value (+ and - values)
    [dat, volInfo] = iimg_threshold(image_names, 'thr', .005, 'abs', 'imgtype', 't', 'threshtype', '

    % Threshold a p-value image directly
    [dat, volInfo] = iimg_threshold('X-M-Y_pvals.img', 'thr', [0 .05], 'outnames', 'X-M-Y_pvals_thre
    [dat, volInfo, cl] = iimg_threshold(inname, 'thr', [0 .05], 'outnames', outname);

    % Threshold a p-value image, with cluster sizes
    [dat, volInfo, cl] = iimg_threshold(inname, 'thr', [0 .05], 'k', 10);

    % Threshold and display a t-image using FDR, getting both positive and negative results
    [dat, volInfo, cl] = iimg_threshold('contrast_t.img', 'imgtype', 't', 'df', 37, 'thr', .2, 'thre
    cluster_orthviews(cl);
    spm_orthviews_hotcool_colormap(cat(2,cl.Z), 1.52);
```

Index_image_manip_tools.**iimg_weighted_ttest**(*image_names*, *varargin*)

fast weighted test

Examples

```
    iimg_weighted_ttest(image_names,'mask',maskname)
```

Index_image_manip_tools.**iimg_write_images**(*dat*, *volInfo*, *outnames*)

Write a series of Analyze images given inputs.

Usage

```
    iimg_write_images(dat,volInfo,outnames)
```

Inputs

dat: voxels x images matrix of image data in index format

volInfo: spm-style info structure; see iimg_read_img

outnames: a string matrix (or cell array) of output filenames, or the name of a 4-D file to create

Index_image_manip_tools.**iimg_xyz2indx**(*xyz*, *xyztype*, *V*)

Usage

```
    [indx, wh] = iimg_xyz2indx(xyz, input type:['mm' or 'vox'], [V: needed if 'mm'])
```

Index_image_manip_tools.**iimg_xyz2spheres**(*xyz*, *mask_xyzlist*, *r*)

Usage

```
    indx = iimg_xyz2spheres(xyz, mask_xyzlist, r)
```

Inputs

mask_xyzlist: is a list of voxel coords of all in-mask voxels

xyz: is a list of voxels to be convolved with spheres

r: is sphere radius (voxels)

more generally, finds mask_xyzlist entries that are within r units of xyz i.e., to find database points within r mm of a cluster xyz list: indx = iimg_xyz2spheres(clusterxyz,databasexyz,r)

indx is length mask_xyzlist and contains all in-mask, in-sphere voxels for all xyz coordinates.

3.9 Misc_utilities

Misc_utilities.**CERTreader**(*fname*)

Given an output file from CERT, returns a struct with the data

Input

fname: filename

Output struct with the data

Misc_utilities.**append**(*a*, *b*, *field*)

Appends second parameter to the end of the first. If field is set, writes into the field instead of directly into the variable.

Usage

```
a = append(a, data, [field])
```

This function solely exists because Matlab has no ability to create empty objects (e.g. with one dimension being 0) and thus, trying to append to an empty array results in type cast errors, since empty matrices are double by default.

Misc_utilities.**blank_struct**(*A*, *blank_val*)

Utility function to create a blank structure, using an existing one as a template.

Inputs

A: template structure

blank_val: all fields will be set to this value, e.g. [] or ‘’

Output

C: structure containing fields of A , with values set to empty array

Only works for single-element non-nested structures, for now

Misc_utilities.**checkMatlabVersion**(*datestr*)

Usage

```
ok = checkMatlabVersion(datestr)
```

check Matlab version against release date of version needed (or recommended)

Inputs

datestr: format is one of the forms taken by datenum.

Example check for Matlab release date on or after Aug. 3, 2006

```
ok = checkMatlabVersion('8/3/2006')
```

Misc_utilities.**circle**(*center*, *radius*, *varargin*)

draws a circle

Usage

```
[h, fillh] = circle(center, radius, ['fill'], [color string or numbers])
```

‘fill’ requires the geom2d toolbox, by David Legland

Misc_utilities.**combine_structs**(*A*, *B*, *prefix*)

Utility function to combine the fields of two structures.

Inputs

A, B: structs to be combined

prefix: string applied to fields of structure B

Output

C: structure containing fields of both A and B

Only works for single-element non-nested structures, for now

Misc_utilities.**condf2indic**(*X*)

Create N x k indicator matrix of ones/zeros from N x 1 list of numeric values (X)

Usage

```
[indic, xlevels] = condf2indic(X)
```

Outputs

indic: is returned as single precision type, because it can then be used as a design matrix in a GLM

xlevels: are the values of X corresponding to columns of indic

Misc_utilities.**depfun_aggregate**(*funname*, *excludes*, *save_dir*)

depfun_aggregate is for packaging all the files needed for a function in one spot

Usage

```
dependencies = depfun_aggregate(funname, excludes, save_dir)
```

Inputs

funname: function to analyze

excludes: cellstrs to exclude from the matches

save_dir: directory to copy files to - defaults to 'tmp_files'

Example

```
dependencies = depfun_aggregate('whole_brain_fir', {'spm2', 'spm5'}, 'fir_files')
```

Misc_utilities.**distance**(*u*, *v*)

Euclidean distance between u and v

Usage

```
d = distance(u, v)
```

Inputs

u and v: should be column vectors or matrices in k-dim space, where k is the number of columns of u and v.

if u is a single row, replicates u to size(v,1)

Misc_utilities.**distance_euclid**(*u*, *v*)

Euclidean distance between u and v

Usage

```
d = distance_euclid(u, v)
```

Inputs

u and v: should be column vectors or matrices in k-dim space, where k is the number of columns of u and v.

if u is a single row, replicates u to size(v,1)

Misc_utilities.**erase_and_display**(*old_str*, *new_str*)
Backspaces the number of chars of the length of old_str, then prints new_str

Misc_utilities.**erase_string**(*str*)
Backspaces the number of chars of the length of str

Misc_utilities.**explode**(*string*, *delimiters*)
EXPLODE Splits string into pieces.

EXPLODE(STRING,DELIMITERS) returns a cell array with the pieces of STRING found between any of the characters in DELIMITERS.

Usage

```
[SPLIT,NUMPIECES] = EXPLODE(STRING,DELIMITERS)
```

also returns the number of pieces found in STRING.

Inputs

STRING: the string to split (string)

DELIMITERS: the delimiter characters (string)

Outputs

SPLIT: the split string (cell array), each cell is a piece

NUMPIECES: the number of pieces found (integer)

Examples

```
STRING = 'ab_c,d,e fgh'
DELIMITERS = '_,'
[SPLIT,NUMPIECES] = EXPLODE(STRING,DELIMITERS)
SPLIT = 'ab'    'c'    'd'    'e fgh'
NUMPIECES = 4
```

See also

IMPLODE, STRTOK

Misc_utilities.**fast_conv_fft**(*hrf*, *x*, *varargin*)

Usage

```
y = fast_conv_fft(hrf,x)
```

Much faster than conv or using matrix multiplication.

Inputs

hrf: should be length of x

Examples

```
    y = fast_conv_fft(hrf,x);
    y = fast_conv_fft(hrf,x3,'deconv');
```

Misc_utilities.**getRandom**(*stimList*)

Randomizes rows of a matrix, preserving dependencies between columns

Usage

```
        stimList = getRandom(stimList)
```

Input a col. vector or matrix of stimulus conditions,

e.g. [1 1 1 1 2 2 2 2 3 3 4 4]'

output: a randomized permutation of this vector or matrix all columns are resorted with the same order

Misc_utilities.**get_first_help_lines**(*functionname*, *maxlines*)

Returns the first n lines of the help for a function in a cell array

Usage

```
    helptext = get_first_help_lines(functionname, [maxlines])

    helptext = get_first_help_lines(functionname, maxlines)
```

Examples

```
    helptext = get_first_help_lines('fmri_data.apply_mask', 5);
    char(helptext{:})
```

Misc_utilities.**implode**(*pieces*, *delimiter*)

IMPLODE Joins strings with delimiter in between.

IMPLODE(PIECES,DELIMITER) returns a string containing all the strings in PIECES joined with the DELIMITER string in between.

Inputs

PIECES: the pieces of string to join (cell array), each cell is a piece

DELIMITER: the delimiter string to put between the pieces (string)

Output

STRING: all the pieces joined with the delimiter in between (string)

Examples

```
    PIECES = {'ab','c','d','e fgh'}
    DELIMITER = '->'
    STRING = IMPLODE(PIECES,DELIMITER)
    STRING = ab->c->d->e fgh
```

See also

EXPLODE, STRCAT

Misc_utilities.**naninsert**(*nanvec*, *y*)

Insert NaNs back into data series from which they were removed

Usage

```
yout = naninsert(nanvec, y)
```

nanvec is indicator for NaNs, of size size(yout). y is data.

If y is a matrix, length(nanvec) should equal number of rows in y. NaNs are inserted for true values in nanvec in all columns of y.

See also

nanremove.m

Misc_utilities.**nanremove**(*varargin*)

Usage

```
varargout = nanremove(varargin)
[nanvec, x1, x2, etc.] = nanremove(x1,x2,etc...)
```

removes cases that are NaN in any column of any variable x1...xn

Misc_utilities.**oneinsert**(*removed_voxels*, *data*)

Insert ones (for p values) back into data series (i.e., obj.p) from which they were removed.

Usage

```
yout = oneinsert(removed_voxels, data)
```

removed_voxels is indicator for the locations of removed voxels

This function is a modified version of naninsert.m, but different.

See also

naninsert.m, nanremove.m, zeroinsert.m

Misc_utilities.**orthviews_multiple_objs**(*imgs*)

plot multiple image objects on one orthviews

Input cell array of image_vectors or statistic_images

Misc_utilities.**pad**(*x*, *L*)

Usage

```
x = pad(x,L)
```

pads x with zeros of length L

or, if L is a vector, to longest of two (NOT DONE)

COLUMN VECTORS

Misc_utilities.**padwithnan**(*A*, *B*, *dim*)

returns the two input arrays, with the smaller padded to the size of the larger in the particular dimension(s) with NaNs

Usage

```
[Anew Bnew] = padwithnan(A, B, dim)
```

Misc_utilities.**parse_char_to_cell**(*invar*, *sepval*)

take a row of characters and separate into cells, breaking at either spaces or tabs

Usage

Usage

```
outcell = parse_char_to_cell(invar,sepval)
```

Examples

```
% copy from Excel as row, then parse:
disorder = ['SAD    PTSD    PTSD    PTSD    PTSD    PTSD    SP      SAD     PTSD    PTSD    PTSD
disorder = parse_char_to_cell(disorder, 'tab');
```

Misc_utilities.**parse_edat_txt**(*fname*)

Reads EPrime .txt output (equivalent to EDAT) directly into matlab cell arrays/structures

Inputs:

fname: name of file to parse

Outputs

edat_struct: Structure containing the following fields:

header: 1-element structure whose fields are header items from EDAT

run: 1-element structure whose fields are run-specific items from EDAT

trials: n-element structure whose fields are trial-specific items from EDAT where n is the number of trials

edat_cells: Structure containing the following fields:

header_cols: 1-row cell array whose fields are header column names from EDAT

run_cols: 1-row cell array whose cells are run-specific column names from EDAT

trials_cols: 1-row cell array whose cells are trial-specific column names from EDAT

header: 1-row cell array whose cells are header items from EDAT

run: 1-row cell array whose cells are run-specific items from EDAT

trials: n-row cell array whose cells are trial-specific items from EDAT where n is the number of trials

Two passes are necessary:

1. find all fields used in the file
2. actually read the data

Misc_utilities.**print_matrix**(*x*, *varargin*)

prints matrix values as tab delimited, 2 decimal places

Usage

```
print_matrix(x,[col names cell array], [row names cell], [format string])
```

Examples

```
t = [1 2; 3 4; 5 6];
print_matrix(t,{'col1' 'col2'},{'row1' 'row2' 'row3');

print_matrix(rand(5), [], [], '%3.2f');
print_matrix(rand(5), {'A' 'B' 'C' 'D' 'E'}, {'A' 'B' 'C' 'D' 'E'}, '%3.2f');
print_matrix(rand(5), {'A' 'B' 'C' 'D' 'E'}, {'A' 'B' 'C' 'D' 'E'}, '%d');
```

Misc_utilities.**progressbar**(*meth*, *val*)

Create a progress bar window with tag 'progressbar' that can be updated

Usage

```
progressbar(meth,val)
```

Inputs

meth: can be 'init' or 'update'

x-axis limits are 0 - 100, so val should be % complete for best results

Outputs the f and ax handles are not really needed as 'update' finds the axis with 'progressbar' tag.

Example

```
progressbar('update',100*i./nvars);
```

Misc_utilities.**read_edat_output_2008**(*fname*, *varargin*)

Function that creates a structure DATA containing columns of the edat file output (saved in text tab delimited "excel" format)

Usage

```
DATA = read_edat_output_2008(fname, varargin)
```

For this code to work on a Mac, you must: 1) export .edat2 file as an Excel file, then 2) open this file in Excel on a Mac and save as a .csv, 3) read that .csv file

Examples

```
fname = 'myfile.txt';
DATA = read_edat_output_2008(fname)
```

Defaults

These are the default formats this function expects: tab delimited, 1 header row, then row of column names, then data

You can override some of them by using the following – E.g., for zero header rows and comma delimited data:

```
DATA = read_edat_output_2008(fname, 'nheaderrows', 0, 'mydelimiter', ',')
```

You can force the number of columns to be a certain value by doing the following:

```
DATA = read_edat_output_2008(fname, 'nheaderrows', 1, 'numc', 103);
```

This could be useful if your last row contains empty cells at the end, which will mess up the automatic calculation of number of columns.

Misc_utilities.**robustcsvread**(*filename*, *varargin*)

ROBUSTCSVREAD reads in CSV files with different number of columns on different lines

This returns a struct, with one field per column of the csv file. Each field is a cell array whose length = rows in the csv file. Column names are assumed to be in the first row. If column names are invalid struct field names, edits them by replacing funky characters with an underscore, or if first char is a number, I prepend **aa_** to the field name.

Inputs

varargin: cols: how many cols to read in, by defaults reads them all

rows_to_skip: how many rows to skip

delim: cell delimiter

missing: followed by cell array, first cell is val for missing, second cell is what to replace with

Misc_utilities.**scn_get_datetime**(*varargin*)

Usage

```
str = scn_get_datetime
```

pass in 'ymd' to get the string in yyyy_mm_dd-HH_MM format, so that alphanumeric order will correspond to chronological order

Returns a string with the date and time Useful for annotating data and output

Misc_utilities.**scn_mat_conform**(*in*)

Usage

```
function in = scn_mat_conform(in)
```

sets flipping to 0 (no flip) in SPM2 and adjusts mat file accordingly input in spm-style mat file or struct with .mat or .M fields

Misc_utilities.**search_struct_fields**(*search_struct*, *fieldname*, *fieldpath*)
Returns a list of all paths inside structure search_struct that match the fieldname (or start with it)

Usage

```
found_paths = search_struct_fields(search_struct, fieldname)
```

Examples

```
foo = [];
foo.foo = [];
foo.foo.foo = [];
search_struct_fields(foo, 'foo')
ans =
   'foo.foo'
   'foo.foo.foo'

% or

search_struct_fields(SPM, 'x')
ans =
   'SPM.xX'
   'SPM.xM'
   'SPM.xsDes'
   'SPM.xX.xVi'
   'SPM.xX.xKXs'
   'SPM.xM.xs'
```

Misc_utilities.**strip_path_dirs**(*patterns*)
Removes directories from the Matlab path, based on the rgex patterns passed in.

Usage

```
function strip_path_dirs(regexes)
```

Misc_utilities.**strip_svn_dirs**()
Removes the .svn dirs from the path

Misc_utilities.**strrep_recurse**(*old_var*, *old_string*, *new_string*, *depth*)
Recursively traverses depth-first through an entire variable, replacing old_string with new_string everywhere it goes

Usage

```
new_var = strrep_recurse(old_var, old_string, new_string)
```

Misc_utilities.**struct_strrep**(*old_struct*, *old_string*, *new_string*, *depth*)
Traverses depth-first through an entire structure, replacing old_string with new_string everywhere it goes

Usage

```
new_struct = struct_strrep(old_struct, old_string, new_string)
```

Misc_utilities.**tor_ga**(*gensize*, *numgen*, *inputs*, *ofun*, *varargin*)

Usage

```
[best_params,fit,beff,in,isconverged] = tor_ga(gensize,numgen,inputs,ofun,[optional in any order
```

Inputs a cell array describing the inputs to the optimization function (parameters to be optimized).

Each cell of inputs is a p x q matrix of parameters. p and q are arbitrary, as each organism is described by a p x q matrix...but the objective function must be able to handle inputs in the format you provide. Internally, a set of 'organisms' is created that is params x params x organisms (3-D). This matrix is subject to crossover across orgs. separately for each cell. If inputs is a p x q matrix, it will be placed in a single cell.

By default, it is only necessary to enter a single set of params for an example organism. The range of those input values is used to generate random starting values for each organism.

There can be more than one set of parameters that are combined in some way by ofun to produce a fitness value. if there is more than one set of input parameters, inputs should be entered as a cell array, one cell per input. inputs should be in ORDER of inputs entered to ofun!

RECOMMENDED

If you enter each cell of inputs as a 3-D array so that inputs(:,:,1) is the min acceptable value for each param and inputs(:,:,2) is the max acceptable value, then the ga will create a series of organisms at start that evenly span the range of the multivariate parameter space, with the spacing between values determined by the gensize. This can provide a huge advantage in efficiency for the GA. This is the idea behind the "Sobol sequence," which chooses values that evenly span a multivariate space. With this option, if gensize is sufficiently large and the param space is sufficiently small, then the ga may find the correct solution on the first iteration. However, it is not likely to work well if the num. params >> gensize

Examples

```
start = [-15 -15];
start(:,:,2) = [15 15];
[best_params,fit,beff,in] = tor_ga(324,30,{start},objfun_ga,'genconverge',5);
```

ofun

- the objective function that combines the inputs.

There are two options for passing this in:

1. enter the name of the function as a string. the program creates a handle for the function, and evaluates it using inputs specified in the inputs variable. In this case, pass in fixed inputs after ofun, in the varargin fields fixed inputs optional, fixed inputs that do not change! same structure as inputs.
2. You can also enter ofun as a function handle durectly, with fixed inputs already embedded before running the program. The function should take as input a param list, and return fitness. e.g.,

```
objhan = @(params) my_function_name(params,fixed_inputs1,fixed_inputs1,fixed_inputs1);
objhan = @(wh) prospect_organism(ceil(wh),pop,truep,iter);
```

Pass in objhan as the 'ofun' input argument

genfun

- [optional] input param generation function

A function handle that generates a parameter set for each organism

command strings

- 'noverbose' turn off verbose reporting and plots
- 'genconverge' followed by integer x: converge if no change in last x generations

Examples

Example for fitting indscal model:

```
inputs{1} = X; fixin{1} = sp; fixin{2} = B1; fixin{3} = B2;
tor_ga(30,10,inputs,'indscalf',fixin);

W = rand(size(W)); W(1,:) = [10 10];, W(2,:) = [-10 -10];
inputs{2} = W;
```

Example: Optimize gambles for prospect theory model See prospect_optimize_design.m for definition of population of gambles from which to draw (pop), truep, iter (all fixed inputs)

```
objhan = @(wh) prospect_organism(ceil(wh),pop,truep,iter);
genfun = @() randsample(gindx,ntrials,'true')';
[best_params,fit,beff,in] = tor_ga(5,3,wh,objhan,genfun);
```

Using string inputs to control behavior:

```
[best_params,fit,beff,in] = tor_ga(300,30,{[15; -15]},objfun_ga,'genconverge',5,'noverbose');
```

Misc_utilities.**zeroinsert**(*wasbad*, *y*)

Re-insert removed CASES (rows) and fill with zeros

Usage

```
yout = zeroinsert(wasbad, y)
```

wasbad is indicator for removed cases, of size size(yout). y is data.

if you enter y', inserts VARIABLES (cols). here, pass in v x n matrix, y' to fill empty/removed vars

See nanremove.m and naninsert.m

3.10 Model_building_tools

Model_building_tools.**fmri_spline_basis**(*TR*, *varargin*)

Usage

```
xBF = spline_hrf_basis(TR, optional args)
```

Inputs

imTR: repetition time; sampling resolution of data

'plot': optional: plot basis set

'nbasis': optional: number of knot points

'order': optional: order of spline model (# matched derivatives)

'length': optional: length of window to model, in seconds

Outputs: :Inputs:

xBF.dt: time bin length {seconds}

xBF.name: description of basis functions specified

xBF.length: window length (seconds)

xBF.order: order

xBF.bf: Matrix of basis functions

32 second long spline basis set for fmri model

xBF_hires: Sampled at high resolution, TR * 16

xBF: Sampled at TR

Examples

```
[xBF_hires, xBF] = fmri_spline_basis(TR, varargin)
[xBF_hires, xBF] = fmri_spline_basis(2, 'length', 12, 'nbasis', 3, 'order', 3, 'plot');
```

Model_building_tools.**getPredictors**(*stimList*, *HRF*, *varargin*)

Build predictors and delta functions, given a condition function or delta function and either a convolution matrix or vector.

Usage

```
[model,delta] = getPredictors(stimList, HRF, varargin)
```

IMPORTANT: YOU MUST ADD THE INTERCEPT YOURSELF!

Inputs

stimList: condition function OR delta function (1/0 indicator)

HRF:

1. hemodynamic response function
2. Basis set (columns)
3. or convolution matrix (columns are HRF), defined as: HRF = tril(toeplitz(hrf));

   multiple column vectors for HRF are treated as basis functions!

varargin for downsampleing: 'dsrate': takes every nth element of the design matrix

'dslen': the target number (length) you want to downsample to

Other Optional Inputs

'force_delta': getPredictors tries to determine if the input stimList is a condition function with integers or a delta function with indicators, but this can fail in some cases. Use this to force it to treat as a delta function.

1. a col. vector of stimulus conditions OR a delta function matrix
2. **an HRF vector sampled at the frequency of the stimulus vector, OR** a convolution matrix H (empty for default)
3. Optional: downsampling factor for final design (i.e., TR)
4. Optional: parametric modulator keyword and modulator values

'parametric_singleregressor' [Parametrically modulate onsets by]

modulator values, using single regressor with modulated amplitude Enter a cell array with modulator values for each event type, with a column vector (empty cell for no modulation)

'parametric_standard' [Parametrically modulate onsets by] modulator values, using two regressors per event type - One to model the average response, and one for the mean-centered modulator values

Outputs

1. a n x 2 matrix of regressors (cols) for each condition
2. a n x k delta matrix with onsets

Example TR = 2, 16 samples per second in hi-res delta dhr

```
X = getPredictors(dhr, hrf, 'dsrate', res*TR);
X = getPredictors(dhr, hrf, 'dslen', len/(res*TR));
```

stimList can be condition function e.g., [1 3 2 4 3 2 1]' or delta matrix (n x k), n samples and k conditions, e.g., [1 0 0 0 1 0 1]'

Resampling the default N in matlab resample has built-in antialiasing,

but may not be good for fmri designs! The appropriate downsampling is expected to be res*TR (res is units of samples/s), but we use 0 because the model will depend on the analysis method used, and this is the most veridical approach. With N = 0, every ith sample is used, where i is the downsampling factor you input. Popular choices are 16*TR (for onsets2delta.m), using the SPM default res of 16. Delta is NOT resampled.

Example TR = 2, 16 samples per second in hi-res delta dhr

```
[tmp,d] = downsample_delta(dhr,16*2); X=getPredictors(d,hrf);
```

Model_building_tools.**ideal_deconv6**(*conditions*, *mspec*, *ttype*)

Tests deconvolution matrix directly against idealized data you put in the exact temporal sequence to be deconvolved, in the form of the DX matrix.

Usage

```
[rmsd,msdstd,msdb,biasmean,meanest,min95est,max95est,ALLINFO,hrf,snr,TR] = ideal_deconv6(conditi
```

Inputs

DX: deconvolution matrix

tp: time points estimated for each condition in DX

TR: repetition time of scan

ttype: trial types to test (out of 1:n different conditions in DX) recommended for time saving to use ttype = a single number only

This function is like ideal_deconv5, but tests variability across designs as well.

Model_building_tools.**intercept_model**(*nvols_per_run*, *varargin*)

Build design matrix X for intercepts given vector of session lengths [s1 s2 s3] in images

Usage

```
x = intercept_model(nvols_per_run, [indx of dummy scans in each session])
```

Examples

```
nvols_per_run = [166 166 144 137];
x = intercept_model(nvols_per_run);

x = intercept_model(repmat(166, 1, 5));

Xi = intercept_model(EXPT.FIR.nruns, 1:2);
```

Model_building_tools.**modifiedconv**(*tr*, *condf*, *varargin*)

Usage

```
model = modifiedconv(tr,condf,heighteq [all opt],delayeq,ttopeakeq,uonseteq)
```

Inputs

tr: repetition time (sampling rate) of scanning, in seconds

condf:

condition function an indicator vector of zeros and ones, where ones indicate event onsets

USES nonlinear saturation in height only with a guess as to what the decrease in saturation is as a function of the time since previous stimulation (exponential model, alpha version)

Examples

```
condf = [1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]';
X = modifiedconv(2,condf);
% X is convolved predictor
plot(X)
X2 = conv(condf,spm_hrf(2)./max(spm_hrf(2)));
hold on;
plot(X2,'r');
legend({'Modified' 'Linear'})
```

Please see

Wager, T. D., Hernandez, L., Vasquez, A., Nichols, T., and Noll, D. C. (in press). Accounting for nonlinear BOLD effects in fMRI: Parameter estimates and model for accurate prediction in variable-duration blocked and rapid event-related studies. Neuroimage.

heighteq = []; delayeq = []; peakeq = []; uonseteq = [];

- defaults

Model_building_tools.**onsets2delta**(*ons*, *len*)

Builds high-res delta function, given cell array of onset times

Usage

```
delta = onsets2delta(ons, [length: num rows in original units])
```

Tor Wager, 2 / 24 / 04, update 10 / 12 / 10

Inputs

ons: onsets for each of a series of conditions

One cell per condition per session, e.g., ons{1} = [24 27 29 44]';

Units are arbitrary (e.g., TRs or seconds)

Onsets are assumed to start at time 0 (0 is start of run/session)

e.g., from an SPM.mat fmri design structure, for one session:

```
ons = cell(1, nconds);
[ons{:}] = deal(reportmod_model.Sess(1).U(:).ons)
```

len:

optional: number of rows in original units. Useful for making

a design matrix with the right number of rows after convolution and downsampling

Output

delta: indicator matrix of vectors for each condition with 1/0 for each onset

Type is logical; you may want to do double(delta) before operating

Resolution of high-res delta functions = original units (secs or TRs) * 16

The number of rows is the max onset + 1, times 16

For what to do with output, see: getPredictors : for design-matrix building downsample_canlab : for downsampling to TR/secs

See also ONSETS2FMRIDESIGN and object-oriented fmri_model object

(methods: build, etc.)

Model_building_tools.**onsets2dx**(*onsets*, *TR*, *scansperrun*, *numseconds*)

Usage

```
[DX,delta] = onsets2dx(onsets, TR, scansperrun, numseconds)
```

Inputs

onsets: cell array whose length is num runs * num conditions, e.g., {run 1 ant onsets, run 1 stim onsets, run 2 ant onsets, run 2 stim onsets}

TR: TR in seconds

scansperrun: number of volumes in each run

numseconds: number of seconds after event onsets to generate regressors for [default: 32]

Examples

```
    TR = 2;
    scansperrun = [192 196 196 184 190 192];
    numseconds = 30;
    [DX,delta] = onsets2dx(onsets, TR, scansperrun, numseconds)
    EXPT.FIR.model{subjectnumber} = DX;
```

Model_building_tools.**onsets2fmridesign**(*ons*, *TR*, *varargin*)

Builds design matrix X and delta function, given cell array of onset times in s One cell per condition per session, e.g., ons{1} = [24 27 29 44]';

Usage

```
    [X, delta, delta_hires, hrf] = onsets2fmridesign(onsets, TR, [len], [custom hrf or basis set nam
```

Summary:

- handles multiple conditions
- handles custom HRFs and multiple basis functions
- handles input event durations
- handles two kinds of parametric modulators
- handles variable-duration onsets
- handles nonlinear saturation (see hrf_saturation.m)
- Can build single-trial model
- not yet: variable-duration parametric modulators
- See the code comments for a discussion of absolute scaling of regressors and efficiency.

Inputs

onsets:

- 1st column is onsets for events,
- 2nd column is optional durations for each event
- Enter single condition or cell vector with cells for each condition (each event type).

TR: RT in seconds

Optional Inputs First two are fixed, then keywords:

len: optional length in s for model, or [] to skip if using additional. "len" is usually the number of images multiplied by TR.

HRF name:

string or actual values

1. a string used by spm_get_bf.m
2. a custom HRF, sampled in seconds
3. or [] to skip

'norm': mean-center, orthogonalize, and L2-norm basis set

'parametric_singleregressor':

Parametrically modulate onsets by modulator values, using single regressor with modulated amplitude Enter a cell array with modulator values for each event type, with a column vector (empty cell for no modulation)

'parametric_standard': Parametrically modulate onsets by modulator values, using two regressors per event type - One to model the average response, and one for the mean-centered modulator values of modulator values in each cell

'noampscale': Do not scale HRF to max amplitude = 1; default with SPM basis sets is to scale. Custom HRF entries are not scaled.

'noundershoot': Do not model undershoot - ONLY when using the canonical HRF

'customneural': Followed by a vector of custom neural values (instead of standard event/epochs), sampled in sec.

Limitation: can only handle two events max within the same TR

Outputs

X: model, sampled at TR

delta: indicator mtx, sampled at TR

delta_hires: indicator sampled at 16 Hz

hrf: hemodynamic response function, sampled at 16 Hz

Examples

```
X = onsets2fmridesign(ons, TR, size(imgs, 1) .* TR, 'hrf (with time derivative)');

X = onsets2fmridesign({[0 30 60]' [15 45 90']}, 1.5, 180, spm_hrf(1), 'parametric_standard', {[2
X = onsets2fmridesign({[0 30 60]' [15 45 90']}, 1, 180, spm_hrf(1), 'parametric_standard', {[2 .
X = onsets2fmridesign({[0 30 60]' [15 45 90']}, 2, 180, spm_hrf(1), 'parametric_standard', {[2 .
X = onsets2fmridesign({[0 30 60]' [15 45 90']}, 2, 180, spm_hrf(1), 'parametric_singleregressor'
X = onsets2fmridesign({[0 30 60]' [15 45 90']}, 1, 180, spm_hrf(1), 'parametric_singleregressor'

% Here, spm_hrf(1) is for the canonical HRF in spm.

X = onsets2fmridesign(ons, 2, size(dat.dat, 2) .* 2, [], [], 'noampscale');

X2 = onsets2fmridesign(onsp, 2, length(dat.removed_images) .* 2, [], [], 'customneural', report_
plotDesign(ons,[], TR, 'yoffset', 1);
```

A note on absolute scaling and efficiency: Scaling of the response influences efficiency. It does not affect model fits or power when the scaling is equated (constant across events in a design), but it does affect efficiency simulations.

Event duration: assumes that the max neural sampling rate is about 1 Hz, which produces responses that do not exceed about 5x the unit, single-event response. This is a reasonable assumption, and affects only the absolute scaling across ISIs/block lengths.

An event duration of about 1 sec produces a response of unit amplitude. The sampling resolution is 0.1 sec, so that is the lowest you can go - and it will produce lower response amplitude/less efficient designs, as less neural activity is being delivered. If event durations are not entered, then events will have unit amplitude by default.

See Also tor_make_deconv_mtx3, [or 2], plotDesign

Model_building_tools.**onsets2parametric_mod_X**(*ons*, *pm_vals*, *nscan*, *basisset*, *varargin*)

Examples

```
    ons = ons{1}; %obj.Sess(1).U(1).ons ./ TR;
    pm_vals = obj.Sess(1).U(1).P.P;
    nscan = obj.nscan(s);
```

Model_building_tools.**plot_ideal_deconv5**(*rmsd*, *msdstd*, *msdb*, *biasmean*, *meanest*, *min95est*, *max95est*, *INFO*, *hrf*, *snr*, *TR*, *varargin*)

Usage

```
    function plot_ideal_deconv5(rmsd,msdstd,msdb,biasmean,meanest,min95est,max95est,INFO,hrf,snr,TR,
```

delta should be matrix of column vectors

optional: truer, vector of "true" responses for each trial type

Model_building_tools.**spm_mat2batchinput**(*SPM*)
Extract info from SPM structure and format in batch input mode

Usage

```
    [imgs, TR, scanspersess, names, onsets, durations] = spm_mat2batchinput(SPM)
```

:Input into SPM8:

```
    tmod = time modulator (number per cell, order, linear = 1)
```

For parametric modulation include a structure array, which is up to 1 x n in size, called pmod. n must be less than or equal to the number of cells in the names/onsets/durations cell arrays. The structure array pmod must have the fields: name, param and poly. Each of these fields is in turn a cell array to allow the inclusion R for regressors of no interest

NOTES: This is a preliminary version of this function and does not extract covariates, parametric modulators, etc.

Returns output in the format you would use to enter into the SPM GUI to set up a model. Same format is used by canlab_spm_fmri_model_job.m

Model_building_tools.**tor_make_deconv_mtx3**(*sf*, *tp*, *eres*, *varargin*)

Usage

```
    function [DX,sf] = tor_make_deconv_mtx(sf,tp,eres,[opt] TRs before stim onset,[num. sessions],[d
```

Inputs

sf: cell array of stick functions, one per condition all sf cells should be of the same length

Or matrix of stick functions, 1 column per condition

tp: number of timepoints to estimate in hrf deconvolution matrix

eres: timebins in sf array for each TR

DX: deconvolution matrix estimates O.tp time points for each condition Time resolution is in TRs

Optional

1. TRs before: 0 or number of time-points to shift LEFT
2. number of sessions; if > 1, adds session-specific intercepts
3. docenter, 1/0 for do/do not center columns, default 0

4. **scanspersess: how many scans per session? prevents regressors from** running over into the next session (recursive).

No parametric modulation of sf's allowed.

Outputs

sf: stick function resampled at TR

3.11 Parcellation_tools

Parcellation_tools.**cluster_princomp**(*clusters*, *varargin*)

Usage

```
function [clusters,subclusters] = cluster_princomp(clusters,[behavioral score vector],[corr flag
```

ALSO TRY: subcluster_montage(subclusters{1}) % to plot the output

Inputs

clusters: is structure of clusters from tor_extract_rois.m

behavioral vector is row vector of behavioral or other scores to correlate

corr flag: *1 = work on correlations among voxels, 2 = work on covariance

plotflag: *1 = yes, 0 = no. plots.

locflag: 1 yes, *0 no; add XYZ voxel locations (scaled) to data submitted to clustering pushes voxels closer in space to be classified in the same cluster

try this to test the program on random data:

```
cl(1).all_data = randn(23,30);cl(1).numVox = 30;cl = cluster_princomp(cl,EXPT.behavior,1,1);
cl(1).all_data(:,1:10) = cl(1).all_data(:,1:10) + 10; cl = cluster_princomp(cl,EXPT.behavior,1,1
cl(1).all_data(:,25:30) = cl(1).all_data(:,25:30) + repmat((EXPT.behavior .* 3)',1,6);
cl(1).all_data(:,21:24) = cl(1).all_data(:,21:24) + repmat((1:23)',1,4);
cl = cluster_princomp(cl,EXPT.behavior,1,1);
```

mean-center everything now:

```
cl.PCA = [];
cl.all_data = cl.all_data - repmat(mean(cl.all_data),size(cl.all_data,1),1);
cl = cluster_princomp(cl,EXPT.behavior,1,1);
```

add another correlated group:

```
cl.all_data(:,1:5) = cl.all_data(:,1:5) + repmat(rand(23,1)*5,1,5);
cl = cluster_princomp(cl,EXPT.behavior,1,1);
```

if component scores are used and correlated with behavior, this means that the subjects tend to show the behavioral effect who also show the pattern associated with comp. x. this may mean high on a number of voxels, or high on some and low on others. the weights may be used to interpret what the components mean, and this can be done graphically.

t-tests on component scores have ambiguous interpretations, because a high t-score may indicate negative values or close-to-zero values on some voxels. a component could have the interpretation, "high on this component means high on V1 and low on V2."

classifying voxels is done using cluster analysis (hierarchical, centroid linkage) on the voxels (observations) using the PCA weights (eigenvectors) as variables. This lets the clustering algorithm work in the reduced

variable space with dimensionality equal to the number of components. The max number of clusters is restricted based on the gradient of the eigenvalues in the PCA maxclusters = 1 + the number of eigenvalues with gradient at least 20% of the initial drop from 1 to 2 eigenvalues.

Requires clustering library in Matlab. Robust option also uses the robust PCA algorithm RAPCA, created by: Hubert, M., Rousseeuw, P.J., Verboven, S. (2002),

> "A fast method for robust principal components with applications to chemometrics", by Mia Hubert, Peter J. Rousseeuw, Chemometrics and Intelligent Laboratory Systems, 60, 101-111.

Parcellation_tools.**inconsistent**(*Z*, *depth*)

Inconsistent values of a cluster tree.

Y = INCONSISTENT(Z) computes the inconsistent value of each non-leaf node in the hierarchical cluster tree Z. Z is a (M-1)-by-3 matrix generated by the function LINKAGE. Each inconsistent value is a measure of separation between the two clusters whose merge is represented by that node, compared to the separation between subclusters merged within those clusters.

Y = INCONSISTENT(Z, DEPTH) computes inconsistent values by looking to a depth DEPTH below each node.

Y is a (M-1)-by-4 matrix, with rows corresponding to each of the non-leaf nodes represented in Z. INCONSISTENT computes the inconsistent value for node (M+i) using S_i, the set of nodes less than DEPTH branches below node (M+i), excluding any leaf nodes. Then

- Y(i,1) = mean(Z(S_i,3)), the mean height of nodes in S_i
- Y(i,2) = std(Z(S_i,3)), the standard deviation of node heights in S_i
- Y(i,3) = length(S_i), the number of nodes in S_i
- Y(i,4) = (Z(i,3) - Y(i,1))/Y(i,2), the inconsistent value

The default value for DEPTH is 2.

See Also PDIST, LINKAGE, COPHENET, DENDROGRAM, CLUSTER, CLUSTERDATA.

Parcellation_tools.**mask_princomp**(*clusters*, *varargin*)

Usage

```
[clusters] = mask_princomp(clusters,[behavioral score vector],[corr flag],[plotflag],[saveflag])
```

This function is just like cluster_princomp, except that it works on the SET of activations in all clusters, rather than within each cluster.

clusters is structure of clusters from tor_extract_rois.m

behavioral vector is row vector of behavioral or other scores to correlate

corr flag: 1 = work on correlations among voxels, 2 = work on covariance

plotflag: 1 = yes, 0 = no. plots.

try this to test the program on random data:

```
cl(1).all_data = randn(23,30);cl(1).numVox = 30;cl = cluster_princomp(cl,EXPT.behavior,1,1);
cl(1).all_data(:,1:10) = cl(1).all_data(:,1:10) + 10; cl = cluster_princomp(cl,EXPT.behavior,1,1
cl(1).all_data(:,25:30) = cl(1).all_data(:,25:30) + repmat((EXPT.behavior .* 3)',1,6);
cl(1).all_data(:,21:24) = cl(1).all_data(:,21:24) + repmat((1:23)',1,4);
cl = cluster_princomp(cl,EXPT.behavior,1,1);
```

mean-center everything now:

```
cl.PCA = []; cl.all_data = cl.all_data - repmat(mean(cl.all_data),size(cl.all_data,1),1);
cl = cluster_princomp(cl,EXPT.behavior,1,1);
```

add another correlated group:

```
cl.all_data(:,1:5) = cl.all_data(:,1:5) + repmat(rand(23,1)*5,1,5);
cl = cluster_princomp(cl,EXPT.behavior,1,1);
```

if component scores are used and correlated with behavior, this means that the subjects tend to show the behavioral effect who also show the pattern associated with comp. x. this may mean high on a number of voxels, or high on some and low on others. the weights may be used to interpret what the components mean, and this can be done graphically.

t-tests on component scores have ambiguous interpretations, because a high t-score may indicate negative values or close-to-zero values on some voxels. a component could have the interpretation, "high on this component means high on V1 and low on V2."

classifying voxels is done using cluster analysis (hierarchical, centroid linkage) on the voxels (observations) using the PCA weights (eigenvectors) as variables. This lets the clustering algorithm work in the reduced variable space with dimensionality equal to the number of components. The max number of clusters is restricted based on the gradient of the eigenvalues in the PCA maxclusters = 1 + the number of eigenvalues with gradient at least 20% of the initial drop from 1 to 2 eigenvalues.

Requires clustering library in Matlab. Robust option also uses the robust PCA algorithm RAPCA, created by: Hubert, M., Rousseeuw, P.J., Verboven, S. (2002),

> "A fast method for robust principal components with applications to chemometrics", by Mia Hubert, Peter J. Rousseeuw, Chemometrics and Intelligent Laboratory Systems, 60, 101-111.

Parcellation_tools.**parcel_cl_nmds**(*parcel_cl_avgs*)

Usage

```
[parcel_cl_avgs, NMDS, class_clusters] = parcel_cl_nmds(parcel_cl_avgs)
```

Examples

```
load Parcellation_info/parcellation.mat
[parcel_cl_avgs, NMDS, class_clusters] = parcel_cl_nmds(parcel_cl_avgs)

parcel_cl_nmds_plots(parcel_cl_avgs, NMDS, 'save')
parcel_cl_nmds_plots(parcel_cl_avgs, NMDS, 'save', 'savedir', 'Parcellation_info')
```

Complete methods example

1.Dimension reduction

Clustering of multivariate data is most stable when the data is not sparse, i.e., the dimensionality is low relative to the number of observations. To limit the dimensionality of the data, a spatio-temporal dimension-reduction step is first performed on the [n x v x N] data matrix of AUC data for n trials x v voxels x N participants (here, n = 48 trials [usually], v = 17,112 voxels , and N = 26 participants). A temporal data reduction is first performed to identify components with correlated AUC trial time series within each participant, followed by a spatial reduction to identify components with correlated spatial patterns across subjects. First, the [n x v] matrix of AUC data for each participant was subjected to PCA, using the [v x v] correlation matrices. Based on the scree plots across subjects, we saved the first 7 eigenvectors (spatial maps). These eigenvectors explained 74 +- 2.5% (st. dev. across subjects) of the variance in the full dataset. These eigenvectors were scaled by their variances (eigenvalues) and concatenated across subjects to form an [v x N*7] matrix of eigenvectors. This matrix was subjected to second (across participant) PCA step to identify components with similar spatial maps

across participants. We retained 12 eigenvectors (maps) based on the scree plot, which explained 65% of the variance across individuals. Component scores in this space were used for clustering. This is a data reduction step, and the results are not expected to depend strongly on the number of eigenvectors retained at either step, as long as most of the variance in the data is explained.

2.Parcellation

Hierarchical agglomerative clustering with average linkage was used to group voxels into parcels–sets of contiguous voxels with similar profiles–in the [v x 12] matrix of component maps. The goal of parcellation was to reduce the space from voxels to parcels (regions) for non-metric multidimensional scaling (NMDS)-based clustering of regions, so that NMDS is computationally tractable. Voxels whose trial AUC time series did not correlate with that of other voxels in the same parcel at p < .001 (in a random effects analysis across participants) were pruned from each parcel. Out of 140 parcels total, 127 parcels with more than 3 voxels after pruning were retained for subsequent analysis.

3.NMDS and clustering

Parcels were now treated as the unit of analysis, and the AUC trial time series data were extracted from each parcel for each subject. Values in the [n trials x parcels x N subjects] data matrix were z-scored within participant to remove inter-subject differences in scaling, then concatenated into an [n*N x parcels] data matrix. Correlations among parcels were converted into a [parcels x parcels] matrix of distances using the formula distance = (1 - r) / 2. The NMDS stress plot (which operates on ranked distances and is therefore more robust to outliers and does not require a strictly Euclidean distance space) was examined and 7 dimensions (which explained 79% of the variance in distances) were retained for the final cluster analysis. Hierarchical agglomerative clustering with average linkage was used to cluster the parcels in this space into interconnected networks with correlated AUC trial time series. To choose the number of clusters in the final solution (k), for every possible choice of clusters between k = 2 and 20, we compared the cluster solution to the average and standard deviation of 1,000 clustering iterations with permuted parcel time series. This yielded a Z-score ([actual solution - mean permuted solution] / standard deviation of permuted solution for each value of k. The best solution was k = 13, with a value of Z = 5.57 compared to the null-hypothesis single-cluster solution (p < .0001).

Parcellation_tools.**parcel_cl_nmds_plots** (*parcel_cl_avgs*, *NMDS*, *varargin*)

Plot: data panel

case ‘save’, dosave = 1; case {‘savedir’, ‘mysavedir’}, mysavedir = varargin{i+1}; case {‘figs’, ‘dotfigs’}, savedotfigs = 1;

Doc not complete yet. Please update me!

See parcel_clusters.m See parcel_cl_nmds.m

load Parcellation_info/parcellation.mat

Examples

```
parcel_cl_nmds_plots(parcel_cl_avgs, NMDS, 'save')
parcel_cl_nmds_plots(parcel_cl_avgs, NMDS, 'save', 'savedir', 'Parcellation_info')
parcel_cl_nmds_plots(parcel_cl_avgs, NMDS, 'save', 'savedir', 'Parcellation_info_tor_mask_try1',
```

Parcellation_tools.**parcel_clusters** (*clpos_data*, *clneg_data*)

Usage

```
parcel_clusters(clpos_data, clneg_data)
```

No outputs. Saves all output in separate directory.

First, get eigenvectors for each subject.

We’re interested obtaining PARCELS of voxels that tend to co-activate, or have the same activation profile.

We can find these by using clustering algorithms to group voxels with similar profiles.

Because we have a many voxel x many voxel covariance matrix for each subject (lots of data!), it's important to reduce the dimensionality of the problem and peform clustering on a REDUCED_DIMENSIONAL space.

We use PCA to do this. Instead of clustering activation profiles (e.g., time-courses) directly, we cluster eigenvector loadings for each voxel on a reduced set of components that explains most of the variance in the data.

Similar voxels will have similar loadings across the set of eigenvectors. e.g., two voxels may load high on components [1 3 and 5], and low on components [10 and 13]. If they have the same pattern of loadings, they should be considered part of the same CLASS. Groups of voxels that are contiguous in space and are members of the same CLASS are called parcels.

Images and outputs are saved in their own subdirectory called Parcellation_info

The main outputs are: parcel_cl % parcels, one cell per subject, one parcel per

element within cells. same format as clpos_data

parcel_cl_avgs % parcels, one parcel per element within cells. same format as clpos_data2

parcel_cl_avgs(x).timeseries contains one cell per subject, with data averaged across voxels within that parcel for that subject This kind of output is useful, because you can input it directly into other mediation analyses. [paths, stats2] = mediation(SETUP.data.X, SETUP.data.Y, parcel_cl_avgs(1).timeseries, 'plots', 'verbose', 'names', {'Hi-Low Cue' 'Pain Report' 'Parcel'}, 'boot'); cluster_orthviews(parcel_cl_avgs(1), {[0 1 0]}, 'add');

```
cd('/Volumes/SCNAlpha/Data_and_Tools/SpeechTask/analysis/wb_multisubject_correl_HR_corrected/med
load cl_b_fdr05_002_01_k3_1_1_prune
```

then run

There are 2 dimension-reduction steps:

1. within-subjects
2. is on eigenvectors concatenated across subjects

Parcellation_tools.**parcel_complete_sets**(*s*, *varargin*)

Usage

```
[mysets,n_in_set,sets_by_vars,classes] = parcel_complete_sets(s,['dounique','nofuzzy'])
```

Find sets of elements in logical n x n matrix s in which all pairs in set are 'true' (1 in matrix s)

Optional: input a distance/correlation/etc. matrix and threshold

Optional Inputs

'dounique': provides single-variable sets as well

'nofuzzy': chooses closest set for each var so that each var

can belong to only one set

'threshold': followed by threshold thr for matrix s.

if thr is a string, thr will be evaluated on s: e.g., 's < 10'

in this case, thr should be a logical expr. involving s

if thr is a number, s >= thr will be evaluated

the resulting logical matrix will be used to determine sets

'min' or 'max': works only with 'nofuzzy' option.

if max, uses max to find most similar set; good if s is a covariance matrix

if min, uses min to find closest set; good if s is a distance matrix

default is max

runs on Matlab 7.2 or higher %%%

Examples

```
% Find sets of coordinates within 10 mm of one another
xyz = cat(1,cl{1}.mm_center);
d = pdist(xyz); d = squareform(d);
[mysets,n_in_set,sets_by_vars,classes] = parcel_complete_sets(d,'dounique','nofuzzy','threshold'
```

Parcellation_tools.**parcel_images**(*image_names*, *extract_mask*, *nuisance_covs*)

Usage

```
parcel_images(image_names, extract_mask, nuisance_covs)
```

This function performs the following steps, in this order

•map mask to functional space

•extract data from in-mask voxels

•remove nuisance covariates (before assessing connectivity)

•data reduction (pca)

•plot cases (detect outliers)

•separate data into a priori anatomical regions (LBPA40 hard-coded right now; downloadable from web; see wiki) (save label image mapped to functional space)

•cluster voxels in each region to get parcels

•save parcels and images

•NMDS on the parcels to group them into "networks" (default = use rank data)

•Visualization of the networks

Outputs

Creates and goes to a new directory: parcel_images_output

Outputs saved to disk include

1. An image with unique numerical codes for each parcel
2. A 'clusters' structure containing the parcels, with image data extracted and averaged over voxels within each parcel

Inputs

image_names: names of images to extract data from, and to use for functional parcellation. SEE ALTERNATE FORMAT BELOW FOR DIRECT DATA INPUT

extract_mask: a mask image

nuisance_covs: columns of a matrix to remove from the data, so that this subspace is not used to determine connectivity

e.g., nuisance_covs = SPM.xX.X(:, 1:3); if these are nuisance covariates...

Alternate input formats for image_names:

If you have data already extracted, image_names can be a structure with these fields:

image_names.V, spm_vol-style volume info for the mask volume and image space image_names.data, extracted data from all valid in-mask (non-zero, non-nan) voxels, one column per voxel, in standard matlab (:) order. The mask and the data must match!

3.12 peak_coordinates

peak_coordinates.**cluster_manova**(*clusters*, *fnames*, *varargin*)

Usage

```
cluster_manova(clusters,fnames,verbose)
```

Inputs

clusters: is output of clusters2database, with all fields from database

fnames: is cell array of strings with names to test e.g., {'Rule' 'Task'}

uses stats toolbox

verbose: optional, produces more output and tests

peak_coordinates.**extract_ind_peak**(*imnames*, *cl*, *varargin*)

This function gets individual spatial peaks from clusters.

Usage

```
[clusters] = extract_ind_peak(imnames [can be empty],clusters,[vols])
```

Inputs

imnames:: a matrix of image names, in spm_list_files output format

if imnames is empty, enter full data (vols) as 3rd argument or else this program will use cl.all_data to get data

cl: clusters, see tor_extract_rois

NOTE (WARNING): WORKS ON XYZ VOXEL COORDINATES - NO TRANSFORMATION TO DIFFERENT SPACES see transform_coordinates.m for this.

Functions called C:matlabR12toolboxmatlabdatatypessqueeze.m

c:tor_scriptsvoistatutilitynanmean.m (calls other spm functions)

See Also cluster_manova

peak_coordinates.**image2coordinates**(*img*)

Threshold a statistic image and turn it into a list of x, y, z coordinates (in mm "world space") with SPM

Usage

```
XYZmm = image2coordinates(img)
```

Examples

```
    img = 'h25_aerger.img';
    XYZmm = image2coordinates(img)
    figure;
    plot3(XYZmm{1}(1, :)', XYZmm{1}(2, :)', XYZmm{1}(3,:)', 'ko');
    addbrain
    axis image
```

Example with multiple images:

```
    % list all images in dir
    imgs = filenames(fullfile(pwd, '*img'), 'char', 'absolute')
    XYZmm = image2coordinates(imgs);
    % save images names and coordinates
    % imgs2 has cell array of image names, without full paths
    imgs2 = filenames(fullfile(pwd, '*img'));
    save silke_senders_xyz_coordinates imgs imgs2 XYZmm
```

peak_coordinates.**spatial_contrast**(*XYZ1*, *XYZ2*)

This function tests relative locations of individual spatial peaks from clusters.

Usage

```
    [str,dcon] = spatial_contrast(XYZ1,XYZ2)
```

plots position of XYZ2 relative to XYZ1 thus, an 'anterior' group position means that XYZ2 peaks are anterior to XYZ1 peaks

Inputs

XYZ1,2: n x 3 coordinates (in mm)

Output

con: contrast vector, e.g., [1 -1]

See help conf_region for details of the test.

3.13 ROI_drawing_tools

ROI_drawing_tools.**add2mask**(*mask*, *x*, *r*, *varargin*)

Adds or subtracts spheres around x coordinates to/from existing mask

Usage

```
    add2mask(mask, x, r,varargin)
```

Inputs

mask: is a string filename

x: is n x 3 list of coordinates

r: is radius

a 4th argument causes us to SUBTRACT!

Examples

```
    mask = 'insula_from_part1.img'
    x = [-29.6 28.6 5.3; 37.0 27.5 3.2; -34.9 8.5 -14.8;38.1 10.6 -14.8];
    add2mask(mask,x,8);
```

ROI_drawing_tools.**clusters2roimask**(*cl*)

The purpose of this function is to facilitate making masks with ROIs for future studies, give a clusters structure. ROIs are constrained to be within activation blobs specified by input clusters, and are masked by selected ICBM regions. Clusters may be smoothed before or after masking.

Usage

```
    [clout] = clusters2roimask(cl)
```

Option to do 3 things, in order:

1. Enlarge selected clusters
2. Mask clusters with anatomical regions from ICBM atlas
3. Subdivide clusters using hierarchical clustering of voxel coordinates

Output is a clusters file and a mask file in a 2 x 2 x 2 standard brain space.

ROI_drawing_tools.**draw_anatomical_roi_2008**(*meth*, *varargin*)

Quick start guide: type draw_anatomical_roi_2008

Usage

```
    draw_anatomical_roi_2008('init');
    draw_anatomical_roi_2008('load', 'ROI_midbrain.img');
    draw_anatomical_roi_2008('init', 'overlay',  'remi_mean_T2.img');
```

Inputs no arguments: init

'init': initialize gui and orthviews and remove previous ROI

'load': load an ROI from a mask

'free': draw an ROI freehand (click on one of the slices in the Slices window 1st)

'poly': don't run this yet

'add': add a region you've drawn on a slice to your ROI

'remove': remove a region you've drawn from your ROI

'smooth': 3-d smoothing of ROI

'write': write mask image of ROI and return clusters to workspace

'exit': exit. ROI data is stored in the Slices figure, so you can continue to edit, etc. after exiting.

Note

You can use cluster_orthviews to image multiple blobs, and then draw relative to those. this function saves it's data in the Slices figure, so you can draw, re-initialize the orthviews, and keep drawing before you save.

Examples

```
cluster_orthviews(red, {[1 0 0]}, 'overlay', 'remi_mean_T2.img');
cluster_orthviews(stn, {[0 1 0]}, 'add');
set(findobj('Tag','Graphics'), 'WindowButtonUpFcn', 'draw_anatomical_roi_2008(''moveslice'');');
% Use the spm_orthviews menu to ZOOM IN...and keep drawing!
```

Example of brainstem ROI drawing:

```
draw_anatomical_roi_2008('init', 'overlay',  'remi_mean_T2.img');
set(findobj('Tag','Graphics'), 'WindowButtonUpFcn', '');
cluster_orthviews(red, {[1 0 0]}, 'overlay', 'remi_mean_T2.img');
stn = mask2clusters('ROI_STN.img');
cluster_orthviews(stn, {[0 1 1]}, 'add');
% Now zoom in to the midbrain in SPM orthviews and draw new ROIs
```

ROI_drawing_tools.**sphere_mask**(*P*, *XYZmm*, *r*, *outname*, *varargin*)

Creates mask images and clusters for a set of spheres defined around coordinates you specify. Used for creating regions of interest (ROIs).

Usage

```
[clusters,maskCLU] = sphere_mask(fname,XYZmm,r,outname,[maskname],[overlay])
```

Spheres may be masked with an anatomical mask file.

Inputs

P: is input image name with correct dimensions and vox sizes for your study

XYZmm: is mm coordinates (row vector) for sphere center

r: is radius in mm

outname: is string for output mask name, e.g., 'sphere_mask.img'

[maskname]: is optional mask .img containing additional constraints (e.g., gray matter mask, etc.), can be in different dimensions

See Also mask2clusters, montage_clusters

Examples

```
tmp = sphere_mask(EXPT.SNPM.P{1}(1,:),[25.6 -45.5 77.0],10,'test.img','ICBM_area74.img');
icbm_localize(tmp)

M =which('ICBM_brainonly_1mm_seg1.img')
cl = sphere_mask(d(1).name,[35 -57 54; -23 -59 56;13 -63 62; -7 -77 50; 25 -79 30; -23 -81 18;41
```

Matlab 6.5/OSX bug gives seg fault or something if mask is too big.

3.14 Statistics_tools

Statistics_tools.**Bspline**(*t*, *k*, *u*, *v*, *ForceSup*)

Usage

```
Bspline(t,k,u[,v])
```

Create B-Spline basis of order k, with knots u, evaluated at t. If control verticies v are specified then then B is the spline function instead of the basis.

u must be at least length(k)+1

Statistics_tools.**ContinuousAccuracy**(*obj*, *pattern*, *predrange*, *unit*)

Calculate forced choice accuracy for unit increases in continuous predictions. Requires units to be ranked ordered from 1:end. Will work for Gianaros or Pain datasets. Accuracies are not penalized for missing cases. Probably best to run this on single subjects and then aggregate accuracies across subjects.

Usage

```
acc = ContinuousAccuracy(obj, pattern, unit)
```

Inputs

obj: fmri_data() object with data stacked by increasing levels of prediction. Make sure obj.Y includes the training labels

pattern: fmri_data() object with weight pattern

predrange: specify the range of predictions (e.g., 1:5)

unit: specify the unit increase in prediction (e.g., 1 or 2)

Outputs

acc: accuracy of prediction for specified units

Examples

```
acc = ContinuousAccuracy(dat, pine, 1:5, 1)
```

Statistics_tools.**F_test_full_vs_red**(*y*, *X*, *Xred*, *px*, *pxred*)

Usage

```
[F, p, resid, df_model, df_error] = F_test_full_vs_red(y, X, Xred, px, pxred)
```

Examples

```
X = randn(100, 3); Xred = X(:,1); y = X(:,2) + randn(100, 1);
px = pinv(X); pxred = pinv(Xred);
[F, p, resid] = F_test_full_vs_red(y, X, Xred, px, pxred);


% Test full-model F-value against regress.m
Xred = X(:,end); % intercept only
px = pinv(X);
pxred = pinv(Xred);
[F, p, resid, dfm, dfe] = F_test_full_vs_red(y, X, Xred, px, pxred); % full model F-test
[b, bint, r, rint, stats] = regress(y, X);
```

Statistics_tools.**F_test_no_intercept**(*X*, *y*, *s*)

Test of the null hypothesis that the ENTIRE regression model X explains no variance in the data (y)

Usage

```
[Fobs, p, dfb, dfe] = F_test_no_intercept(X,y,s)
```

Examples

Assess false positive rate with robust regression:

```
    iter = 5000;
    warning off
    fvals = zeros(1,iter); pvals = fvals;
    for i = 1:iter
      X = randn(10,1); y = randn(10,1);
      [bb,stats]=robustfit(X,y);
      % [bb,dev,stats] =glmfit(X,y);
      [Fobs, p, dfb, dfe] = F_test_no_intercept(X,y,stats.s);
      fvals(i) = Fobs; pvals(i) = p;
    end
    fpr = sum(pvals<.05) ./ iter
    warning on
```

Based on

Johnson & Wichern, 5th ed., p. 371, Result 7.6

for a regression model with k predictors, and q regression parameters in reduced model, test the addition of q+1...k by extra sums of squares [n(s2red - s2full) / (k-q)] / [n*s2full / (n-k-1)] ~ F(k-q),(n-k-1)

Statistics_tools.**ICC**(*cse*, *typ*, *dat*)

Function to work out ICCs according to shrout & fleiss' schema (Shrout PE, Fleiss JL. Intraclass correlations: uses in assessing rater reliability. Psychol Bull. 1979;86:420-428).

Usage

```
        iccvalue = ICC([1 to 6],['single' or 'k'], data matrix)
```

Inputs

'dat': is data whose *columns* represent k different raters (judges) & whose *rows* represent n different cases or targets being measured. Each target is assumed to be a random sample from a population of targets.

'cse': is either 1,2,3. 'cse' is: 1 if each target is measured by a different set of raters from a population of raters, 2 if each target is measured by the same raters, but that these raters are sampled from a population of raters, 3 if each target is measured by the same raters and these raters are the only raters of interest.

'typ': is either 'single' or 'k' & denotes whether the ICC is based on a single measurement or on an average of k measurements, where k = the number of ratings/raters.

This has been tested using the example data in the paper by shrout & fleiss.

Example: out = ICC(3,'k',S_Fdata) returns ICC(3,k) of data 'S_Fdata' to double 'out'.

Kevin Brownhill, Imaging Sciences, KCL, London kevin.brownhill@kcl.ac.uk

Additional documentation

iccvalue = ICC([1 to 6],['single' or 'k'], data matrix)

Here, columns are 'judges', or more generally, 'measures' that are usually ideally intercorrelated. Rows are items being assessed. The ICC assesses the proportion of variance attributed to the items,

shared across measures.

As the correlation between the measures grows, the icc grows. Another way of saying this is that if the rows are consistently different across measures, the icc will be high.

Think of rows as criminals, and columns as judges. The data values are 'guilt scores', where higher is more guilty. If all the judges agree, the most guilty cases will be rated as most guilty by all judges, and the icc will be high. This is actually consistent with Case 2 or 3 in Shrout and Fleiss.

In Case 1, the columns don't have any real meaning, as there are different 'judges' for each row, and variance components due to judge cannot be separated from error and the judge x target interaction. In Case 2 and 3, they are crossed. Case 2 treats judge as a random effect, whereas Case 3 treats judge as a fixed effect.

If the data were an individual differences study of cognitive performance, then the rows would be subjects, and the columns would be tests. A high icc would indicate a high correlation across the tests, which indicates that subjects are reliably different from one another, i.e., that a large proportion of the total variance is related to subject. In such a case, as tests are fixed entities, then Case 3 might be appropriate.

Cronbach's alpha is equal to ICC(3, k) - case 3, k This assumes no target x rater interaction

Examples

```
dat = mvnrnd([1 1 1], [1 .5 .5; .5 1 .5; .5 .5 1], 50); whos dat
corrcoef(dat)
ri = ICC(2, 'k', dat)
dat = mvnrnd([1 1 1], [1 .9 .9; .9 1 .9; .9 .9 1], 50); whos dat
corrcoef(dat)
ri = ICC(2, 'k', dat)
```

In the example below, judges (measures) have systematically different means, and the ICC values are different. ICC(1, 1) is low because judge is not considered as a source of variance. ICC(2, 1) is higher, but intermediate, because judge is considered as a random effect and modeled, but we want to generalize to new judges. ICC(3, 1) is highest, because judge is modeled

dat = mvnrnd([1 2 3], [1 .5 .5; .5 1 .5; .5 .5 1], 50); whos dat ri = ICC(1, 'single', dat) ri = ICC(2, 'single', dat) ri = ICC(3, 'single', dat)

number of raters/ratings

Statistics_tools.**ancova**(*groups*, *x*, *y*, *varargin*)

Usage

```
b,t,p,pthandles] = ancova(groups,x,y,[plot],[covs of no interest])
```

Outputs

Elements of b, t, p: 1st = intercept, 2nd = group effect, 3rd = slope, 4th = grp x slope interaction

recursive – call ancova repeately if y is a matrix get pairwise standardized slopes (corrs) and group diffs and slope interaction for all pairs of y vectors

Statistics_tools.**barplot_get_within_ste**(*dat*, *varargin*)

Usage

```
[se_within, stats] = barplot_get_within_ste(dat)
```

Compute within-subjects standard errors

Note The old version of this used average s.e.'s for contrasts

of interest, which depend on contrast scaling and are less appropriate for use as error bars.

The new version as of 12/10 uses the Loftus & Masson 1994 method and has been checked against the data in that paper.

See help below for the L & M data and additional code for getting the ANOVA decomposition.

Useful for barplots of conditions when calculating within-subject SEs

Examples

```
%% data from Loftus & Masson, Table 2

mtx = [1 10 13 13 12.00
2 6 8 8 7.33
3 11 14 14 13.00
4 22 23 25 23.33
5 16 18 20 18.00
6 15 17 17 16.33
7 1 1 4 2.00
8 12 15 17 14.67
9 9 12 12 11.00
10 8 9 12 9.67];

dat = mtx(:, 2:4);  % data from Loftus & Masson, Table 2

[se_within, stats] = barplot_get_within_ste(dat)
fprintf('Within ste: %3.2f, 95%% CI: mean +/- %3.2f\n', se_within, stats.ci);
```

Extra stuff from ANOVA table

Mean square for condition: Variance of condition means * sample size...average squared variance accounted for by condition means

```
[n, k] = size(dat);
MS_cond = var( mean(dat) - mean(dat(:)) ) * n;

MS_subject = var( mean(dat') - mean(dat(:)) ) * k;

datv = dat(:);
MS_total = scale(datv, 1)' * scale(datv, 1);
```

Statistics_tools.**bayes_get_probabilities**(*Y*, *Xi*, *k*)

Usage

```
[priors, pa1_given_t, pa0_given_t, pt_given_act1, pt_given_act0, pa1_given_not_t] = bayes_get_pr
```

Inputs

k: is regularization param, biases towards 0.5

Y: is data matrix, obs x features, 1/0 (active/not)

Xi: is task indicator matrix

This is a sub-function of classify_naive_bayes.m For complete help, see classify_naive_bayes.m

Statistics_tools.**bayes_get_probabilities_2010**(*Y*, *Xi*, *k*)

Usage

```
[priors_Pt, pa1_given_t, pa0_given_t, pt_given_act1, pt_given_act0, pa1_given_not_t] = bayes_get
```

Inputs

k: is regularization param, biases P(activity | task class) towards 0.5

kY: is data matrix, obs x features, 1/0 (active/not)

kXi: is task indicator matrix

This is a sub-function of classify_naive_bayes.m For complete help, see classify_naive_bayes.m

Statistics_tools.**bayes_meta_feature_abstract**(*Y*, *thresh*, *shrink*, *bayes_model*, *volInfo*, *doplot*)

Purpose: Instead of original matrix of activations in studies x voxels, we may want to work with a data matrix of activations in contiguous regions x voxels

This function calculates Y = 0,1 for Y = 1 in any voxel in each contiguous cluster

Examples

```
reducedY = bayes_meta_feature_abstract(Y, .1, bayes_model, volInfo);
xval = classify_naive_bayes('xval', reducedY, Xi, 0, 0, bestk, bestg );
xval.prop_correct_by_class
```

needs bayes_model.pa1_given_t, nclasses, params.k

Statistics_tools.**binotest**(*X*, *p*)

Test the number of "hits" in each column of X against a null-hypothesis proportion p, using a binomial test.

Usage

```
RES = binotest(X, p)
```

Assumes elements of each column of X are independent Bernoulli trials.

Inputs

X: is a matrix of "hits" and "misses", coded as 1s and 0s.

p: is the null hypothesis proportion of "hits", e.g., often p = 0.5

Statistics_tools.**binotest_dependent**(*X*, *Po*)

This function runs several different types of tests on dependent binomial data.

Usage

```
[varargout] = binotest_dependent(X, Po)
```

Overall, it tests the number of "hits" for each subject (row in X) against a null-hypothesis proportion p, across all subjects using a Z-test (two-tailed). The second level null hypothesis should be approximated by a normal distribution with a mean of p. This approach assumes that each subject has an equal number of independent Bernoulli trials (columns in X) and that the number of subjects exceeds n=20 the test will be more accurate as n -> infinity. Also, calculates tests for each separate trial (e.g., subject columns), and the difference between proportions (two proportion z-test).

Inputs

X: X is a matrix of "hits" and "misses", coded as 1s and 0s. where rows = subjects and columns = observations within subject

Po: Po is the null hypothesis proportion of "hits", e.g., often p = 0.5

Outputs

RES [1:5]: a structure containing the output of the stats for the z-test, includes the number of subject (N), number of overall hits (hits), the overall proportion of hits (prop), the standard deviation (SE), z-statistic (Z), and the two tailed p-value (pval) trial across all subjects. Assumes independence

RES1: Independent Single Interval Test for Column 1 (Column 1 only against Po)

RES2: Independent Single Interval Test for Column 2 (Column 2 only against Po)

RES3: Two proportion dependent difference z-test (Column 1 minus Column 2 against 0)

RES4: Dependent single-interval test (Mean of Column 1 and Column 2 against Po)

RES5:

Two proportion dependent addition z-test (Column 1 plus Column 2 against 2 * Po)
(Similar to mean, not sure what this will be used for)

Examples

```
[RES1, RES2, RES3, RES4, RES5] = binotest_dependent([1,1,1,1,0; 1,0,1,0,1]',.5)
```

Statistics_tools.**cancor**(*X*, *w*, *varargin*)

Usage

```
[cc,stats] = cancor(X,w,[permutations],[MCD robust outlier removal])
```

Inputs

X: data matrix, columns are variables, rows observations

w: first w columns are Set 1, rest are Set 2

cc: canonical correlations

UV: canonical variates

UVa: caonical variates for set a

UVb: canonical variates for set b

ccor: correlations between canonical variates and X variables

ab: weights of canon. variates in row vectors, e.g., U = Xa'

Johnson & Wichern, Applied Multivariate Statistical Analysis, 5th ed. tested on examples from the book. cc and ab are right, not positive about ccor

Example

Compute canonical correlations between first 2 and last 2 columns of X, testing against permuted column data with 1000 iterations.

```
[cc,stats] = cancor(X,2,1000,1);
```

Statistics_tools.**canlab_connectivity_predict**(*dat*, *subject_grouping*, *varargin*)
Connectivity and multivariate pattern-based prediction for multi-subject timeseries data Currently runs predictive algorithm(s) on pairwise correlations among regions

Usage

```
OUT = canlab_connectivity_predict(dat, subject_grouping, ['outcome', outcome_dat])
```

Features

- Within-subject correlation matrices and 'random effects' statistics
- [optional] Prediction with LASSO-PCR/SVR/SVM of outcomes from pairwise connectivity
- Time-lagged cross-correlation options

- Graph theoretic measures
- [optional] Prediction with LASSO-PCR/SVR/SVM of outcomes from graph measures
- Can easily be extended to handle partial regression/correlation coefficients

Inputs

dat: concatenated data matrix of time points (t) within subjects x variables (e.g., ROIs) [subj x time] x [variables]

subject_grouping: [subj x time]-length integer vector of which observations belong to which subjects, e.g., [1 1 1 ... 2 2 2 ... 3 3 3 ...]'

Optional Inputs

'outcome':

followed by outcome data for multivariate prediction. connectivity values and graph metrics are used to predict outcome data.

'shift_by': Followed by integer value for max number of time points to shift

'partialr': Use partial correlation instead of raw correlation

Outputs

OUT: A structure containing subject correlation matrices, the mean matrix, and raw and FDR-thresholded group matrix Also contains matrices with [subjects x variables] pairwise correlation elements and graph metrics

Examples

```
% Use partial correlations:
OUT = canlab_connectivity_predict(dat, subject_grouping, 'partialr');

% Omit graph met
OUT = canlab_connectivity_predict(dat, subject_grouping, 'outcome', y, 'nograph');
```

See also

parcel_cl, parcel_cl_nmds_plots, canlab_force_directed_graph, canlab_connectivity_preproc

Statistics_tools.**classify_bayes**(*meth*, *y*, *Xi*, *varargin*)

Usage

```
[corrclass, taskclass, realclass, likeratio, m, misclass,ptask,indx,cl] = classify_bayes(meth,y,
```

Inputs

y: observations (e.g., studies) x variables (e.g., brain voxels)

y = SOMResults.dat';

Xi: Xi = MC_Setup.Xi(:,1:2);

meth: can be: {'linear','diagLinear','quadratic','diagQuadratic','mahalanobis'}

discriminant analysis 'bayes' : simple Bayes posterior prob classifier

Optional inputs

Feature selection parameters:

selectivity_cutoff: max probability of task given a response in a variable, divided by number of tasks.

1 = variable must exceed .5 for 2 tasks, .2 for 5 tasks, etc.

1.5 = .3 for 5 tasks, .75 for 2 tasks, etc.

0 = no selectivity

activation_cutoff: max proportion of studies of some type that produced a response in a variable (e.g., voxel)

.1 is default

0 is no selectivity

Examples

```
[corrclass, taskclass, realclass, likeratio, m, misclass] = ...
classify_bayes('bayes',MC_Setup.unweighted_study_data',Xi,'selectivity_cutoff',1); corrclass

[corrclass, taskclass, realclass, likeratio, m, misclass] =
classify_bayes('bayes',MC_Setup.unweighted_study_data',Xi,'selectivity_cutoff',1,'activation_cut
```

Batch modes: Permutation test: input 'permtest' followed by number of permutations Do not request more than one output variable [corrclass_nullhyp] = classify_bayes('bayes',dat,Xi,'selectivity_cutoff',1.8,'activation_cutoff',.02,'permtest',5);

To get mask index of which areas meet feature selection: whsave = sum(indx > 0, 2) > 0;

select features first, instead of in x-validation (makes x-val invalid; don't do it)

Statistics_tools.**classify_choose_most_likely**(*ptask*, *testvec*)

Usage

```
[taskclass,maxlike,likeratio, taskprob] = classify_choose_most_likely(ptask,testvec)
```

Inputs

ptask: likelihood of each task given activation, p(task | activation) classses x variables (features, brain voxels)

testvec: activation values across features variables x 1

Output

taskclass: integer for which is max likelihood class

maxlike :

log likelihood of chosen class given data (if testvec is 1/0 indicator)

likeratio : likelihood ratio for most likely vs. least likely class

Examples

```
% voxels in original image space that were in dataset
whsave = sum(indx > 0, 2) > 0;

[tc, ml, lr] = classify_choose_most_likely(ptask, MC_Setup.unweighted_study_data(whsave,1))
```

Statistics_tools.**classify_naive_bayes**(*meth*, *varargin*)

Naive Bayes classifier

Usage

```
% set up model structure
bayes_model = classify_naive_bayes('setup', Y, Xi, [activation_cutoff, selectivity_cutoff]);
```

Inputs

Y: is full (not sparse); empty features will be eliminated

Xi: is obs x classes, an indicator matrix of 1's and 0's

TEST test classifier; make a prediction about classes from data

```
[class_est log_joint best_log map p_obs_act_given_class] = classify_naive_bayes('test', Y, bayes
```

EVAL evaluate classification accuracy

```
[prop_correct, confusion_mtx, misclass, prop_correct_by_class, chance, chance_95_ci] = classify_
```

APPARENT apparent classification; with full dataset

```
bayes_model = classify_naive_bayes('apparent', Y, bayes_model);
```

XVAL crossvalidate

```
xval = classify_naive_bayes('xval', Y, Xi);

bayes_model = classify_naive_bayes('write', bayes_model, Y, volInfo, conditionnames)
bayes_model = classify_naive_bayes('write', bayes_model, Y, MC_Setup.volInfo, MC_Setup.Xinms);
```

To add feature abstraction step within xval:

```
xval = classify_naive_bayes('xval', Y, Xi, 0, .9, .05, 1, volInfo );
```

PLOT

```
classify_naive_bayes('plot', bayes_model, ['pa|t', 'lr', 'lr surface',  'map plot', or 'class pl
```

Optional Inputs (any order)

Threshold (abs. value), and colors in cell array

```
classify_naive_bayes('plot', bayes_model, 'lr', .10, {[1 .7 0] [0 0 1]});
```

Examples

```
[bayes_model, Y] = classify_naive_bayes('setup', Y, Xi);

% Test obs. 2
tic, [class_est log_joint best_log map] = classify_naive_bayes('test',Y(2,:), bayes_model); toc

% Get apparent classification rate and look at confusion matrix
```

```
    tic, bayes_model = classify_naive_bayes('apparent', Y, bayes_model); toc
    bayes_model.apparent.confusion_mtx

    % Cross-validate
    bayes_model.xval = classify_naive_bayes('xval', Y, Xi);
    bayes_model.xval.confusion_mtx
```

Example 2

```
     Select features, and do apparent and cross-validated classification
    Y = MC_Setup.unweighted_study_data';
    wh = sum(Y) > 5;
    Y = Y(:, wh);
    whos Y
    [bayes_model, Y] = classify_naive_bayes('setup', Y, Xi);
    bayes_model = classify_naive_bayes('apparent', Y, bayes_model);
    bayes_model.apparent.prop_correct, bayes_model.apparent.confusion_mtx
    bayes_model.xval = classify_naive_bayes('xval', Y, Xi);
    bayes_model.xval.prop_correct, bayes_model.xval.confusion_mtx
```

Example 3 create_figure('hist'); hist(bayes_model.pa1_given_t, 100); xval = classify_naive_bayes('xval', Y, Xi, .05, .3);

Get results from key regions and run classifier only on those regions:

```
     cl = classify_naive_bayes('plot', bayes_model, 'lr', .10, {[1 .7 0] [0 0 1]});
    [studybyroi,studybyset] = Meta_cluster_tools('getdata',cl{1},dat',volInfo);
    bayes_model_regions = classify_naive_bayes('setup', studybyroi, Xi);
    bayes_model_regions = classify_naive_bayes('apparent', studybyroi,bayes_model_regions);
    disp('Apparent confusion')
    disp(bayes_model_regions.apparent.confusion_mtx)

    bayes_model_regions.xval = classify_naive_bayes('xval', studybyroi, Xi);
    disp('Cross-validated confusion')
    disp(bayes_model_regions.xval.confusion_mtx)

    fprintf('Proportion correct: Apparent: %3.0f%%  Xval: %3.0f%%\n', 100*bayes_model_regions.appare
    fprintf('Proportion correct by class: \t'); fprintf('%3.0f%%\t', 100*bayes_model_regions.xval.pr
    fprintf('\n');
    sz = cat(1,cl{1}(:).numVox);
    cl{1}(sz < 10) = [];

    subcl = subclusters_from_local_max(cl{1}, 10);
```

Statistics_tools.**classify_naive_bayes_2010** (*meth*, *varargin*)

Naive Bayes classifier

Usage

```
    % set up model structure
    bayes_model = classify_naive_bayes('setup', Y, Xi, [activation_cutoff, selectivity_cutoff]);
```

Inputs

Y: is full (not sparse); empty features will be eliminated

Xi: is obs x classes, an indicator matrix of 1's and 0's

TEST test classifier; make a prediction about classes from data

```
[class_est log_joint best_log map p_obs_act_given_class] = classify_naive_bayes('test', Y, bayes
```

EVAL evaluate classification accuracy

```
[prop_correct, confusion_mtx, misclass, prop_correct_by_class, chance, chance_95_ci] = classify_
```

APPARENT apparent classification; with full dataset

```
bayes_model = classify_naive_bayes('apparent', Y, bayes_model);
```

XVAL crossvalidate

```
xval = classify_naive_bayes('xval', Y, Xi);

bayes_model = classify_naive_bayes('write', bayes_model, Y, volInfo, conditionnames)
bayes_model = classify_naive_bayes('write', bayes_model, Y, MC_Setup.volInfo, MC_Setup.Xinms);
```

To add feature abstraction step within xval:

```
xval = classify_naive_bayes('xval', Y, Xi, 0, .9, .05, 1, volInfo );
```

:PLOT

```
classify_naive_bayes('plot', bayes_model, ['pa|t', 'lr', 'lr surface',  'map plot', or 'class pl
```

Optional Inputs (any order)

Threshold (abs. value), and colors in cell array

```
classify_naive_bayes('plot', bayes_model, 'lr', .10, {[1 .7 0] [0 0 1]});
```

Examples

```
[bayes_model, Y] = classify_naive_bayes('setup', Y, Xi);

% Test obs. 2
tic, [class_est log_joint best_log map] = classify_naive_bayes('test',Y(2,:), bayes_model); toc

% Get apparent classification rate and look at confusion matrix
tic, bayes_model = classify_naive_bayes('apparent', Y, bayes_model); toc
bayes_model.apparent.confusion_mtx

% Cross-validate
bayes_model.xval = classify_naive_bayes('xval', Y, Xi);
bayes_model.xval.confusion_mtx
```

Example 2

Select features, and do apparent and cross-validated classification

```
Y = MC_Setup.unweighted_study_data';
wh = sum(Y) > 5;
Y = Y(:, wh);
whos Y
[bayes_model, Y] = classify_naive_bayes('setup', Y, Xi);
bayes_model = classify_naive_bayes('apparent', Y, bayes_model);
bayes_model.apparent.prop_correct, bayes_model.apparent.confusion_mtx
bayes_model.xval = classify_naive_bayes('xval', Y, Xi);
bayes_model.xval.prop_correct, bayes_model.xval.confusion_mtx
```

Example 3:

```
create_figure('hist'); hist(bayes_model.pa1_given_t, 100);
xval = classify_naive_bayes('xval', Y, Xi, .05, .3);
```

Get results from key regions and run classifier only on those regions:

```
cl = classify_naive_bayes('plot', bayes_model, 'lr', .10, {[1 .7 0] [0 0 1]});
[studybyroi,studybyset] = Meta_cluster_tools('getdata',cl{1},dat',volInfo);
bayes_model_regions = classify_naive_bayes('setup', studybyroi, Xi);
bayes_model_regions = classify_naive_bayes('apparent', studybyroi,bayes_model_regions);
disp('Apparent confusion')
disp(bayes_model_regions.apparent.confusion_mtx)

bayes_model_regions.xval = classify_naive_bayes('xval', studybyroi, Xi);
disp('Cross-validated confusion')
disp(bayes_model_regions.xval.confusion_mtx)

fprintf('Proportion correct: Apparent: %3.0f%%  Xval: %3.0f%%\n', 100*bayes_model_regions.appare
fprintf('Proportion correct by class: \t'); fprintf('%3.0f%%\t', 100*bayes_model_regions.xval.pr
fprintf('\n');

sz = cat(1,cl{1}(:).numVox);
cl{1}(sz < 10) = [];

subcl = subclusters_from_local_max(cl{1}, 10);
```

Statistics_tools.**classify_naive_bayes_objfun**(*dat*, *Xi*, *volInfo*, *a*, *s*, *g*, *k*, *t*, *h*)

Usage

```
goodness = classify_naive_bayes_objfun(dat, Xi, volInfo, 0, .9, 1, .05, .1, .1);
```

Examples

```
objfun = @(t, h) classify_naive_bayes_objfun(dat, Xi, volInfo, 0, .9, 1, .05, t, h);
goodness = objfun(.1, .1)
```

Statistics_tools.**classify_viz_regions**(*indx*, *colors*, *ptaskcutoff*, *sizecutoff*, *maskimage*, *names*)

Visualize output of classify_bayes.m, or comparable output

Usage

```
cl = classify_viz_regions(indx,colors,ptaskcutoff,sizecutoff,maskimage,names)
```

indx should be voxels x images, with prob task (ptask) in non-zero elements

Examples

```
cl = classify_viz_regions(indx,[],.6,5);
cl = classify_viz_regions(indx,[],.6,5,[],names);
```

Statistics_tools.**contrast_code**(*vec*)

Usage

```
[vec,outnames] = contrast_code(vec)
```

Changes values to 1, -1, or 0 for contrast coding

Statistics_tools.**correl_compare_dep**(*y1*, *y2*, *varargin*)
Compare dependent correlations between pairs of vectors in y1 and y2.

Usage

```
out = correl_compare_dep(y1,y2,['alpha',myalpha],['rank'],['table'])
```

Each of y1 and y2 would contain at least 2 columns, which would be correlated and saved in r1 and r2 matrices in output. Then, the r1 and r2 matrices are subtracted, and P-values are returned for the differences.

- In the simplest case, y1 would contain vectors [a b] and y2 would contain vectors [a c]. tests are provided on the a-b vs. a-c difference in correlations.

Repeats dep. correl. analysis for each pair of columns

Returns results in correlation matrix form, where number of rows and cols. are the number of pairs [y1(:,i) y2(:,i)]

myalpha is 2-tailed alpha value; p-values are 2-tailed FDR correction is at .01, 2-tailed

Based on Steiger, 1980, tests for comparing dependent correlations.

Examples

```
for i = 1:length(cl), y1(:,i) = cl.CONTRAST.data(:,2); y2(:,i) = cl.CONTRAST.data(:,1); end
for i = 1:length(cl), y1(:,i) = cl(i).CONTRAST.data(:,2); y2(:,i) = cl(i).CONTRAST.data(:,1); en

% y1 is matrix of obs x data vectors for condition 1
% y2 is matrix of obs x data vectors for condition 2
out = correl_compare_dep(y1,y2)

figure('Color','w');nmdsfig(c.GroupSpace,c.ClusterSolution.classes, ...
c.names,out.sig,1,{'Pos' 'Neg'});
nmdsfig_legend(c.ClusterSolution.X,c.r)
```

Examples

```
c_compare = correl_compare_dep(y1avg,y2avg,'alpha',.05,'rank','table','names',c.APPLY_CLUSTER.na

out = correl_compare_dep([ypred pain],[ypred temp], 'alpha', .06, 'table', 'names', {'biomarker
```

Statistics_tools.**correl_compare_dep_permtest**(*y1*, *y2*, *varargin*)
Compare dependent correlations between pairs of vectors in y1 and y2

Usage

```
out = correl_compare_dep_permtest(y1,y2,['alpha',myalpha],['rank'],['table'])
```

PERMUTATION TEST for correl_compare_dep

Repeats dep. correl. analysis for each pair of columns Returns results in correlation matrix form, where number of rows and cols. are the number of pairs [y1(:,i) y2(:,i)]

myalpha is 2-tailed alpha value; p-values are 2-tailed FDR correction is at .01, 2-tailed

Based on Steiger, 1980, tests for comparing dependent correlations.

Examples

```
for i = 1:length(cl), y1(:,i) = cl.CONTRAST.data(:,2); y2(:,i) = cl.CONTRAST.data(:,1); end
for i = 1:length(cl), y1(:,i) = cl(i).CONTRAST.data(:,2); y2(:,i) = cl(i).CONTRAST.data(:,1); en
y1 is matrix of obs x data vectors for condition 1
y2 is matrix of obs x data vectors for condition 2
out = correl_compare_dep(y1,y2)

figure('Color','w');nmdsfig(c.GroupSpace,c.ClusterSolution.classes, ...
c.names,out.sig,1,{'Pos' 'Neg'});
nmdsfig_legend(c.ClusterSolution.X,c.r)

% compare correlations on cluster averages
c_compare = correl_compare_dep(y1avg,y2avg,'alpha',.05,'rank','table','names',c.APPLY_CLUSTER.na
```

Statistics_tools.**correl_compare_dep_search**(*seed1*, *seed2*, *y1*, *y2*, *varargin*)

Compare dependent correlations between seed1<->y1 and seed2<->y2

Usage

```
correl_compare_dep_search(seed1,seed2,y1,y2,['alpha',myalpha],['rank'],['mask',maskimage])
```

Repeats dep. correl. analysis for each pair of columns Returns results in correlation matrix form, where number of rows and cols. are the number of pairs [y1(:,i) y2(:,i)]

myalpha is 2-tailed alpha value; p-values are 2-tailed FDR correction is at .01, 2-tailed

Based on Steiger, 1980, tests for comparing dependent correlations.

Examples

```
for i = 1:length(cl), y1(:,i) = cl.CONTRAST.data(:,2); y2(:,i) = cl.CONTRAST.data(:,1); end
for i = 1:length(cl), y1(:,i) = cl(i).CONTRAST.data(:,2); y2(:,i) = cl(i).CONTRAST.data(:,1); en
y1 is matrix of obs x data vectors for condition 1
y2 is matrix of obs x data vectors for condition 2
out = correl_compare_dep(y1,y2)

figure('Color','w');nmdsfig(c.GroupSpace,c.ClusterSolution.classes, ...
c.names,out.sig,1,{'Pos' 'Neg'});
nmdsfig_legend(c.ClusterSolution.X,c.r)

% compare correlations on cluster averages
c_compare = correl_compare_dep(y1avg,y2avg,'alpha',.05,'rank','table','names',c.APPLY_CLUSTER.na

% get image names
EXPT.subjects = {'ambar_carvalho' 'andreas_nguyen' 'angela_valle' 'brad_wilson' 'dominic_ricci'}
EXPT = getfunctnames2(EXPT,'con_0004.img','tmp','SPM_analysis/physical_pain/full_model_gv_p_v_np
self = str2mat(EXPT.tmp{:})

EXPT = getfunctnames2(EXPT,'con_0003.img','tmp','SPM_analysis/Videos/event_pain_only')
other = str2mat(EXPT.tmp{:})

maskimage = which('scalped_avg152T1_graymatter_smoothed.img')

% get seeds
```

```
    cd Jamil/GROUP_ANALYSES/Overlaps/Overlap_29_Aug/
    cl = mask2clusters('con_0002.img');
    cl = cl(4)
    cl = extract_contrast_data([{self} {other}],cl);
    seedself = cl(1).CONTRAST.data(:,1);
    seedother = cl(1).CONTRAST.data(:,2);

    correl_compare_dep_search(seedself,seedother,self,other,'alpha',.005,'mask',mask);

    % RESULTS:
    cl = mask2clusters('Correl_seed1_sig.img'); cluster_orthviews(cl,'bivalent');

    % Try the whole thing on ranks:
    correl_compare_dep_search(seedself,seedother,self,other,'alpha',.005,'mask',mask,'rank');
```

Statistics_tools.**correl_compare_indep**(*y_g1*, *y_g2*, *varargin*)

Compute statistics on the difference between correlation coefficients for indepndent samples g1 and g2.

Usage

```
    [r1, r2, rdiff, rdiff_p, rdiff_zscore] = correl_compare_indep(y_g1, y_g2, varargin)
```

Correlations are computed over y (n observations x k variables) for each group.

Source, Hubert Blalock, Social Statistics, NY: McGraw-Hill, 1972: 406-407. From notes on Garson stats website.

Inputs

y_g1: Matrix of variables for group 1 (n observations x k variables)

y_g2: Matrix of variables for group 2 (n observations x k variables)

Optional Inputs

varargin: String 'noverbose' can be used to prevent output from being printed to command line

Outputs

r1: Matrix of correlation coefficients for group 1

r2: Matrix of correlation coefficients for group 2

rdiff: Difference in correlation coefficient matrices (r1 - r2)

rdiff_p: p values testing against no difference between r1 and r2. Hubert Blalock, Social Statistics, NY: McGraw-Hill, 1972: 406-407.

rdiff_zscore: z scores for the difference between r1 and r2.

Examples

```
    %generate two random input matrices (20 subjects x 5 variables)
    y_g1 = rand(20,5);
    y_g2 = rand(20,5);
    [r1, r2, rdiff, rdiff_p, rdiff_zscore] = correl_compare_indep(y_g1,y_g2);

    %supress output
    [r1, r2, rdiff, rdiff_p, rdiff_zscore] = correl_compare_indep(y_g1,y_g2,'noverbose');
```

See also

- correl_compare_indep_inputr, correl_compare_permute*

Statistics_tools.**correl_compare_indep_inputr**(*r1*, *r2*, *N1*, *N2*, *varargin*)

Compare two Pearson's correlation values collected from independent samples :Based on: www.stat-help.com/ :Usage:

```
[rdiff, Z, pval, stats] = correl_compare_indep_inputr(r1, r2, N1, N2, varargin)
```

Inputs

r1: Correlation coefficient for group 1

r2: Correlation coefficient for group 2

N1: Sample size for group 1

N2: Sample size for group 2

Outputs

rdiff: Difference in correlation coefficient matrices (r1 - r2)

Z: Z score for difference in correlation coefficients.

pval: p value for two-tailed z-test

stats: Structure of variables including r1, r2, N1, N2, rdiff, zdiff, Z, pval, myalpha, sig

Examples

```
 %generate two random input matrices (20 subjects x 5 variables)
 r1 = .77;
 r2 = .33;
 N1 = 100;
 N2 = 100;

[rdiff, Z, pval, stats] = correl_compare_indep_inputr(r1, r2, N1, N2);
```

See also

- correl_compare_indep_inputr, correl_compare_permute*

Statistics_tools.**correl_compare_permute**(*meth*, *dat*, *nperms*, *condition*)

General function for comparing correlation matrices on two different sets of observations

Usage

:: correl_compare_permute(meth,dat,nperms,condition)

dat is obs x variables, and an n x n association matrix will be computed on the n(n - 1) pairs of columns using one of the methods in correlation.m and specified by meth

Inputs

meth: String indicating the method for computing the correlation coefficient (e.g., 'taub') see correlation.m for details

dat: Matrix of observations (n instances by p variables)

nperms: Number of permutations for inferential tests

condition: Vector indicating condition membership for each instance (currently only works for two conditions)

Outputs

OUT: Output stats structure

See also

- correl_compare_indep_inputr, correl_compare_indep, correlation*

Statistics_tools.**correlation**(*meth*, *x*, *varargin*)

Multiple types of correlations, including Spearman's rho (nonparametric) and phi (dichotomous)

Usage

```
[corr,t,p,fdrp, fdrthresh] = correlation(method,x,[y],['matrix'])

IN PROGRESS : Warning : Use at your own risk.
Some methods are not adequately tested yet.
Spearman's rho does not correct for ties
```

Inputs

Methods: String indicating the method for computing the correlation coefficient - Pearson's r. Enter: {'r','pearson',[]} - IRLS Enter: {'irls','robust'} - Phi Enter: {'phi'} - Spearman's rho Enter: {'rho','spearman'} - Kendall's Tau (a) Enter: {'taua','kendalla'} - Tau (b) Enter: {'tau','kendall','taub','kendallb'} - Gamma Enter: {'gamma','kruskal'}

x: Matrix of observations (n instances by p varianbles)

Optional Inputs

varargin: To be documented

Outputs

OUT: Output stats structure

Examples

```
% Corelation between two variables, Pearson's
x = rand(10,1); y = rand(10,1);
[corr,t,p] = correlation('r',x,y);

% Correlation matrix of 10 variables, phi correlation:
studybyroi = magic(10);
[corr,t,p] = correlation('phi',studybyroi);
```

See Also correlation_fast_series.m

Statistics_tools.**correlation_fast_series**(*Xi*, *Yi*)

Usage

```
[r, p, t] = correlation_fast_series(Xi, Yi)
```

Fast, memory-efficient way to get correlation between each column of Xi and Yi

Inputs

Xi: Matrix of observations (n instances by p variables)

Yi: Matrix of observations (n instances by p variables)

Outputs

r: Pearson correlation coefficients

p: Corresponding p-value for correlation coeffieints

Tstat: T statistic for comparing correlation coefficients against 0

Examples

```
r = ((Xi' * Yi) ./ N) ./ sqrt(diag(Xi' * Xi) ./ N * (Yi' * Yi) ./ N);
[r2, p, Tstat] = correlation_fast_series(Xi, Yi);
[r3, p2] = corrcoef([Yi, Xi]); r3 = r3(1, 2:end); p2 = p2(1, 2:end);
create_figure('test', 1, 3); plot(r3, r, 'kx'); axis equal; grid on; subplot(1, 3, 2); plot(r3,
```

Statistics_tools.**create_orthogonal_contrast_set**(*nconditions*)

Create an orthogonal contrast set across nconditions conditions

Usage

```
create_orthogonal_contrast_set(nconditions)
```

Contrasts sum to zero, are orthogonal, and positive values and negative values each sum to 1.

This is useful in evaluating designs, and for F-tests across all pairwise differences (e.g., one-way ANOVA contrast).

Statistics_tools.**dice_coeff_image**(*imagelist*)

dice coefficient for the overlap between each pair of a set of images

Usage

```
dice_coeff = dice_coeff_image([imagelist or fmri_data object])
```

tor wager, June 2010 ..

Statistics_tools.**doquality**(*Xcx*, *X*)

Usage

```
[equality] = doquality(Xcx,X)
```

Inputs

Xcx: binary indicator matrix of cluster assignments,

X: stimulus coordinates in group space

also: takes group spaces with zeros;

Statistics_tools.**fisherz**(*r*)

Fisher's r to z' transform, and the inverse

Outputs

z: z = z', treating input r as correlation

r: treating input r as a z' score

Statistics_tools.**fit_gls**(*y*, *X*, *c*, *p*, *varargin*)

Fit a linear model using generalized least squares and an AR(p) model

Usage

```
[beta, t, pvals, convals, con_t, con_pvals, sigma, Phi, df, stebeta, conste, F] = fit_gls(y,X,c,
```

This program uses the Cochrane-Orcutt algorithm to iteratively find the GLS solution and estimate the noise parameters.

Step 1: Find the OLS solution.

Step 2: Use residuals from the previous fit to estimate the parameters in the AR(p) noise model.

Step 3: Find the GLS solution using the covariance matrix corresponding to an AR(p) model with parameters estimated in Step 2 inserted.

Step 4: Repeat steps 2-3 until convergence.

Inputs

y: fMRI time course (T x 1 vector)

X: Design matrix (T x param matrix)

c: contrast vector(s) (param x # contrasts matrix)

p: order of AR model.

PX: pinv(X), for speeded, repeated calculations with different y vectors

Note: if using weights, px = inv(X' * W * X) * X' * W; where W = diag(Weights);

Weights: Optional, vector of weights for each observation

Empty or missing: Unweighted analysis.

Note that setting p=0 implies a white noise model.

Output

t: t-value for the contrast c'beta

df: degrees of freedom using Satterthwaite approximation

beta: beta vector

Phi: vector of coefficients in AR(p) model

sigma: standard deviation

stebeta: standard error of betas

Statistics_tools.**fit_gls_brain**(*imgs*, *X*, *arorder*, *conditionnames*, *maskimg*, *varargin*)

Run Generalized Least Squares model with AR-model autoregression (optional) at each voxel in a set of images.

Usage

```
fit_gls_brain(imgs, X, arorder, conditionnames, maskimg, ['contrasts', contrast_mtx, 'contrastna
```

Single trial model to get trial amplitude, width, AUC, etc. Take one of those (e.g., AUC) and get a list of trial images Pass that into this function.

fit_gls_brain(trial_amp_imgs, eventdesign{1}, contrasts

You must add the intercept to X yourself if you want one!

contrasts can be empty, and if it is, this function will write images for statistics on individual predictors. If you enter contrast values, then it will write images for contrasts instead.

conditionnames = column or contrast image names, in cell array

e.g. eventnames{1} = {'high' 'medium' 'low' 'warm'}

Examples

This one looks @ significance for betas in X model:

```
load ../Multilev_mediation-try4(resliced)_10k/mediation_SETUP.mat
imgs = SETUP.data.M{1};
X = eventdesign{1};
arorder = 1;
contrasts = [];
conditionnames = eventnames{1};
maskimg = spm_select(1); % try gray matter mask...
fit_gls_brain(imgs, X, arorder, contrasts, conditionnames, maskimg)
```

Now define contrasts and re-run on contrast values:

```
contrasts = [3 1 -1 -3; .25 .25 .25 .25; 1 0 0 -1]'
conditionnames = {'Linearpain' 'Average_resp' 'High-Low'};
fit_gls_brain(imgs, X, arorder, contrasts, conditionnames, maskimg)
```

Statistics_tools.**getmeanquality**(*X*, *c*, *linkagetype*)
point estimate cluster solution

Statistics_tools.**glmfit_general**(*Y*, *X*, *varargin*)
This function was designed to run a second-level GLS analysis on coefficients and variance estimates from a 1st-level model, but it may be used generally to implement weighted GLS with options to bootstrap or run a sign permutation test to get p-values.

Inputs

Y: data for n subjects on k variables

run tests on each column of y independently

if this is a 2nd-level analysis in a two-level model, Y is coefficients from 1st level (mu-hat)

s2: estimates of variance (sigma-squared??) for k variables

X: design matrix; include intercept (first) + predictors

Estimation method

- 'unweighted' / 'weighted'
- 'robust' / 'nonrobust'
- 's2'

NEW: followed by n-length cell array of var/cov mtx (xtxi*sigma2) estimates from previous level

OLD: followed by n x k matrix of first-level variance estimates

Inference method

- 't-test'
- 'bootstrap'
- 'signperm'

Optional Inputs

'names': names of each col. of y: string followed by input

**'name' : text string tag for analysis

'verbose': print verbose output and tables

'nresample': number of boot/sign perm samples to run; string followed by input

'noint': do not require intercept to be first column of X.

Notes

Same as glmfit.m (2007a) without weights.'

With weights, we estimate w and use Satterthwaite, whereas glmfit uses n - k for dfe (assumes correct weights are known.)

Outputs

stats.b_star: Weighted fixed effects estimate (if using 'weighted' option)

stats.Y_star: Y_star'; % Empirical Bayes estimates; don't use for group inference, but save;

if this is run in a context of a multi-level model, these are the individual subjects' (first-level) Empirical Bayes slope estimates

Output to screen

Coeff is "beta", the effect magnitude STE is the standard error, a function of variance and df

- T = beta / STE
- Z = beta / the STE for infinite df
- p

Examples

```
Y = randn(100, 4); X = Y(:,1) + randn(100,1); X = [ones(size(X)) X];
stats = glmfit_general(Y, X, 'verbose');
stats
first_lev_var = rand(100, 4);

stats = glmfit_general(Y, X, 'weighted', first_lev_var, 'dfwithin', 20, ...
'verbose', 'names', {'DMPFC' 'ACC' 'VMPFC' 'SII'}, ...
'beta_names', {'Mean activity' 'Rating covariate'}, ...
'analysisname','My Sample ROI analysis');
```

Statistics_tools.**glmfit_multilevel**(*Y*, *X1*, *X2*, *varargin*)

Usage

```
stats = glmfit_multilevel(Y, X1, X2, varargin)
```

Inputs

Y: is data in cell array, one cell per subject. Column vector of subject outcome data in each cell.

X1 and X2:

are first and 2nd level design matrices

- X1 in cell array, one cell per subject design matrix for each subject in each cell. *columns must code for the same variable for all subjects*
- X2 in rect. matrix
- can be empty (intercept only)

E.g., with one 2nd-level predictor: stats = glmfit_multilevel(Y, X1, X2, ... 'names', {'Int' 'Temp'}, 'beta_names', {'2nd-level Intercept (overall group effect)' '2nd-lev predictor: Group membership'});

Output

stats: is structure with results. Intercept is always added as first column! (do not add intercept to input predictors)

See glmfit_general.m for varargin variable input options.

Examples

```
len = 200; sub = 20;
x = zeros(len,sub);
x(11:20,:) = 2;                    % create signal
x(111:120,:) = 2;
c = normrnd(0.5,0.1,sub,1);        % slope between-subjects variations
d = normrnd(3,0.2,sub,1);          % intercept between-subjects variations
% Create y: Add between-subjects error (random effects) and measurement noise
% (within-subjects error)
for i=1:sub, y(:,i) = d(i) + c(i).*x(:,i) + normrnd(0,0.5,len,1);
end;

for i = 1:size(y, 2), YY{i} = y(:, i); end
for i = 1:size(y, 2), XX{i} = x(:, i); end

% one-sample t-test, weighted by inv of btwn + within vars
stats = glmfit_multilevel(YY, XX, [], 'verbose', 'weighted');

statsg = glmfit_multilevel(y, x, covti, 'names', {'L1 Intercept' 'L1 Slope'},...
'beta_names', {'Group Average', 'L2_Covt'});
```

Input Options

General Defaults

- case 'names', Names of first-level predictors, starting with 'Intercept', in cell array
- case 'analysisname', analysisname = varargin{i+1}; varargin{i+1} = [];
- case 'beta_names', beta_names = Names of 2nd-level predictors, starting with 'Intercept', in cell array

Estimation Defaults

- case 'robust', robust_option = 'yes';
- case {'weight', 'weighted', 'var', 's2'}, weight_option = 'weighted';

Inference defaults

- case {'boot1', 'boot', 'bootstrap'}, inference_option = 'bootstrap';
- case {'sign perm', 'signperm', 'sign'}, inference_option = 'signperm';
- case {'t-test', 'ttest'}, inference_option = 't-test';

Display control defaults

- case 'plots', doplots = 1; plotstr = 'plots';
- case 'noplots', doplots = 0; plotstr = 'noplots';
- case {'dosave', 'save', 'saveplots'}, dosave = 1; savestr = 'save';
- case 'verbose', verbose = 1; verbstr = 'verbose';
- case 'noverbose', verbose = 0; verbstr = 'noverbose';

- case {'savefile', 'savefilename'}, savefilename = varargin{i + 1}; varargin{i+1} = [];

Bootstrap defaults

- case 'nresample', nresample = varargin{i+1};
- case {'pvals', 'whpvals_for_boot'}, whpvals_for_boot = varargin{i+1};

Sign perm defaults

- case {'permsign'}, permsign = varargin{i+1};

Statistics_tools.**glmfit_multilevel_varexplained**(*X*, *Y*, *b*, *varargin*)

Variance explained estimates, table, and pie chart for multi-level linear model

Usage

```
vardecomp =  glmfit_multilevel_varexplained(XX,YY, b, ['noplots' 'notable'])
```

Inputs

b: is a column vector of beta weights for fixed effects (i.e., returned by glmfit_multilevel: stats.beta')

X: is a cell vector containing the design matrix for each subject, each subject in a cell. *columns must code for the same variable for all subjects*

Y: is the data vector for each subject in each cell

Optional Inputs

'noplots', 'notable': suppress output

'colors': followed by n x 3 vector of rgb colors other inputs to wani_pie OK too.

Output

vardecomp: structure of variance explained and description

See also

glmfit_multilevel, igls_multicond, glmfit_general, wani_pie

Statistics_tools.**intercept**(*X*, *meth*)

Intercept-related functions for working with design matrices, etc.

Return which columns are intercept(s), if any

```
wh = intercept(X, 'which')
```

Remove an intercept, if there is one

```
X = intercept(X, 'remove');
```

Add an intercept to the end

```
X = intercept(X, 'add');
```

Ensure that the intercept is at the end, moving or adding it as necessary

```
X = intercept(X, 'end');
```

Statistics_tools.**loess_multilevel**(*X*, *Y*, *varargin*)

Usage

```
stats = loess_multilevel(X, Y, varargin)
```

Options:

- case 'fit_and_average', meth = 'fit_and_average';
- case 'regularization', regularization = varargin{i+1};
- case 'order', loessorder = varargin{i+1};
- case {'robust', 'dorobust'}, dorobust = varargin{i+1};
- case {'nboot', 'bootsamples'}, nboot = varargin{i+1};
- case 'plot', doplot = 1;
- case {'plotall'}, plotsummary = 0; doplot = 1;
- case 'color', color = varargin{i+1};
- case {'existingfig', 'samefig'}, newfig = 0;

Example

```
stats = loess_multilevel(models{i}.X, models{i}.Y, 'regularization', .8, 'order', 1, 'color', [.
```

See Also scn_stats_helper_functions

Statistics_tools.**matrix_direct_effects**(*sig*, *data*)

Usage

```
[direct_mtx, mediated_mtx, mediators] = matrix_direct_effects(sig, data);
```

Take an n x n matrix of significant correlations and the data for each variable. Test whether each link is completely mediated by another variable. Prune the correlation significance matrix, also noting which variables are mediators.

Example: Take output from cluster_nmds (c structure) and prune to return direct links only:

```
[c.direct_mtx, c.mediated_mtx, c.mediators] = matrix_direct_effects(c.STATS.sigmat, c.dat);
nmdsfig_tools('removelines');
nmdsfig_tools('drawlines',c.GroupSpace, c.direct_mtx);
```

Statistics_tools.**matrix_direct_effects_ridge**(*data*, *varargin*)

Usage

```
[out] = matrix_direct_effects_ridge(data, [optional args]);
```

Take an n x v matrix of data for each of N subjects. data are entered in a square matrix within each cell of data, data{1}, {2}, etc.

Uses ridge regression to assess linear slopes for each variable on each other one.

Optional Inputs

'k': followed by ridge parameter value. Default = 0.

'scale' or 'zscore': which z-scores within-subjects

Future: Consider weighting based on multicolinearity

Note: Matrix is asymmetrical! Rows predicting columns...

Examples

Take mediation brain results clusters cell clpos_data{...} Concatenate data matrix and run to get connections.

```
for i = 1:length(clpos_data), data{i} = cat(2, clpos_data{i}.timeseries); end
out = matrix_direct_effects_ridge(data);
[out.GroupSpace,out.obs_dist,out.implied_dissim] = shepardplot(out.dissim,[]);
create_figure('nmdsfig');
nmdsfig(out.GroupSpace,'classes',ones(out.k, 1),'names',[],'sig',out.fdrsig, 'legend',{'Pos' 'Ne
```

Example 2: Include mediation predictor and outcome

```
for i = 1:length(clpos_data), data{i} = [SETUP.data.X{i} SETUP.data.Y{i} data{i}]; end
out = matrix_direct_effects_ridge(data);
```

Example of MDS and plotting direct relationships:

```
OUT.ridge = matrix_direct_effects_ridge(data);
D = OUT.ridge.mean; D(find(eye(size(D)))) = 1;
D = (D' + D) ./ 2;
OUT.ridge.D = (1 - D) ./ 2;
[OUT.stats_mds.GroupSpace,OUT.stats_mds.obs,OUT.stats_mds.implied_dissim] = shepardplot(OUT.ridg
OUT.stats_mds = nmdsfig_tools('cluster_solution',OUT.stats_mds, OUT.stats_mds.GroupSpace, 2:10,
OUT.stats_mds.colors = {'ro' 'go' 'bo' 'yo' 'co' 'mo' 'ko' 'r^' 'g^' 'b^' 'y^' 'c^' 'm^' 'k^'};
create_figure('nmdsfig');
nmdsfig(OUT.stats_mds.GroupSpace,'classes',OUT.stats_mds.ClusterSolution.classes,'names',OUT.sta
hh = nmdsfig_fill(OUT.stats_mds);
```

Prediction using multiple regions:

```
stats = rsquare_multiple_regions_multilevel(Y, X, varargin)
```

Visualization of networks:

```
classes = OUT.stats_mds.ClusterSolution.classes;
% create_figure('Surfaces');
% shan = addbrain;
shan = mediation_brain_surface_figs({}, {});
scolors = {[1 0 0] [0 1 0] [0 0 1] [1 1 0] [0 1 1] [1 0 1] [0 0 0] [1 0 0] [0 1 0] [0 0 1] [1 1
for j = 1:max(classes)
wh = (classes == j); sum(wh)
montage_clusters([], cl(wh), scolors(j))
% create_figure('Surfaces', 1, 1, 1);
cluster_surf(cl(wh), 5, scolors(j), shan);
end
```

See Also xcorr_multisubject.m

Statistics_tools.**matrix_eval_function**(*Y*, *fhandle*, *varargin*)

Evaluate any arbitrary function fhandle on each column of an input matrix Y

Usage

```
varargout = matrix_eval_function(Y,fhandle,varargin)
```

evaluate fhandle on paired columns of X and Y

fhandle is a function handle:

```
fhandle = @(variable inputs) fit_gls(variable_input,fixed_inputs);
fhandle = @(y) fit_gls(y,X,c,p,PX);
```

Function is fhandle should return (multiple possible) outputs, each in a row vector. Matrix outputs of this function are Y-cols x output values for each output argument.

Example Generalized least squares fitting on 100 Y-variables, same X

```
y = rand(100,100); X = y + rand(100,1); X(:,end+1) = 1; c = [1 0]'; p = 2; PX = pinv(X);
fhandle = @(y) fit_gls(y,X,c,p,PX);
[t, df, beta, Phi, sigma,stebeta, F] = fhandle(y);
```

Statistics_tools.**monotonic_regression**(*x*, *y*, *doplot*)
See lsqisotonic.m

Statistics_tools.**moving_average**(*meth*, *data*, *varargin*)
Symmetrical moving average filters (matlab's internal moving avg filter functions are asymmetric)

Works on each column of the data input matrix

Examples

```
y = moving_average('gaussian',data,fwhm)

% fwhm is not actually fhwm now...it's related to width though (temporary)
y = moving_average('gaussian',data,20);
```

Statistics_tools.**noise_arp**(*n*, *phi*, *sigma*, *e*)

Usage

```
w = noise_arp(n, phi)
```

Generate n-length AR(p) noise vector w given phi (default = [.5 .1])

Based on Gaussian noise process e with standard deviation sigma (default = 1)

w will have a variance greater than that of the underlyling gaussian process e

Statistics_tools.**nonlin_fit**(*y*, *x*, *varargin*)
Multi-purpose nonlinear fitting to data

Usage

```
[p,errval,fit] = nonlin_fit(y,x,['link',linktype],['err',objtype],['start',startp], ...
                            ['plot'],['verbose'],[noverbose'],['quickstart'],['smartstart'])
```

(Optional arguments are in [ ])

Inputs

y: data vector

x: vector of x-values

linktype: functional form to fit to data. options are:

'sigmoid' 'exp0' 'exp2' 'exp3' 'exps'

objtype: error measures to be minimized. options are:

'sse' : sums of squared errors

'mad' : median absolute deviation (more robust to outliers)

'plot': Create plot of data and fit

startp: Vector of starting parameters for minimization

Functions 'sigmoid':

'exp1', 'exp2', 'exp3':

Exponential function

- 'exp1' y = e^(ax) with free parameter a
- 'exp2' y = b*e^(ax) with free a, b
- 'exp3' y = c + b*e^(ax) with free a, b, c

'pow0' 'pow2' 'pow3' 'pows1' 'pows2' 'pows':

Power functions:

- **'pows' a + -(bx)^(-c) with params a, b, c** simplifies to 1 - (1/x) with all p = 1 range: (for + b and c) -Inf at x = 0, to asymptote = a at x = Inf.

Usage

Default behavior: sigmoid fit, SSE objective:

```
[p,sse,fit] = nonlin_fit(y,x,'plot');
```

Specify sigmoid function and SSE objective, starting at params [2 1 1]:

```
[p,sse,fit] = nonlin_fit(y,x,'start',[2 1 1],'link','sigmoid','err','sse');
```

Specify median absolute deviation error minimization (more robust):

```
[p,sse,fit] = nonlin_fit(y,x,'err','mad','plot');
```

Fit and add a fit line on a graph

```
[p,errval,fit,linkfun,fhan] = nonlin_fit(y,x,'linktype','exps','start',[1.2 .5 .8]);
fitx = 1:.1:5; fitline = fhan(p,fitx);
hold on; plot(fitx,fitline,'r');
```

Examples

```
% sigmoid fit to y
% Generate fake data:
x = -5:.01:5;
sigmoid = inline('p(1) .* ( 1 ./ (1 + p(2)*exp(-p(3)*x)) )','p','x');
y = sigmoid([2 1 1.5],x);
y = y + .3 .* randn(1,size(y,2)) ;

% Fit, starting at param values [1 1 1]:
[p,sse,fit] = nonlin_fit(y,x,'plot','start',[1 1 1]);

% Bit more complicated data
x2 = -5:.01:5;
y = sigmoid([2 1 .5],x2);
y = [y 2*ones(1,100)];
y = y + .3 .* randn(1,size(y,2)) ;
x = 1:length(y);

% Exponential function (see also exp1, exp2, exp3 keywords)
expfun = inline('p(1)*exp(p(2)*x)','p','x');

% simulated sample data
x = -1:.01:5;
y = expfun([.5 .7], x);
```

```
    y = y + 1 .* randn(1,size(y,2));

    % function handle to pass in
    funhandle = @(p, x) expfun(p, x);
    [p,sse,fit] = nonlin_fit(y,x,'linktype', funhandle, 'plot','start',[1 1]);

    % Then, try to fit one-parameter exponential model to the same data:
    [p,sse,fit] = nonlin_fit(y,x,'linktype', 'exp1', 'plot','start', 1);
```

Statistics_tools.**nonlin_param_mod_brain**(*ons*, *modulator*, *image_names*, *SETUP*, *varargin*)

Nonlinear fits with a parametric modulator on a set of brain images

Usage

```
    nonlin_param_mod_brain(X, image_names, SETUP, [SETUPional inputs])
```

Inputs

ons: onsets for each condition; one cell per condition, one col. vector per series of onsets

modulator: modulator values for each condition; same format as above

image_names: outcome variable; Images (volume names) for each subject, in string matrix (list of image names); 3-D for now!

SETUP.(fields)

.mask: name of mask image

.preprocX: flag for whether to HP filter X data

.preprocY: flag for whether to HP filter Y data

'nopreproc': Turn off preproc

.TR: repetition time of volume (image) acquisition

.HPlength: high-pass filter length, in s

.scans_per_session: vector of # volumes in each run, e.g., [128 128 128 128 128]

.dummyscans: indices of images in each run that will be modeled with separate dummy variables

.startslice: starting slice number (to resume analysis)

SETUPional inputs: Any of the SETUPions in mediation.m

Also: 'nopreproc' to skip preprocessing (i.e., for trial-level inputs)

Statistics_tools.**nonlin_param_modulator**(*y*, *ons*, *modulator_centered*, *tr*, *xvals*)

Usage

```
    [h_mean h_by_mod d_mean d_by_mod intcpt auc_mean_trial auc_by_modulator errval] =  nonlin_param_
```

USE THIS with nonlin_param_mod_brain.m

This function takes data (y) and other things (onsets, etc.) and produces parameter estimates for amplitude, duration, amp*modulator, dur*modulator

See rt_fit_brain.m for more info, and for examples creating fit plots, etc.

Examples

```
      y = rand(360, 1);
      [h_mean h_by_mod d_mean d_by_mod intcpt auc_mean_trial auc_by_modulator errval] =  nonlin_para
      nonlin_parammod_predfun(y, ons, scale(RTs), [h_mean h_by_mod d_mean d_by_mod intcpt], 'plot');
      hold on; plot(y, 'k')
      % THIS stuff is the same for each voxel
     RTcenter = RTs - mean(RTs);
     xvals = (1:length(y))';

    % fitting function: times is a dummy var to get this to work with nonlin_fit
     fhan = @(p, times) nonlin_parammod_predfun(y,ons,RTcenter,p);

     fitting_fun = @(y) nonlin_fit(y, xvals, 'link', fhan, 'start',[1 1 1 1 mean(y)]);
```

Statistics_tools.**nonlin_parammod_predfun**(*y*, *ons*, *pm_vals*, *p*, *varargin*)

Predict data given onsets, parametric modulator values, and parameter estimates for effects of events and PMs on the magnitude and delay of an estimated neural 'boxcar.'

A standard HRF model is assumed, and the parameters are estimates of how trials affect the 'neural' stimulus function.

This function replaces nonlin_parammod_predfun.m, which is obsolete

Usage

```
[yhat,yhi,loTime,highTime,x] = nonlin_parammod_predfun(y,ons,pm_vals,p,varargin)
```

Inputs

p:

params

1. mag. scale (amplitude)
2. mag. X RT slope (linear effect of RT)
3. duration intercept (neural epoch duration)
4. duration x RT slope (linear effect of RT on duration)
5. overall fitted response intercept

y: data. Can be dummy data, needs only be the correct size This function does not actually fit the data, so y is only used to get the correct vector size

ons: onsets (in samples)

pm_vals: modulator values, preferably centered

p: parameter values; see below

varargin: optional inputs, see below

Special cases of parameter sets

p = [1 0 1 0 0]; % "Impulse model" : fixed boxcar of 1 s

p = [1 0 3 0 0]; % "Epoch model" : fixed boxcar of 3 s

p = [1 1 1 0 0]; % "Parametric modulator" : Impulse height modulated by RT (with centered pm_vals)

p = [1 0 0 1 0]; % "Variable epoch": convolve RT with duration (with non-centered, raw pm_vals). Parametric modulation of duration

p = [1 1 1 1 0]; % combo of Parametric and Duration modulators of impulses

p = [1 1.5 3 1.5 0]; % combo of Parametric and Duration modulators of a 3-s epoch model

event signal magnitude = p1 * p2*RT % was previously: RT^p2 event signal duration = p3 + pm_vals * p4

Optional Inputs

- case 'random', dorandom = 1;
- case 'plot', doplot = 1;
- case 'hrf', hrf = varargin{i + 1}; % HIGH_RES HRF
- case 'tr', followed by TR

Examples of generating predicted BOLD timeseries

```
y = rand(330,1);
ons = [1:20:320]'; pm_vals = rand(size(ons));
[yhat,yhi] = nonlin_parammod_predfun(y,ons,pm_vals,[1 1 1 0 0],'plot');  % linear RT modulation
[yhat,yhi] = nonlin_parammod_predfun(y,ons,pm_vals,[1 0 1 10 0],'plot');  % linear RT modulation

[yhat,yhi] = nonlin_parammod_predfun(y,ons,pm_vals,[.5 1 1 0 0],'plot');  % other linear RT modu
[yhat,yhi] = nonlin_parammod_predfun(y,ons,pm_vals,[.5 .7 1 5 0],'plot');  % saturated RT modula
```

Example of fitting observed timeseries and estimating parameters:

•SEE ALSO nonlin_param_modulator

```
RTcenter = pm_vals - mean(pm_vals);
true_p = [1 1.5 3 1.5 0];          % true parameters
xvals = (1:length(y))';
```

Fitting function: times is a dummy var to get this to work with nonlin_fit

```
fhan = @(p, times) nonlin_parammod_predfun(y,ons,RTcenter,p);

y = fhan(true_p);                  % generate simulated "true" signal

fitting_fun = @(y) nonlin_fit(y, xvals, 'link', fhan, 'start',[1 1 1 1 .5]);
[p, errval, fit] = fitting_fun(y)     % get parameters for a timeseries of interest

[p,errval,fit] = nonlin_fit(y,(1:length(y))','link',fhan,'start',[1 1 1 1 .5]);
hold on; plot(fit,'r');
```

A second example using the genetic algorithm:

```
objfun = @(p) sum(abs(y - fhan(p))); % objective function: absolute error
start = [-10 -10 0 -10 0]; start(:,:,2) = [10 10 10 10 2000];
[best_params,fit,beff,in] = tor_ga(200,50,{start},objfun,'genconverge',5,'noverbose');
%***note: does not work now, needs debugging***
```

Simulation: Test cov of param estimates

Run 1000 times to see cov. of param estimates Right now: ****development: p(1,2) are highly + corr, p(3:4) are high - corr should probably choose one or the other for each.

```
for i = 1:1000
   yi = y + randn(length(y),1) * 10; % new noise
   p(i,:) = fitting_fun(yi);
   if mod(i,10) == 0, fprintf(1,' %3.0f',i); end
end
```

Note: Could create linear basis set of plausible forms

Statistics_tools.**pairwise_diffs**(*x*, *varargin*)

Pairwise operations on columns of a matrix x, for each row

Usage

```
d = pairwise_diffs(x, [function handle])
```

Inputs

x: an n x k matrix

Optional: a function handle for the operation to perform on pairwise elements of x(i, :)

Outputs if x is n x k, d is n x (k*(k-1)/2)

columns of x are arranged this way, e.g., with k = 5:

[x(1, 1) - x(1, 4) x(1, 1) - x(1, 5) x(1, 2) - x(1, 3) ...]

squareform(d(1, :)) is a matrix of pairwise diffs for d

Examples

```
x = magic(5)                    % generate data
d = pairwise_diffs(x);          % d = pairwise differences
row1 = squareform(d(1, :));     % square matrix of pairwise diffs for row 1

d = pairwise_diffs(x, @(a, b) a + b);  % return pairwise sum instead
```

Statistics_tools.**partition_variables_indevel**(*x*, *k*, *obsk*)

partition n x v matrix into k classes

maximize within-class condition number or minimize cov. determinant

minimize between-class condition number or maximize cov. determinant...

Simulate two-class data

```
nvars = 100; nsubj = 20; corval = .5;
S = eye(nvars./2) + corval*(1 - eye(nvars./2));
S = blkdiag(S,S); S(S==0) = corval;
x = mvnrnd(zeros(1,nvars), S, nsubj); det((corrcoef(x)))
figure; imagesc(corrcoef(x)); colorbar
```

Statistics_tools.**permute_setupperms**(*n*, *nperms*)

Set up permutations x observations matrix of observation indices for permutation test

Usage

```
% approximate test with nperms obs.
permindx = permute_setupperms(n,nperms)
permindx = permute_setupperms(n,[])

% exact test (all permutations) of n observations
permindx = permute_setupperms(n)
```

Statistics_tools.**permute_signtest**(*data*, *nperms*, *w*, *permsign*)

One-sample t-test against zero on each column of data Using weighted sign permutation test

Usage

```
[p, Z, xbar, permsign, pmean] = permute_signtest(data, nperms, [w], [permsign])
```

p-values are 2-tailed

Inputs

data: n x k matrix of k vectors to test against zero

nperms: number of permutations

w: n x k matrix of weights

weights for each column should sum to 1

default (for empty or missing input) is equal weights, i.e., 1/n

permindx: nperms x n matrix of exact permutation indices default (for empty or missing input) generates nperms permutations

Example Generate 5-vector dataset and test, first generating perms, then using already-generated ones

```
tic, [p, z, xbar, permsign] = permute_signtest(x, 1000); toc
tic, [p2, z2, xbar, permsign] = permute_signtest(x, [], [], permsign); toc
```

Example: Generate fake data and simulate false positive rate

```
permsign = [];  % initialize to empty for first iteration
nperms = 2000; nreps = 5000;
tic, for i = 1:nreps
x = randn(20,1); stat = mean(x);
[p(i), z(i), xbar, permsign] = permute_signtest(x, nperms, [], permsign);
if mod(i,10) == 0,fprintf(1,'%03d ',i); end
end
fprintf(1,'\n');
```

Example: Generate fake data and simulate false positive rate This uses the column-wise capabilities of this function and is much faster %

```
nperms = 2000; nreps = 5000; nsubj = 15;
x = randn(nsubj, nreps);
[p, z, xbar, permsign] = permute_signtest(x, nperms);
```

Statistics_tools.**plssquash**(*X*, *Y*, *varargin*)

Decomposes data X into K components that are ordered in their covariance with Y, designed to predict orthogonal parts of Y

Usage

```
[V, S, varexp, w, Yhat] = plssquash(X, Y, varargin)
```

Optional Inputs

- case {'noplot'}, turn off plotting
- case 'ndims', save only first ndims (K) vectors

Outputs

V: 'eigenvetors', or weights, on data (columns)

S: score matrix, N x K

varexp: sqrt(r-square) with first k components predicting Y

w: V*b, integrated weights. for predicting new data, pred = X*w

Yhat: X*V*b, or S*b

Statistics_tools.**princomp_largedata**(*x*, *econFlag*)
Principal Components Analysis

This is a version created for large data sets by Matthew Davidson

The default Matlab PRINCOMP is naive to large data sets. Out-of-memory errors are easily obtained on imaging data. The solution is to replace concise but inefficient calls to repmat with loops and to eliminate large, unused variables.

Tested on random data sets and produces identical output as original PRINCOMP. Speed penalty is drastic for small data. 50% slower on 50x1000 element data set, but on a 50x10000, only 2% slower.

COEFF = PRINCOMP(X) performs principal components analysis on the N-by-P data matrix X, and returns the principal component coefficients, also known as loadings. Rows of X correspond to observations, columns to variables. COEFF is a P-by-P matrix, each column containing coefficients for one principal component. The columns are in order of decreasing component variance.

PRINCOMP centers X by subtracting off column means, but does not rescale the columns of X. To perform PCA with standardized variables, i.e., based on correlations, use PRINCOMP(ZSCORE(X)). To perform PCA directly on a covariance or correlation matrix, use PCACOV.

[COEFF, SCORE] = PRINCOMP(X) returns the principal component scores, i.e., the representation of X in the principal component space. Rows of SCORE correspond to observations, columns to components.

[COEFF, SCORE, LATENT] = PRINCOMP(X) returns the principal component variances, i.e., the eigenvalues of the covariance matrix of X, in LATENT.

[COEFF, SCORE, LATENT, TSQUARED] = PRINCOMP(X) returns Hotelling's T-squared statistic for each observation in X.

When N <= P, SCORE(:,N:P) and LATENT(N:P) are necessarily zero, and the columns of COEFF(:,N:P) define directions that are orthogonal to X.

[...] = PRINCOMP(X,'econ') returns only the elements of LATENT that are not necessarily zero, i.e., when N <= P, only the first N-1, and the corresponding columns of COEFF and SCORE. This can be significantly faster when P >> N.

See Also BARTTEST, BIPLOT, CANONCORR, FACTORAN, PCACOV, PCARES, ROTATEFACTORS.

References

1. Jackson, J.E., A User's Guide to Principal Components, Wiley, 1988.
2. Jolliffe, I.T. Principal Component Analysis, 2nd ed., Springer, 2002.
3. Krzanowski, W.J., Principles of Multivariate Analysis, Oxford University Press, 1988.
4. Seber, G.A.F., Multivariate Observations, Wiley, 1984.

Statistics_tools.**prplot_multilevel**(*Y*, *X*, *wh_col*)
Partial correlation plot for multi-level analysis

Usage

```
[X_resid, Y_resid, handles] = prplot_multilevel(Y, X, wh_col)
```

Uses unweighted estimates.

do not enter intercept in X

Statistics_tools.**r2z**(*r*, *n*, *varargin*)
Fisher's r to Z transformation for providing CIs for a correlation

Usage

```
[rci,sig,Z,p,rcrit] = r2z(r,n,[alph])
```

Inputs

n: n, # of observations going into correlation (count each row/subject once)

df = n - 3

alph: two-tailed p-value cutoff, default is p < .05

Outputs

rci: confidence interval in correlation values

sig: significant at alpha value?

Z: z-scores of correlations

p: p-values

can take a vector of r values

Examples

```
[rci,sig,z] = r2z(.1:.05:.9,5,.05); figure('Color','w');hold on; plot(.1:.05:.9,rci(:,1),'g','Li
[rci,sig,z] = r2z(.1:.05:.9,10,.05); hold on; plot(.1:.05:.9,rci(:,1),'r','LineWidth',2)
[rci,sig,z] = r2z(.1:.05:.9,20,.05); hold on; plot(.1:.05:.9,rci(:,1),'b','LineWidth',2)
[rci,sig,z] = r2z(.1:.05:.9,40,.05); hold on; plot(.1:.05:.9,rci(:,1),'m','LineWidth',2)
[rci,sig,z] = r2z(.1:.05:.9,80,.05); hold on; plot(.1:.05:.9,rci(:,1),'k','LineWidth',2)
set(gca,'FontSize',18)
legend({'n = 5' 'n = 10' 'n = 20' 'n = 40' 'n = 80'})
title('.05 Confidence interval lower bound on Pearson''s r')
xlabel('Correlation (r)')
ylabel('CI Lower Bound (r)')
c=.2:.01:.5;,[rci,sig,z,p,rcrit]=r2z(c,39,.05);[c' sig p]

ind=1;for i=1:10:5000,[rci,s,z,p,rc(ind)]=r2z(.5,39,.05/i);,ind=ind+1;,end
figure;plot(1:10:5000,rc);title('Critical r with Bonf correction'),xlabel('Comparisons')
```

Statistics_tools.**regress_best_subsets_ga**(*X*, *Y*)

Usage

GA-based best subsets regression

```
[wh_predictors, betas, b_subset, stat_subset] = regress_best_subsets_ga(X, Y)
```

Inputs

Y: is outcome data

X: is predictor matrix

wh_predictors: is the primary outcome – it is vector of which predictors to include in the model

the objective criterion is AIC

Statistics_tools.**repeated_ancova**(*X*, *Y*, *wicons*, *btwnnames*, *winames*, *ynames*, *varargin*)

Repeated measures ANCOVA with table and plot uses Robust IRLS

Usage

```
[b,stats,yadj] = repeated_ancova(X,Y,wicons,btwnnames,winames,ynames,varargin)
```

Examples

```
Y = rand(15,2);
X = Y + rand(15,2);
cons = [-1 1 -1 1; 1 1 -1 -1];  % placebo vs control, hot vs. warm

X = R.X(:,1:2);
Y = cl(2).CONTRAST.data;
cons = [-1 1 0 0; 0 0 -1 1];  % placebo vs control for heat then warm
repeated_ancova(X,Y,cons,{'Reported Placebo (C - P)' 'Order'},{'P-C Heat' 'P-C Warm'},{'CH' 'PH'
```

DOES NOT WORK WITH FIXED BTWN-SUBJECTS COVARIATES

X must be a random variable that is observed multiple times for each subject, as does Y

Statistics_tools.**rmanova2**(*data*, *alpha*, *doplot*, *ttst*)

Repeated-measures two-way ANOVA

Usage

```
stats = rmanova2(data,[alpha],[doplot],[ttst]);
```

Inputs

data: can be one of two formats: 1. Cell array - each row represents a level of factor 1, and

each column represents a level of factor 2. Each cell contains a vector of values of the dependent variable for each subject.

2. Matrix - each row represents a trial, with the following columns:

- column1 - dependent variable
- column2 - grouping variable for subject
- column3 - grouping variable for factor 1
- column4 - grouping variable for factor 2

alpha: (optional) p-value threshold (default: 0.05)

doplot:

(optional) if 1, will produce a line plot. Works only for cell input data (default: 1)

ttst: (optional) if 1, will perform pairwise t-tests (default: 0)

Statistics_tools.**robust_reg_pooled**(*X*, *Y*)

Usage

```
[betas,w] = robust_reg_pooled(X,Y)
[betas,stats] = weighted_reg(X,Y,'w',w,'uni');
```

Statistics_tools.**roc_boot**(*input_vals*, *binary_outcome*, *thr*, *verbose*)

Usage

```
[ci, names] = roc_boot(input_vals, binary_outcome, thr, [verbose flag])
```

Returns bootstrapped 95% confidence intervals for sensitivity, specificity, and PPV at a given threshold.

Examples

```
thr = ROC.class_threshold
input_vals = input(ind);
binary_outcome = outcome(ind);
```

Statistics_tools.**roc_calc**(*input_vals*, *binary_outcome*, *xvals*)

Calculate Receiver Operating Characteristic plot (ROC) given P-values

Usage

```
[xvals, tpr, fpr, auc, c_bias] = roc_calc(input_vals or input values, binary_outcome, [xvals : t
```

Inputs

input_vals: continuous-valued observations to classify (e.g., fMRI activity)

binary_outcome: 1 / 0 vector of which input observations are "hits"

xvals: Criterion values you put in or every 10th percentile of the input data distribution by default

Outputs

tpr: True positive rate for every step of ROC curve (sensitivity)

fpr: False positive rate (1 - specificity)

auc: Empirical estimate of area under the ROC curve

c_bias: c measure of response bias at each step; MacMillan and Creelman 2005

Examples

```
% May not work for p-values? may need to convert to t or something.
pvals = STATS.WTS.p;
isnull = DATA.true_weights == 0;
[xvals, tpr, fpr] = roc_calc(pvals, isnull);
figure; plot(fpr, tpr, 'ko-','Color', 'k', 'LineWidth', 2);

figure; plot(xvals, fpr, 'bo-')
hold on; plot([0 1], [0 1], 'k', 'LineWidth', 2);
set(gca, 'XLim', [0 .2], 'YLim', [0 .2])
xlabel('Nominal false positive rate');
ylabel('Actual false positive rate');
```

See Also roc_plot.m

Statistics_tools.**roc_plot**(*input_values*, *binary_outcome*, *varargin*)

This function makes a specific kind of ROC curve plot, based on input values along a continuous distribution and a binary outcome variable (logical)

Usage

```
ROC = roc_plot(input_values, binary_outcome, ['include', include])
```

Include is an optional logical variable of cases to include

Optional Inputs

'include': followed by logical vector of cases to include

'threshold': followed by a priori threshold cutoff for determining misclassification

'threshold_type':

followed by thresh type: choices below:

- 'Optimal balanced error rate'
- 'Optimal overall accuracy' [default]
- 'Minimum SDT bias'
- [Enter threshold OR threshold_type]

'color': followed by color, e.g., 'r' or [1 .5 0]

'plotmethod': followed by 'deciles' [default] or 'observed'

'nonormfit': suppress normal curve fitting to ROC

'plothistograms': plot histograms of the signal present/absent distributions

'writerscoreplus': Write text file for input into RScorePlus by Lew Harvey

'boot': [default] Bootstrap 95% confidence intervals for sens, spec, PPV at threshold

'noboot': Skip bootstrap

'balanced': Balanced accuracy for single interval classification

'dependent': followed by vector of subject IDs, e.g., ('dependent',[1,1,2,2,3,3].

This will perform multilevel version of binomial test for single interval classifation.

'noplot': Skip generating plots

'nooutput': Suppress text output

Outputs A structure containing the true and false pos rates (tpr, fpr) along the curve and the criterion threshold values of the input variable (thr) corresponding to these rates.

Uses the function roc_calc.m

Also returns some information about misclassified observations and line handle for ROC line plot and other statistics:

- area under ROC curve
- accuracy statistics based on binomial test
- PPV

Examples

```
ROC = roc_plot(pattern_exp_values, ishot);
ROC = roc_plot(pattern_exp_values, ishot, 'threshold', 2.5);
ROC = roc_plot(pattern_exp_values, ishot, 'color', 'r', 'twochoice');
ROC = roc_plot(pattern_exp_values, ishot, 'color', 'r', 'twochoice', 'nonormfit');
ROC = roc_plot(pexp, logical(outcome), 'color', 'g', 'plothistograms', 'threshold', 0.3188);
ROC = roc_plot(pexp, logical(outcome), 'twochoice', 'color', 'b', 'plothistograms');
ROC = roc_plot(pexp, logical(outcome), 'writerscoreplus');
ROC = roc_plot(pexp, logical(outcome), 'color', 'r', 'plotmethod', 'observed', 'plothistograms')
ROC = roc_plot(pexp, logical(outcome), 'color', 'm', 'plotmethod', 'observed', 'plothistograms',
```

For a whole image with p-values, this may be helpful.

Pre-specifies p-values you want to evaluate at.

```
rocout = roc_plot(1-t.p, truevals, 'plotmethod', 'observed', 'valuestoevaluate', 1 - [.5:-.1:.1
```

Statistics_tools.**rsquare_calc**(*X*, *Y*)

Usage

```
rsquare(X, y)
```

Returns variance in each col. of Y explained by X

Statistics_tools.**rsquare_multiple_regions_multilevel**(*Y*, *X*, *varargin*)

Predict outcome var (y) using data from multiple vars (X, e.g., brain regions) Test R-square against permuted data

Usage

```
stats = rsquare_multiple_regions_multilevel(Y, X, varargin)
```

Inputs Variable args

'colors': followed by colors cell ({'r' 'g' 'b'}) for each col. of X

'nperms: then num perms

Examples

```
% SETUP data
cl = [clpos_data clneg_data];

cl(cat(1, cl.numVox) < min_cluster_size) = [];

% get brain data cell
for i = 1:size(cl(1).all_data, 2)
    for c = 1:length(cl)
        data{i}(:,c) = cl(c).all_data(:, i);
    end
end

% NMDS ANALYSIS
OUT = [];

OUT.ridge = matrix_direct_effects_ridge(data);
D = OUT.ridge.mean; D(find(eye(size(D)))) = 1;
D = (D' + D) ./ 2;
OUT.ridge.D = (1 - D) ./ 2;
[OUT.stats_mds.GroupSpace,OUT.stats_mds.obs,OUT.stats_mds.implied_dissim] = shepardplot(OUT.ridg

OUT.stats_mds = nmdsfig_tools('cluster_solution',OUT.stats_mds, OUT.stats_mds.GroupSpace, 2:max_
OUT.stats_mds.colors = {'ro' 'go' 'bo' 'yo' 'co' 'mo' 'ko' 'r^' 'g^' 'b^' 'y^' 'c^' 'm^' 'k^'};
create_figure('nmdsfig');

OUT.stats_mds.names = [];
nmdsfig(OUT.stats_mds.GroupSpace,'classes',OUT.stats_mds.ClusterSolution.classes,'names',OUT.sta
hh = nmdsfig_fill(OUT.stats_mds);
axis image; axis equal
```

```
    % Multiple regions predict behavior

    % Design matrix with cluster averages
    classes = OUT.stats_mds.ClusterSolution.classes;
    clear X
    for i = 1:length(data)
        for j = 1:max(classes)
            X{i}(:, j) = nanmean(data{i}(:, classes == j), 2);
        end
    end

    OUT.stats_regression = rsquare_multiple_regions_multilevel(acc_x_dist, X, 'colors', OUT.stats_md
```

Statistics_tools.**scn_stats_helper_functions**(*meth*, *varargin*)

Helper functions for stats routines, plotting, and printing

Available Methods

'print': print outcome table from stats structure

'gls': GLS function, weighted or unweighted; with AR model if specified as last input

'boot': Boostrapping of GLS

'signperm': Sign permutation test for intercept of GLS

'plot':

'loess_xy': loess plots of multilevel X and Y data

X is cell array with X data for each subject, Y is cell array, same format

'xycatplot': LOESS (or no loess) plots of N subjects, categorical X vs Y

'loess_partial':

'xyplot': multilevel line plot of x vs y within subjects, grouping trials by optional G variable. X can be categorical or continuous.

'xytspanelplot': X and Y are timeseries data; separate panel plot for each cell

Format Strings

```
    scn_stats_helper_functions('print', stats, stats_within)

    [b, s2between_ols, stats] = scn_stats_helper_functions('gls', Y, W, X)
    [b, s2between_ols] = scn_stats_helper_functions('gls', Y, W, X, arorder);
```

See glmfit_general for context and usage

```
    stats = scn_stats_helper_functions('boot', Y, W, X, bootsamples, stats, whpvals_for_boot, target
    stats = scn_stats_helper_functions('boot', Y, stats.W, X, 1000, stats, 1:size(Y,2), .005, 1 )

    stats = scn_stats_helper_functions('signperm', Y, W, X, nperms, stats, permsign, verbose )
    stats = scn_stats_helper_functions('signperm', Y, W, X, 5000, stats, [], 1 );

    stats = scn_stats_helper_functions('xycatplot', X, Y);
    stats_plot = scn_stats_helper_functions('xycatplot', stats.inputOptions.X, stats.inputOptions.Y,

    scn_stats_helper_functions('xyplot', data(1:19), SETUP.data.Y, 'weighted', 'groupby', G, 'colors
```

Example using sign permutation test

```
    Y = randn(20, 4); X = Y(:,1) + randn(20,1); X = [ones(size(X)) X];
    first_lev_var = rand(20, 4);
    stats = glmfit_general(Y, X, 'weighted', 's2', first_lev_var, 'dfwithin', 20, 'verbose');
    W = stats.W;
    stats = scn_stats_helper_functions('signperm', Y, W, X, 5000, stats, [], 1 );
    % Re-run using already set-up permsign:
    stats = scn_stats_helper_functions('signperm', Y, W, X, 5000, stats, stats.permsign, 1 );


    scn_stats_helper_functions('loess_xy', stats.inputOptions.X, stats.inputOptions.Y)


    % Loading mediation clusters and making line plot
    G = SETUP.data.X;
    whcl = 30; cluster_orthviews(clneg_data{1}(whcl));
    clear data, for i = 1:length(clneg_data), data{i} = clneg_data{i}(whcl).timeseries; end
    scn_stats_helper_functions('xyplot', data, SETUP.data.Y, 'weighted', 'groupby', G, 'colors', {'y
    scn_stats_helper_functions('xyplot', SETUP.data.Y, data, 'weighted', 'groupby', G, 'colors', {'y
```

Note Out of memory errors for large n!

Called by

- mediation.m
- glmfit_general.m

Tor Wager, Sept 2007

Statistics_tools.**sdt_A**(*H*, *F*)

Calculate signal detection theory A statistic: a non-parametric estimate of sensitivity in ROC analysis per Zhang and Mueller (2005)

Usage

```
    A = sdt_A(H, F)
```

Inputs

H: hit rate

F: false alarm rate

Meaning of values where F > H ("reverse skill"):

The meaningful measure of sensitivity in this case is A(F,H) as opposed to A(H,F). Subject is assumed to still be sensitive, but to have inverted their interpretation (or "reverse skill").

To distinguish from the normal case, such results are returned as (1 - sdt_A(F,H)), so the full range from 0 to 1 is used. For comparisons of sensitivity use abs(sdt_A(H,F) - 0.5).

Example

sdt_A(0.6,0.8) = 0.325. This indicates the same sensitivity as sdt_A(0.8,0.6) = 0.675, but with reversed skill.

Statistics_tools.**searchlight_applymask**(*dat1*, *dat2*, *varargin*)

Estimate local pattern weight on train data using SVR and searchlight and apply weights to the test dataset

Usage

```
[dat, sl_size] = searchlight_applymask(train, test, varargin)
```

Inputs

dat1: fmri_data object with train.Y == size(train.dat,2)

dat2: Data to apply local weight map.

Optional inputs

'r': searchlight sphere radius (in voxel) (default: r = 3 voxels)

'parallel':

run subset of voxels to distribute on a cluster. flag must be followed by array specifing id and total number of jobs (e.g., 'parallel',[1,10]);

Outputs

dat: This contains an fmri_data object that contain correlation pattern expression values

sl_size: The number of voxels within each searchlight. Based on this number, you can detect searchlights on the edge (searchlights with low sl_size should be on the edge of the brain.

Examples

```
[r, dat] = searchlight_applymask(train, test, 'r', 5);

[r, dat] = searchlight_applymask(train, test, 'r', 5,'parallel',[1,10]);
```

Statistics_tools.**searchlight_applymask_collate**(*dat2*, *file_list*)

Estimate local pattern weight on train data using SVR and searchlight and apply weights to the test dataset

Usage

```
dat_comb = searchlight_applymask(dat2, file_list)
```

Inputs

dat2: fmri_data test object

file_list: Cellarray of list of file distributed in parallel. Make sure it is sorted correctly (e.g., sort_nat())

Output

dat_comb: An fmri_data object with searchlight weights applied to test dataset.

Example

```
dat_comb = searchlight_applymask_collate(dat2, file_list);
```

Statistics_tools.**searchlight_correlation**(*mask1*, *mask2*, *varargin*)

Calculate the local pattern similarity between two pattern maps using the searchlight approach.

Usage

```
[r_corr, dat, sl_size] = searchlight_correlation(mask1, mask2, [additional_inputs])
```

Inputs

mask1: pattern or activation maps 1

mask2: pattern or activation mediaps 2

Optional inputs

'r': searchlight sphere radius (in voxel) (default: r = 3 voxels)

'type': This calls corr.m, and can take 'type' option.

'Pearson' (default), 'Kendall', 'Spearman'.

Outputs

r_corr: Correlation between weights of two pattern masks

dat: This contains a statistic_image object that contain correlation values between weights of two pattern masks (=r_corr; in .dat) and p values for the correlation values (in .p).

sl_size: The number of voxels within each searchlight. Based on this number, you can detect searchlights on the edge (searchlights with low sl_size should be on the edge of the brain.

Examples

```
mask1 = which('weights_NSF_grouppred_cvpcr.img');
mask2 = which('nonnoc_v6_109subjmap_mean.nii');

[r, dat] = searchlight_correlation(mask1, mask2, 'r', 5);
```

Statistics_tools.**searchlight_disti**(*dat*, *mask*, *dist_i*, *additional_inputs*)

Run the actual searchlight analysis on each brain chunck searchlight_dream.m will generate codes to run this funtion.

Usage

```
out = searchlight_disti(dat, mask, dist_i, [additional_inputs])
```

See Also

searchlight_dream.m, xval_cross_classify.m, fmri_data.predict.m,

Statistics_tools.**searchlight_dream**(*dat*, *dist_n*, *mask*, *varargin*)

This function generates codes for submitting multiple distributed jobs to clusters to run a searchlight analysis on multiple chunks of the brain.

Usage

```
searchlight_dream(dat, dist_n, mask, 'algorithm_name', 'cv_svm' (or 'cv_lassopcr'), 'cv_assign',
```

Features

- generates dist_n scripts in modeldir (or current directory)
- can run a searchlight analysis on one dataset, or two datasets (cross-classification)
- can apply different radius
- can obtain cross-validated results with 'cv_assign' option
- can use SVM (linear svm is a default) and LASSO-PCR. You need to have a spider toolbox and lasso rocha toolbox in your path
- you can save predictive weights for each searchlight or discard them

Inputs

dat: image_vector or fmri_data object with data

dat1.Y(:,1): for svm: true(1) or false(-1) for each observation (image) in Y(:,1) for lassopcr: continuous value for Y(:,1)

dat1.Y(:,[2:n]): Test sets: could be binary: and true(1), false(-1), ignore(0) or continuous values

dist_n: The number of jobs (brain chunks) you want to create

mask: This will be run on voxels within the mask e.g., which('scalped_avg152T1_graymatter.img')

'algorithm_name': should be followed by 'cv_svm' or 'cv_lassopcr'

'cv_assign': a vector of integers for membership in custom holdout set of each fold

Optional Inputs

'dat2': cross-classification; should be followed by dat2 and dat2.Y dat2.Y(:,1) - for training/testing, dat2(:,[2:n]) - for testing

'r': searchlight sphere radius (in voxel) (default: r = 3 voxels)

'modeldir': the directory where all the variables and results will be saved; should be followed by a directory location (default: the current directory)

'scale': z-scored input data in image_vector or fmri_data object (default = false)

'balanced': use the balanced ridge option - balanced ridge value should be followed. (default = false)

'outcome_method':

followed by the following options

- 'correlation' - "default" for for continuous measures

'twochoice'- "default" for binary outcome 'singleinterval' - for binary outcome

'save_weights': save weights for each searchlight (default = false)

'email': should be followed by an email adress (default = false)

Outputs

This function will generate codes that call "searchlight_disti.m", which will save the following output variables.

out.test_results: accuracy, p, and se for binary classification, and correlation (pearson), p for prediction of continuous values For the cross-classification, test_results will save four values for each searchlight. The order of the test results are [dat1-on-dat1, dat1-on-dat2, dat2-on-dat1, dat2-on-dat2]. All results are cross-validated results.

out.stats1 & stats2: stats1 and stats2 are similar to the outputs of predict function.

Examples for lassopcr

```
% data preparation
dat = fmri_data(which('brainmask.nii'));
dat.dat = randn(dat.volInfo.n_inmask, 30);

% setting up training values
dat.Y = dat.dat(111111, :)' + .3 * randn(30, 1);

% setting up testing values
```

```
    dat.Y(:,2) = [ones(10,1); zeros(10,1); -ones(10,1)];
    dat.Y(:,3) = dat.dat(111111, :)' + .3 * randn(30, 1);
    mask = which('scalped_avg152T1_graymatter.img');
    dist_n = 50;

    % data for cross classification
    dat2 = fmri_data(which('brainmask.nii'));
    dat2.dat = randn(dat.volInfo.n_inmask, 30);
    dat2.Y = dat2.dat(111, :)' + .3 * randn(30, 1);
    dat2.Y(:,2) = dat.Y(:,2);
    dat2.Y(:,3) = dat2.dat(111111, :)' + .3 * randn(30, 1);

    % setting up other variables
    r = 6;
    modeldir = '/dreamio/home/chwo9116/Documents/searchlight_dream_test';
    holdout_set = ones(6, 1); for i = 2:5, holdout_set = [holdout_set; i*ones(6, 1)]; end

    % generate scripts in modeldir
    searchlight_dream(dat, dist_n, mask, 'dat2', dat2, 'algorithm_name', ...
    'cv_lassopcr', 'r', 6, 'modeldir', modeldir, 'cv_assign', holdout_set, ...
    'save_weights', 'outcome_method', 'singleinterval');
```

See Also

searchlight_disti.m, xval_cross_classify.m, fmri_data.predict.m,

Statistics_tools.**searchlight_saveresults**(*modeldir*)

This function combines and saves searchlight analysis results.

Usage

```
    searchlight_saveresults(modeldir)
```

Input

modeldir: the directory where the searchlight result mat files (e.g., searchlight_results_*_01Aug2014.mat)

Outputs This function saves the brain maps where each voxel contains a summary stat value (e.g., accuracy or outcome correlation) in the modeldir. The naming convention of the result maps are as follow:

results_searchlight_(a)_(b)_dat(c).nii

1. will be a number - the number of test results
2. acc: accuracy, r: outcome correlation, p: p-values, se: standard error, and thr: threshold for the single-interval test
3. If the test was on one dataset, (c) will be empty, but if the test was cross-prediction, (c) will be 11, 12, 21, or 22.
   - 11: trained on the first dataset, and tested on the first dataset
   - 12: trained on the first dataset, and tested on the second dataset
   - 21: trained on the second dataset, and tested on the first dataset
   - 22: trained on the second dataset, and tested on the second dataset

Output Examples

```
    results_searchlight_1_r_dat11.nii (outcome correlation)
    results_searchlight_1_p_dat11.nii (p value for the correlation values)

    results_searchlight_2_acc_dat11.nii (accuracy)
    results_searchlight_2_p_dat11.nii (p value for the accuracy)
    results_searchlight_2_se_dat11.nii (standard error for the accuracy)
    results_searchlight_2_thr_dat11.nii (threshold for the accuracy test)
```

Statistics_tools.**shift_correl**(*a*, *b*, *varargin*)

Usage

```
    [shiftvals, corrvals, bestshift, bestcorr, aout, bout] = shift_correl(a, b, ['max_shift', max_sh
```

Shifts a backwards and forwards by max_shift elements (default 12) in each direction, computing the correlation between a and b at each shift value

Thus, negative values mean a happens AFTER b Positive values means b happens later

truncates the tails of a and b where necessary to ensure they're the same length.

Inputs

a, b: two vectors to correlate

max_shift: max number of elements to shift (default 12)

shift_step: size of incremental shifts (default .1)

use_rob_stats: robust regression, IRLS (default 0)

return_betas: return betas rather than corr coeffs (default 0)

display_plots: display plot of shift vals and correlations/betas (default 0)

NOTE: robust r value of chosen solution may not be max, as weighted_corrcoef now does not account for variance inflation with low weights for some observations

priors: Incorporate gaussian priors with mean priors(1) and std priors(2)

Examples

```
    % first extract data for a subject into clusters:
    clusters = roi_probe(spm_get(Inf), 'snpm0002_varsm_cov_clusters.mat');

    % OR

    clusters = mask2clusters('SnPMt_filtered.img', spm_get(Inf));

    % Then do the shifting:

    for i=1:length(clusters)
        [shiftvals, corrvals] = shift_correl(xX.X(:, 1), clusters(i).timeseries);
        title(['Cluster' num2str(i)]);
        disp(['Cluster ' num2str(i) ' estimate shift by ' num2str(1.5 * shiftvals(corrvals == max(cc
    end


    % Simulation
    trueval = 1;
    n = 100;
    noisevar = 0;
    a = randn(n,1);
```

```
    b = a + randn(n,1).*noisevar;
    a=smooth_timeseries(a,n./5);
    b=smooth_timeseries(b,n./5);
    a = shift_signal(a,trueval);
    b = b(1:length(a));
    figure;plot(a);
    hold on;
    plot(b,'r')
    [shiftvals, corrvals, bestshift, bestcorr, aout, bout] = ...
    shift_correl(a, b, 'max_shift', 4, 'shift_step', .2, 'use_rob_stats', 1, 'return_betas', 0, 'dis

    r = .7; shiftstep = 1;
    optstr = 'optimize'; %    or 'noopt'
    tic
    for i = 1:10
        ab = mvnrnd([0 0],[1 r; r 1],n);
        a = ab(:,1);
        b = ab(:,2);
        a=smooth_timeseries(a,n./5);
        b=smooth_timeseries(b,n./5);
        a = shift_signal(a,trueval);
        b = b(1:length(a));
        [shiftvals, corrvals, bestshift(i), bestcorr(i), aout, bout] = shift_correl(a, b, 'max_shift
    end
    toc
```

Statistics_tools.**shift_signal**(*y*, *shiftby*, *interp_method*)

Usage

```
    [shiftedy] = shift_signal(y, shiftby, [interpolation method])
```

Shifts signal contained in y back 'shiftby' steps

Note the variable shiftby should be positive.

Statistics_tools.**signtest_matrix**(*dat*)

Usage

```
    stats = signtest_matrix(dat)
```

This is a matric-ized version of Matlab 2010's signtest.m it returns identical P-values to Matlab's function

Statistics_tools.**ste**(*dat*)

Usage

```
    [my_ste,t,n_in_column,p,mean] = ste(dat)
```

standard error of the mean and t-values, columnwise and p-values, if asked for

omits NaN values row-wise within each column

does NOT use n - 1 matches matlab t-test function

Statistics_tools.**stepwise_tor**(*dat*, *y*, *varargin*)

Usage

```
    STEPWISE = stepwise_tor(dat, y, [pred. names], [alpha])
```

Stepwise regression using Matlab with a couple of extras:

Print omnibus F-values for stepwise regression

get adjusted R-squared

save output structure

Statistics_tools.**stouffer**(*p*, *varargin*)

Usage

```
[z,p,sig] = stouffer(p,[alph])
```

Inputs

p: p values in 4-D array

1st 3 dims are within images, dim4 = image

optional: alpha value for thresholding

Outputs

z: stouffer's combined test statistic, compare to normal

p: p-values for combined test

sig: signficance 1 / 0 binary mask, if alpha is specified

Described in

Lazar, N. A., Luna, B., Sweeney, J. A., & Eddy, W. F. (2002). Combining brains: a survey of methods for statistical pooling of information. Neuroimage, 16(2), 538-550.

Stouffer, S. A., Suchman, E. A., DeVinney, L. C., Star, S. A., and Williams, R. M. 1949. The American Soldier: Vol. I. Adjustment During Army Life. Princeton University Press, Princeton.

Statistics_tools.**subset_indicator_matrix**(*n*)

Create a matrix whose rows contain indicators (1/0 values) for all possible subsets of n variables

Usage

```
subsets = subset_indicator_matrix(n)
```

Use this, for example, to create a matrix that tells you all possible combinations of regressors to include in a regression.

Tor Wager, March 07

Statistics_tools.**t_test2**(*x*, *m*, *alpha*, *tail*)

TTEST Hypothesis test: Compares the sample average to a constant.

Usage

```
[H,SIG] = TTEST(X,M,ALPHA,TAIL)
```

performs a T-test to determine if a sample from a normal distribution (in X) could have mean M.

M = 0, ALPHA = 0.05 and TAIL = 0 by default.

The Null hypothesis is: "mean is equal to M".

- For TAIL=0, alternative: "mean is not M".
- For TAIL=1, alternative: "mean is greater than M"
- For TAIL=-1, alternative: "mean is less than M"

- TAIL = 0 by default.

ALPHA is desired significance level.

SIG is the probability of observing the given result by chance given that the null hypothesis is true. Small values of SIG cast doubt on the validity of the null hypothesis.

•H=0 => "Do not reject null hypothesis at significance level of alpha."

•H=1 => "Reject null hypothesis at significance level of alpha."

References [1] E. Kreyszig, "Introductory Mathematical Statistics", John Wiley, 1970, page 206.

Statistics_tools.**testclustnew**(*X*, *clust*, *varargin*)

This function tests for the significance of the cluster solution

Usage

```
[bestpval,bestmyclass,bestnames,bestX,where,clustnames]=testclustnew(X,clust,[r],[nperm],[names]
```

Inputs

X: is the Group Space

clust: is the cluster solutions (between how many and how many solutions is reasonable?) the default for this option is 2:r/2 where r is the number of

REGIONS: r is the number of DIMENSIONS in the solution (from choose_ndims)

nperm:

is the number of permutations for nonparametric testing (default 1000)

names: specifies the names of each region

remove: specifies what to do about elements which fit the cluster solution poorly. There are 3 possibilities:

'keep' - keeps all elements regardless of quality. If you choose this option, you are assuming that every region you enter contributes something to your solution 'thresh' - this removes elements which fail to reach 95% confidence as determined by random permutation testing

'iter' - this option removes elements which fall in the bottom 5% of the permuted distribution, and remcomputes the solution without these elements. This iterative recomputation means that you are pruning your solution *until* you get a good one. The p-values which accompany each solution are thus difficult to intepret if you use this option

linkagetype: is the input to linkage ('single','average',etc)

see help linkage_t

Outputs

bestpval: is the overall significance of the derived solution. significance is calculated by permuting the group space in all dimensions to derive a completely new configuration of elements in n-dimensional space. These elements are then assigned to clusters exactly as for the point estimate solution, and the mean quality (silhouette value) across the entire plot is used to form a distribution against which to test the sihouette value of the true solution

bestmyclass: is the assignment of REGIONS to CLUSTERS for the best soluton

bestnames: contains the names of the REGIONS included in the best solution (if remove==keep, this will be the same as names);

bestX: is the best group space

where: is a vector indexing which of the original regions (in order) are included in the new group space. clustnames is a cell containing the names of the elements in each cluster

Calls getmeanquality,clustquality,pdist1,linkage_t,cluster_t,makebinary

Statistics_tools.**time_varying_estimate**(*meth*, *data*, *varargin*)

Performs correlation operation (default) or any function you pass in (as a function handle) on symmetric moving average of data

Usage

```
y = time_varying_estimate(meth,data,[window width],[function handle])
```

Works on all columns of the data input matrix together so function handles can be multivariate

Window options (meth argument):

- 'gaussian'
- 'tukey'

Optional Input stepby

Sometimes input data is very high resolution, and it would take too much time to work element by element across the inputs. You can enter an option here to compute estimates at every n-th lag.

Examples

```
y =  time_varying_estimate('gaussian',data,20);

% Generate sample data:
x = mvnrnd([0 0], [1 .6; .6 1], 100);

% Correlation between columns of x:
r = time_varying_estimate('tukey', x, 20);

% St. deviation of first column
mystd = time_varying_estimate('tukey', x(:, 1), 20, @(y) std(y));
```

Statistics_tools.**tscv**(*vectorlen*, *varargin*)

Create a crossvalidation test and train index for time series data. Larger h will ensure less dependence. Larger v creates larger test sets Number of folds = vectorlen - (2*h + 2*v)

-See http://robjhyndman.com/hyndsight/tscvexample/ for more info about rolling cv -See Racine, J. (2000). Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. Journal of Econometrics, 99(1), 39-61.

Usage

```
[trIdx, teIdx] = tscv(vectorlen, stepsize)
```

Inputs

vectorlen: length of vector to create holdout cross-validation set

Optional Inputs with their default values

'hvblock' = [h,v]: use hvblock cross-validation with a block size of 'h' (0 reduces to v-fold xval)and number of test observations 'v' (0 reduces to h-block xval)

'rolling' = [h,v,g]: use hvblock cross-validation with g training steps surrounding hv block. Akin to Rolling crossval. Same properties as hvblock.

Outputs

trIdx: structure with training label index

teIdx: structure with test label index

Examples

```
[trIdx, teIdx] = tscv(100, 'hvblock',[5,2]); % use hvblock with h=5 and v=2
[trIdx, teIdx] = tscv(100, 'rolling',[5,2,10]); % use hvblock with h=5, v=2 and g=10
```

Statistics_tools.**tsquaretest**(*X*, *pthresh*, *u0*)

Hotelling's t-square test that the multivariate sample mean of X is different from u0.

X is an observations x variables data matrix If pthresh is entered, the critical t2 value is returned If u0 is not entered, the default null hypothesis is mean zero (the origin)

Statistics_tools.**ttest2_printout**(*sc1*, *sc2*, *varargin*)

Usage

```
[H,p,ci,stats] = ttest2_printout(sc1,sc2,[doplot],[covts])
```

one or two sample t-test printout and plot covariates are not done yet!

Inputs

sc1: data from first group

sc2: data from second group (if missing or empty, performs one-sample t-test)

Statistics_tools.**ttest3d**(*xc*)

calculate t-statistic values for a k x k x n 3-D matrix of values across n subjects.

Usage

```
[mean,t,sig,out] = ttest3d(xc)
```

stats on each element, across 3rd dim

to print output, see also: correlation_to_text(mxc,sig);

Statistics_tools.**var_prctile**(*p*, *n_in_sample*, *varargin*)

Usage

```
[var_prc,pci,Nneeded,pcitarget] = var_prctile(p,n_in_sample,['nboot',nboot],['x',data])

[var_prc,pci,Nneeded,pcitarget] = var_prctile(p,50,'nboot',5000)
```

Inputs

p: is desired prctile of data (threshold)

nboot: is number of bootstrap samples (if bootstrapping)

n_in_sample: is number of obs. in original sample

x: is data sample of distribution of interest (empirical pdf based on this)

Note: using empirical PDF/CDF depends a great deal on choice of h (see code).

Examples

```
Nneeded = [];
for p = [.05:-.001:.001]
    [var_prc,pci,Nneeded(end+1),pcitarget] = var_prctile(x,p);
end
figure;
plot([.05:-.001:.001],Nneeded)
```

Statistics_tools.**weighted_glmfit**(*Y*, *varargin*)

Calculate weighted average using weighted linear least squares See examples below for usage

Model: Y_i = 1*Ypop + noise

Inputs

Y: data matrix (nsub x T)

w: weights

varY: variance of data at each time point (nsub x T) + var between

Outputs

Ymean: weighted mean of each column of Y

dfe: error degrees of freedom, adjusted for inequality of variance (Sattherwaite) and pooled across data columns

Extended output in stats structure

stats.t: t-values for weighted t-test

stats.p: 2-tailed p-values for weighted t-test

r: weighted correlation coeff across columns of Y

xy: weighted covariance matrix

v:

weighted variance estimates for each column of Y

- sqrt(v) is the standard error of the mean (or grp difference)

stats.fits: fits for each group (Ymean by group), low contrast weight group then high

Fastest if no stats are asked for.

Computation time

For FULL stats report

- Triples from 500 -> 1000 columns of Y, continues to increase

For mean/dfe only, fast for full dataset (many columns of Y)

Examples

Basic multivariate stats for 1000 columns of dat, no weighting Multivariate covariances are meaningful if cols of Y are organized, e.g., timeseries

```
[means,stats] = weighted_glmfit(dat(:,1:1000));
```

The same, but return univariate stats only (good for large Y)

```
[means,stats] = weighted_glmfit(dat,'uni');
```

A weighted version, where we put in the weights, and with a design matrix too:

```
[means,stats] = weighted_glmfit(X,dat,'uni','w',weights);
```

A weighted version, where weights are determined from w/i subject variances:

```
[means,stats] = weighted_glmfit(X,dat,'uni','vary',variances);
```

Statistics_tools.**xcorr_multisubject**(*data*, *varargin*)

Cross-correlation and partial correlation matrices for 3-D data, i.e., a cell array of subject data matrices

Usage

```
OUT = xcorr_multisubject(data, [optional inputs])
```

Inputs

data: A cell array, one cell per subject/replicate, of n x k data to be inter-correlated.

Optional Inputs

'partialr': Use partial correlations obtained via ridge regression (k = 1 fixed)

'shift_by': Followed by integer value for max number of time points to shift

Output

OUT: A structure containing subject correlation matrices, the mean matrix, and raw and FDR-thresholded group matrix

Examples

```
% cl cell structure:
for i = 1:length(clpos_data), data{i} = cat(2, clpos_data{i}.timeseries); end

% cl structure:
for i = 1:size(cl(1).all_data, 2), for c = 1:length(cl), data{i}(:,c) = cl(c).all_data(:, i); en

% parcel_cl_avgs or clpos_data2 structure:
for i = 1:length(parcel_cl_avgs), for j = 1:N, data{j}(:,i) = parcel_cl_avgs(i).timeseries{j}; e

create_figure('Xcorr', 1, 3);
imagesc(OUT.stats.mean);
subplot(1, 3, 2);
imagesc(OUT.stats.sig);
subplot(1, 3, 3);
imagesc(OUT.stats.fdrsig);
colormap gray
```

Example of MDS and plotting total (not decomposed) relationships:

```
OUT.stats.D = (1 - OUT.stats.mean) ./ 2;
[OUT.stats_mds.GroupSpace,OUT.stats_mds.obs,OUT.stats_mds.implied_dissim] = shepardplot(OUT.stat
OUT.stats_mds = nmdsfig_tools('cluster_solution',OUT.stats_mds, OUT.stats_mds.GroupSpace, 2:5, 1
nmdsfig(OUT.stats_mds.GroupSpace,'classes',OUT.stats_mds.ClusterSolution.classes,'names',OUT.sta
```

Example of MDS and plotting direct relationships:

```
OUT.ridge = matrix_direct_effects_ridge(data);
D = OUT.ridge.mean; D(find(eye(size(D)))) = 1;
D = (D' + D) ./ 2;
OUT.ridge.D = (1 - D) ./ 2;
[OUT.stats_mds.GroupSpace,OUT.stats_mds.obs,OUT.stats_mds.implied_dissim] = shepardplot(OUT.ridg
OUT.stats_mds = nmdsfig_tools('cluster_solution',OUT.stats_mds, OUT.stats_mds.GroupSpace, 2:10,
nmdsfig(OUT.stats_mds.GroupSpace,'classes',OUT.stats_mds.ClusterSolution.classes,'names',OUT.sta
hh = nmdsfig_fill(OUT.stats_mds);
axis image, axis equal

OUT.stats_mds = nmdsfig_tools('cluster_solution',OUT.stats_mds, OUT.stats_mds.GroupSpace, 2:5, 1

% data is cell, one cell per subject
[OUT.stats_mds.GroupSpace,OUT.stats_mds.obs,OUT.stats_mds.implied_dissim] = shepardplot(OUT.stat
OUT.stats_mds = nmdsfig_tools('cluster_solution',OUT.stats_mds, OUT.stats_mds.GroupSpace, 2:5, 1
```

Statistics_tools.**xcorr_xy_multisubject**(*X*, *Y*)

This function will cross-correlate two variables, X and Y for each of N subjects. Correlation and latency values will be saved.

Second-level tests are done across the N subjects on each of the correlation and latency values.

Inputs

X: must be an observations x N matrix

NOTE: observations are assumed to be timeseries values!

this DOES matter because an AR(2) model is used...see below

Y: must be the same.

cross-correlations will be computed for pairs of columns, separately for each successive column of X / Y

This function ues shift_correl.m, with a two-pass procedure.

The first pass provides initial estimates, including an estimate of the latency standard deviation, which is used as an Empirical Bayes prior. The second pass is used to apply the EBayes priors and estimate latencies and cross-correlation values. An ar(2) model is used to estimate the DF.

The estimates will be biased towards zero in the second pass, but they will likely be lower variance.

stats.latency and stats.association return group-level stats on these computed using a sign pemutation test (to avoid normality assumption). the "association" test is a test of the relationship, but is different than a standard multilevel model because it uses correlation values rather than slope values, and the sign permutation test.

3.15 Visualization_functions

Visualization_functions.**addbrain**(*varargin*)

Usage

```
handle = addbrain([method],enter 2nd arg to suppress lighting changes)
```

quick function to add transparent brain surface to figure

```
han = addbrain;   % lateral surface

han = addbrain('brainstem');
```

NOTE: this version uses structures in SPM2 space (Colin atlas) Available keywords:

CORTICAL SURFACES

'transparent_surface': the default. 2 mm res SPM2 brain surface

'hires': a high-resolution surface (from Caret segmentation)

'hires left': hi-resolution left medial with cerebellum (Caret seg)

'hires right': same, right hem

'left': 2 mm resolution left hem, no cerebellum

'right':

'vmpfc':

CUTAWAY SURFACES 'brainbottom':

COMPOSITES

'limbic': A collection of subcortical nuclei with left surface

'foursurfaces': Lateral and medial views, with brainstem

'BG': Basal ganglia

SUBCORTICAL SURFACES

- 'brainstem'
- 'amygdala'
- 'thalamus'
- 'hippocampus'
- 'midbrain'
- 'caudate'
- 'globus pallidus'
- 'putamen'
- 'nucleus accumbens'
- 'hypothalamus'
- 'cerebellum'
- {'md','mediodorsal'}
- {'cm','centromedian'}
- 'pbn'
- 'rvm'
- 'nts'
- 'lc'
- {'sn', 'substantia nigra'}

- {'stn', 'subthalamic nucleus'}
- {'rn', 'red nucleus'}
- {'olive', 'inferior olive'}
- {'nrm', 'raphe magnus'}

SPECIAL COMMANDS

han = addbrain('colorchange',my_rgb_color,han);

Change gray background to some other color, excluding blobs already rendered

- han: Input handles with patch object
- my_rgb_color: [x x x] color triplet
- Only works for changing from gray background right now.

han = addbrain('eraseblobs',han);

Set all rendered blob colors back to gray; useful for re-rendering on existing surfaces.

- han: Input handles with patch object
- Only works for changing to gray background right now.

See also: cluster_surf, img2surf.m, surface() methods for objects, cluster_cutaways

Visualization_functions.**addbrainleft**(*varargin*)

Usage

```
handle = addbrainleft(enter arg to suppress lighting changes)
```

quick function to add transparent brain surface to figure

Visualization_functions.**addbrainright**(*varargin*)

Usage

```
handle = addbrain(enter arg to suppress lighting changes)
```

quick function to add transparent brain surface to figure

Visualization_functions.**applycolormap**(*h*, *mapname*)

Visualization_functions.**arrow**(*varargin*)

ARROW Draw a line with an arrowhead.

ARROW(Start,Stop) draws a line with an arrow from Start to Stop (points should be vectors of length 2 or 3, or matrices with 2 or 3 columns), and returns the graphics handle of the arrow(s).

ARROW uses the mouse (click-drag) to create an arrow.

ARROW DEMO & ARROW DEMO2 show 3-D & 2-D demos of the capabilities of ARROW.

ARROW may be called with a normal argument list or a property-based list.
ARROW(Start,Stop,Length,BaseAngle,TipAngle,Width,Page,CrossDir) is the full normal argument list, where all but the Start and Stop points are optional. If you need to specify a later argument (e.g., Page) but want default values of earlier ones (e.g., TipAngle), pass an empty matrix for the earlier ones (e.g., TipAngle=[]).

ARROW('Property1',PropVal1,'Property2',PropVal2,...) creates arrows with the given properties, using default values for any unspecified or given as 'default' or NaN. Some properties used for line and patch objects are used in a modified fashion, others are passed directly to LINE, PATCH, or SET. For a detailed properties explanation, call ARROW PROPERTIES.

Start The starting points. B Stop The end points. /|\ ^ **Length Length of the arrowhead in pixels.** /||| | BaseAngle Base angle in degrees (ADE). //|||\ L| TipAngle Tip angle in degrees (ABC). ///|||\ e| Width Width of the base in pixels. ////|||\\ n| Page Use hardcopy proportions. /////|D|\\ g| CrossDir Vector || to arrowhead plane. //// ||| \\ t| NormalDir Vector out of arrowhead plane. /// ||| \ h| Ends Which end has an arrowhead. //<——>|| \ | ObjectHandles Vector of handles to update. / base ||| V

E angle||<———>C

ARROW(H,'Prop1',PropVal1,...), where H is a |||tipangle vector of handles to previously-created arrows ||| and/or line objects, will update the previously- ||| created arrows according to the current view –>|A|<– width and any specified properties, and will convert two-point line objects to corresponding arrows. ARROW(H) will update the arrows if the current view has changed. Root, figure, or axes handles included in H are replaced by all descendant Arrow objects.

A property list can follow any specified normal argument list, e.g., ARROW([1 2 3],[0 0 0],36,'BaseAngle',60) creates an arrow from (1,2,3) to the origin, with an arrowhead of length 36 pixels and 60-degree base angle.

The basic arguments or properties can generally be vectorized to create multiple arrows with the same call. This is done by passing a property with one row per arrow, or, if all arrows are to have the same property value, just one row may be specified.

You may want to execute AXIS(AXIS) before calling ARROW so it doesn't change the axes on you; ARROW determines the sizes of arrow components BEFORE the arrow is plotted, so if ARROW changes axis limits, arrows may be malformed.

This version of ARROW uses features of MATLAB 5 and is incompatible with earlier MATLAB versions (ARROW for MATLAB 4.2c is available separately); some problems with perspective plots still exist.

`Visualization_functions.`**`bar_wani`**(*y*, *e*, *bar_width*, *varargin*)

Draw a bar plot with error bars with some additional useful features (work with up to the 2014a matlab).

Usage

```
h = bar_wani(y, e, bar_width, varargin)
```

Inputs

y: y values for bars (row: bar grouping, column: bar values within a bar group) (e.g., if there are m bar groups and n bars for each group, y should be a m x n matrix)

e: error bars (m x n matrix)

bar_width: value for bar width between 0 and 1. This will determine the intervals between bars.

Optional Inputs Enter keyword followed by variable with values

'ylim': y axis range, (e.g., 'ylim', [-1 1])

'ytick': y tick values (e.g., 'ytick', -.002:.001:.002)

'errbar_width': the horizontal width of error bars (e.g., 'errbar_width', [-.01 .01])

'colors': bar colors: each row determines the color for each bar in order (n x 3 matrix)

'ast': put asterisks according to p values, which should be given. (e.g., 'ast', p [m x n]) *p<.05, **p<.01, ***p<.001

'btwlines': this option puts lines between bar groups. This is followed by the line style (e.g., 'btwlines', '–');

'dosave': followed by savename (e.g., 'dosave', savename)

Some advanced options

'scatter': show individual data points, which should be in cell array

'text': this will put a number for each bar (e.g., 'text', round(y) [m x n])

'ast_adj_y_pos': When the asterisk locations (on y axis) for bars with positive values are off, you can adjust it using this option

'ast_adj_y_neg': When the asterisk locations (on y axis) for bars with negative values are off, you can adjust it using this option

'ast_adj_x': When the asterisk locations (on x axis) are off, you can adjust it using this option

'bar_edgecol': You can use different colors for bar edges (col [n x 3 matrix])

'bar_edgewidth': You can change linewidths for bar edges

Output

h: graphic handles for a bar plot

Examples you can see the output in

http://wagerlab.colorado.edu/wiki/doku.php/help/core/figure_gallery :Examples:

```
% data
y = [-0.6518   -0.6934   -0.5417   -0.6496   -0.5946   -0.3839
    1.1511    0.9090    1.1681    1.2892    0.9346    1.1383];
e = [0.3226    0.2936    0.3080    0.3203    0.3368    0.3167
    0.4026    0.4088    0.4012    0.5586    0.3734    0.4257];
p = [0.0433    0.0182    0.0785    0.0426    0.0775    0.2255
    0.0042    0.0262    0.0036    0.0210    0.0123    0.0075];

col =  [0    0.1157    0.2686
        0.1157    0.2765    0.4725
        0.4843    0.1157    0.1078
        0.3667    0.4765    0.1353
        0.2765    0.1902    0.3824
        0.0922    0.4216    0.5118
        0.7941    0.3235   0];

% draw
bar_wani(y, e, .8, 'colors', col, 'errbar_width', [0 0], 'ast', p, 'ylim', [-2.5 2.5], 'ytick',
set(gca, 'ytickLabel', num2str(get(gca, 'ytick')'));
set(gcf, 'position', [1   531   399   169]);

savename = 'example_barwani.pdf';

try
    pagesetup(gcf);
    saveas(gcf, savename);
catch
```

```
        pagesetup(gcf);
        saveas(gcf, savename);
    end
```

Visualization_functions.**barplot_colored**(*data*, *varargin*)

Make a barplot of data with error bars, with colors specified by colormap or color string.

Usage

```
[h, axh]=barplot_colored(data, [optional arguments])
```

this is a good function though within-subject error bars now added; use 'within'

```
[bar_handles, axis_handle]=barplot_colored(data,varargin)
```

Input arguments Optional

'within': Do within-subject STE bars, average obs x variable interaction Loftus and Masson 1994 style.

Strings, followed by values for each:

COLOR CONTROL

'colormap': followed by colormap name to use

'colors': followed by cell array of colors per bar; supercedes colormap

Display items

- 'fontsize'
- 'title'
- 'XTickLabels'
- 'ylabel'
- 'xlabel'

Bar Locations

- 'x' : followed by x values for bars (locations)

NOTE: For this function, keywords must be even-numbered argument entries, e.g., arg 2, 4, 6. Odd argument entries are values.

For example: This works, and you need the extra empty arg after 'within'

```
[h1, s1] = barplot_colored(pexp1, 'within', ' ', 'title', 'Pattern expression', 'XTickLabels', d
```

You can assign arbitrary colors to bars by setting the colormap:

```
[h, s] = barplot_colored([corr_temp corr_rep]);
cm = [1 .5 0; .5 0 1];
colormap(cm)
```

Example: A grouped barplot

```
dat = rand(20, 4);
create_figure('bars');
[h1, s1] = barplot_colored(dat, 'x', [1 2 4 5]);
% set(h2, 'BarWidth', .9)
```

Change colormap:

```
[h1, s1] = barplot_colored(dat, 'x', [1 2 4 5], 'colormap', 'summer');
```

Enter values:

```
colors = {[.8 .25 .25] [.8 .5 .25] [.4 .5 .8] [.25 .25 .9]};
[h1, s1] = barplot_colored(dat, 'x', [1 2 4 5], 'colors', colors);
```

Set X Tick Label:

```
[h1, s1] = barplot_colored(dat, 'XTicklabels', {'A' 'B' 'C' 'D'});
```

See also: barplot_columns, lineplot_columns

Visualization_functions.**barplot_columns** (*dat*, *varargin*)

Usage

```
[axishandle,adjusted data,x-data, barhandle] = barplot_columns(dat, [other optional arguments])
```

This function makes a barplot of columns of data, with standard error bars. Optional arguments include removing continuous covariates before plotting, robust (IRLS) estimation of means and correlations with covariates, and within-subject error bars based on the subject x condition interaction (overall), which is not quite the standard error contrasts of interest, but is the standard error for a 1-way repeated measures ANOVA.

plots circles around points at z >= 1.96

plots individual points, unless you enter 4th argument

if dat is a cell array, each entry becomes one "bar". Useful if n observations is different for each column.

Examples Just plot means and SE

```
h = barplot_columns(tmp,'Cluster 1',[],1);
```

Optional arguments

1. Title for figure
2. covariates
3. **String Arguments**
   - 'nofig' : do not make figure
   - 'noind' : do not plot individual scores
   - 'plotout': circle potential outliers at z>1.96 in red
   - 'dorob' : do robust IRLS means and correlations
   - 'dolines' : plot lines showing individual effects
   - **'within'** [within-subjects standard errors, followed by contrast] matrix
   - '95CI' : error bars are 95% CI instead of SE
   - 'line' : Make line plot instead of bar plot
   - 'number' : plot case numbers instead of points
   - 'x' : followed by x-axis values for bars
   - **'color'** [followed by color for bars (text: 'r' or [r g b]) OR] cell array with names of colors cell for each line/bar

- 'violin': add violin plot to each bar, with data points

Examples

```
barplot_columns(ctmp,'RT effects by Switch Type',overall_sw,'nofig','dorob')
```

Standard Errors ARE NOT Adjusted for covariate, right now.

Example: within-subjects std. errors

```
barplot_columns(dat, 'Means', [], 'nofig', 'within', c);
```

The example below uses color, width, and xposition arguments to make a grouped

```
barplot showing effects for two groups:
exp_dat = EXPT.error_rates(EXPT.group==1,:);
control_dat = EXPT.error_rates(EXPT.group==-1,:);
barplot_columns(exp_dat, 'Error rates', [], 'nofig', 'noind', 'color', 'r','width', .4);
barplot_columns(control_dat, 'Error rates', [], 'nofig', 'noind', 'color', 'b','width', .4, 'x',
set(gca, 'XLim', [0 10], 'XTick', 1:9)

barplot_columns(nps_by_study, 'NPS by study', [], 'doind', 'colors', mycolors, 'nofig');

create_figure('example_plots', 1, 4);

Y{:,1} = rand(20,1);
Y{:,2} = rand(100,1);

[h, L, MX, MED, bw, F, U] = violinplot(Y,'facecolor',[1 .5 0; 0 .5 1],'edgecolor',[1 .5 0; 0 .5
title('Violinplot.m', 'FontSize', 16)

subplot(1, 4, 2)
barplot_columns(Y, 'nofig')
title('barplot\_columns.m default', 'FontSize', 16)

subplot(1, 4, 3)
barplot_columns(Y, 'nofig', 'violin', 'colors', {[1 .5 0] [0 .5 1]})
title('barplot\_columns.m colored', 'FontSize', 16)

subplot(1, 4, 4)

Y{:,1} = randn(50,1) + 5;
Y{:,2} = Y{1} + .3 * randn(50,1) + 3;

barplot_columns(Y, 'nofig', 'noviolin', 'colors', {[1 .5 0] [0 .5 1]}, 'dolines')
title('barplot\_columns.m parallel coords', 'FontSize', 16)
```

See also: lineplot_columns, barplot_colored, line_plot_multisubject, violinplot ..

Defaults

Visualization_functions.**barplot_columns2**(*dat*, *plottitle*, *varargin*)

Usage

```
[axishandle,adjusted data,x-data,barhandle] = barplot_columns(dat,title,[options])
```

Makes a barplot of columns of data, with standard error bars. Optional arguments include removing continuous covariates before plotting, robust (IRLS) estimation of means and correlations with covariates, and within-subject error bars based on the subject x condition interaction (overall), which is not quite the standard error contrasts of interest, but is the standard error for a 1-way repeated measures ANOVA.

plots circles around points at z >= 1.96 plots individual points, unless you enter 4th argument

if dat is a cell array, each entry becomes one "bar". Useful if n observations is different for each column.

Examples Just plot means and SE

```
h = barplot_columns(tmp,'Cluster 1',[],1);
```

Options

'cov': followed by matrix of covariates

'labels': followed by cellstring of bar labels

'xlabelslant': followed by number of degrees to slant bar labels (DEFAULT: 45)

'nofig': do not make figure

'ind': plot individual scores on top of bars

'plotout': circle potential outliers at z>1.96 in red

'robust': do robust IRLS means and correlations

'indlines': plot lines showing individual effects

'within': within-subjects standard errors, followed by contrast matrix

'line': Make line plot instead of bar plot

'number': plot case numbers instead of points

'x': followed by x-axis values for bars

'color': followed by color for bars (text: 'r' or [r g b]) OR cell array with names of colors cell for each line/bar

'denan': remove rows that have NaNs

Examples

```
barplot_columns(ctmp,'RT effects by Switch Type',overall_sw,'nofig','robust')
```

Standard Errors ARE NOT Adjusted for covariate, right now.

Example: within-subjects std. errors

```
barplot_columns(dat, 'Means', 'nofig', 'within', 'ind');
```

The example below uses color, width, and xposition arguments to make a grouped

```
barplot showing effects for two groups:
exp_dat = EXPT.error_rates(EXPT.group==1,:);
control_dat = EXPT.error_rates(EXPT.group==-1,:);
barplot_columns(exp_dat, 'Error rates', 'nofig', 'color', 'r', 'width', .4);
barplot_columns(control_dat, 'Error rates', 'nofig', 'color', 'b', 'width', .4, 'x', (1:9)+.5);
set(gca, 'XLim', [0 10], 'XTick', 1:9)
```

Visualization_functions.**barplot_columns3**(*dat*, *plottitle*, *varargin*)

Usage

```
[axishandle,adjusted data,x-data,barhandle] = barplot_columns(dat,title,[options])
```

Makes a barplot of columns of data, with standard error bars. Optional arguments include removing continuous covariates before plotting, robust (IRLS) estimation of means and correlations with covariates, and within-subject error bars based on the subject x condition interaction (overall), which is not quite the standard error contrasts of interest, but is the standard error for a 1-way repeated measures ANOVA.

plots circles around points at z >= 1.96

plots individual points, unless you enter 4th argument

if dat is a cell array, each entry becomes one "bar". Useful if n observations is different for each column.

Examples Just plot means and SE

```
h = barplot_columns(tmp,'Cluster 1',[],1);
```

Options

'cov': followed by matrix of covariates

'labels': followed by cellstring of bar labels

'nofig': do not make figure

'ind': plot individual scores on top of bars

'plotout': circle potential outliers at z>1.96 in red

'robust': do robust IRLS means and correlations

'indlines': plot lines showing individual effects

'within': within-subjects standard errors, followed by contrast matrix

'line': Make line plot instead of bar plot

'number': plot case numbers instead of points

'x': followed by x-axis values for bars

'color': followed by color for bars (text: 'r' or [r g b]) OR cell array with names of colors cell for each line/bar

'cons': followed by contrastXweights matrix

To convert from long form to wide form

'subcol': column numbers with subject numbers

'propcols': vector of column numbers with properties

'propnames': cell array of names of properties

Examples

```
barplot_columns(ctmp,'RT effects by Switch Type',overall_sw,'nofig','robust')
```

Standard Errors ARE NOT Adjusted for covariate, right now.

Example: within-subjects std. errors

```
barplot_columns(dat, 'Means', 'nofig', 'within', 'ind');
```

The example below uses color, width, and xposition arguments to make a grouped

```
barplot showing effects for two groups:
exp_dat = EXPT.error_rates(EXPT.group==1,:);
control_dat = EXPT.error_rates(EXPT.group==-1,:);
barplot_columns(exp_dat, 'Error rates', 'nofig', 'color', 'r', 'width', .4);
barplot_columns(control_dat, 'Error rates', 'nofig', 'color', 'b', 'width', .4, 'x', (1:9)+.5);
set(gca, 'XLim', [0 10], 'XTick', 1:9)
```

Visualization_functions.**barplot_grouped**(*dat*, *X*, *xnames*, *seriesnames*, *varargin*)

Usage

```
han = barplot_grouped(dat,X,xnames,seriesnames, [optional args]);
```

Inputs

dat: n x 4 data matrix (2 x 2 grouping only for now, but easy to expand later)

X: covariates, no intercept; will be centered by this function

xnames: x-axis labels

seriesnames: series labels

First two bars are group, and last two bars are group

Optional Inputs (keywords)

'within': within-error flag. 1 = errors based on subject x condition interaction

'stars': put stars for significance on graph (default)

'nostars': do not plot stars

'bars': followed by number of bars in group

'pvals': followed by matrix of p-values (for stars; will calculate if missing)

'inputmeans': input means and errors in first 2 inputs rather than data and X

'colors': followed by colors, e.g., mycol = {[1 0 0] [0 1 0] [1 0 1] [1 1 0] [0 0 1]}

Visualization_functions.**barplotter**(*data*, *varargin*)

Usage

```
h = barplotter(data)
```

Creates a barplot out of the columns or elements (if data is a cell array containing vectors) of data.

```
h =barplotter(..., 'groups', grouping)
```

grouping must have the same number of columns or elements (if data is a cell array) as data, and must consist of integers beginning with 1 and ending with the total number of groups. Data belonging to the same group will be 'clustered' together in the plot. Ommitting groups (eg, [1 1 3 3]) will create extra spacing between groups.

```
h =barplotter(..., 'std')
```

overrides the default behavior and plots standard deviation bars instead of standard error bars

```
h =barplotter(..., 'CI', alpha)
```

this will override the default behavior (as well as 'std') and plot 1-alpha confidence intervals.

```
h =barplotter(..., 'labels', labels)
```

labels must be a cell array of strings with the same number of elements as data has columns or elements. The x-axis will be labeled with these.

```
h =barplotter(..., 'label_groups')
```

Applies labels to groups instead to individual bars

```
h =barplotter(..., 'legend', names)
```

Plots a legend in the figure, with labels corresponding to the elements of the cell vector of strings, names.

```
h =barplotter(..., 'plegend', names, p)
```

As 'legend', above, but p is a vector of the barplots to include in the legend (e.g. if you plotted 6 bars and p = [1 3], only the first and third bar would be included in the legend.

```
h =barplotter(..., 'PlotLineHor', value)
```

Plots a horizontal line at the Y value indicated.

h = barplotter(..., 'LinePlot', colors, styles, markers)

Changes the graph from a bar graph to a line graph. Setting groups will cause one line to be drawn for each group.

h = barplotter(..., 'ErrorWidth', errorwidth)

Sets the line weight (in points) for error bars

```
h =barplotter(..., 'PropertyName', PropertyValue)
```

Properties correspond to various Matlab figure properties, as appropriate. Currently supported properties (more to be added) are:

- 'Title'
- 'XLabel'
- 'YLabel'
- 'YLim'
- 'XLim'
- 'YTick'
- 'YTickLabel'
- 'YMinorTick'
- 'FontSize'
- 'xFontSize' %for xlabel
- 'yFontSize' %for ylabel
- 'tFontSize' %for title
- **'FaceColor' %note that you may specify a matrix of 3-element RGB vectors,** rather than a single vector. Barplotter will then cycle the rows of the matrix until all bars have been drawn.
- 'Colormap'
- 'GridLineStyle' % - | - -| {:} | -. | none

- 'TickDir' % in or out
- 'MarkerSize'
- 'LineWidth'

Visualization_functions.**canlab_force_directed_graph**(*activationdata*, *varargin*)

Creates a force-directed graph from a set of variables, and plots clusters on 3-D brain as well if entered. Requires matlab BGL toolbox.

Usage

```
canlab_force_directed_graph(activationdata OR connection matrix, ['cl', cl])
```

Inputs

activationdata: observations x variables matrix of data to be inter-correlated

OR

signed, thresholded connection matrix to be used (e.g., thresholded t-values from multi-subject group analysis

Optional Inputs Enter keyword followed by variable with values

'cl': followed by clusters or region structure with brain clusters

'threshtype': followed by threshold type; 'bonf' is option now

'connectmetric':

'sizescale': Followed by values to use in sizing of nodes on graph

'setcolors': Cell array of colors for each group, [1 x g]

'rset': Cell of vectors, with indices (integers) of member elements in each group, [1 x g] cell rset can ALSO be a vector of integers, i.e., output from clusterdata

'names':

'namesfield':

Output

stats: structure with descriptive statistics, including betweenness-centrality, degree of each node

Examples

```
[stats, handles] = canlab_force_directed_graph(activationdata, 'cl', cl, 'namesfield', 'shorttit
[stats, handles] = canlab_force_directed_graph(activationdata, 'cl', cl, 'namesfield', 'shorttit
[stats, handles] = canlab_force_directed_graph(activationdata, 'cl', cl, 'namesfield', 'shorttit
```

Visualization_functions.**canlab_results_fmridisplay**(*input_activation*, *varargin*)

Usage

```
canlab_results_fmridisplay(input_activation, [optional inputs])
```

purpose: This function display fmri results.

Input

input_activation: nii, img,

This image has the blobs you want to display. You can also enter a cl "clusters" structure or "region" object.

you can also get a thresholded image like the examples used here from a number of places - by thresholding your results in SPM and using "write filtered" to save the image, by creating masks from meta-analysis or anatomical atlases, or by using mediation_brain_results, robust_results_threshold, robust_results_batch_script, threshold_imgs, or object oriented tools including fmri_data and statistic_image objects.

Optional Inputs

'noblobs': do not display blobs

'nooutline': do not display blob outlines

'addmontages': when entering existing fmridisplay obj, add new montages

'noremove': do not remove current blobs when adding new ones

'outlinecolor: followed by new outline color

'splitcolor': followed by 4-cell new split colormap colors (help fmridisplay or edit code for defaults as example)

'montagetype': 'full' for full montages of axial and sagg slices.

'compact' [default] for single-figure parasagittal and axials slices.

'compact2': like 'compact', but fewer axial slices.

'noverbose': suppress verbose output, good for scripts/publish to html, etc.

Other inputs to addblobs (fmridisplay method) are allowed, e.g., 'cmaprange', [-2 2], 'trans'

See help fmridisplay e.g., 'color', [1 0 0]

You can also input an existing fmridisplay object, and it will use the one you have created rather than setting up the canonical slices.

Example Script

```
input_activation = 'Pick_Atlas_PAL_large.nii';

% set up the anatomical underlay and display blobs
% (see the code of this function and help fmridisplay for more examples)

o2 = canlab_results_fmridisplay(input_activation);

%% ========== remove those blobs and change the color ==========

cl = mask2clusters(input_activation);
removeblobs(o2);
o2 = addblobs(o2, cl, 'color', [0 0 1]);

%% ========== OR

r = region(input_activation);
o2 = removeblobs(o2);
o2 = addblobs(o2, r, 'color', [1 0 0]);

%% ========== Create empty fmridisplay object on which to add blobs:
o2 = canlab_results_fmridisplay
```

```
    o2 = canlab_results_fmridisplay([], 'compact2', 'noverbose');

    %% ========== If you want to start over with a new fmridisplay object,
    % make sure to clear o2, because it uses lots of memory

    % This image should be on your path in the "canlab_canonical_brains" subfolder:

    input_activation = 'pain-emotion_2s_z_val_FDR_05.img';
    clear o2
    close all
    o2 = canlab_results_fmridisplay(input_activation);

    %% ========== save PNGs of your images to insert into powerpoint, etc.
    % for your paper/presentation

    scn_export_papersetup(400);
    saveas(gcf, 'results_images/pain_meta_fmridisplay_example_sagittal.png');

    scn_export_papersetup(350);
    saveas(gcf, 'results_images/pain_meta_fmridisplay_example_sagittal.png');

    Change colors, removing old blobs and replacing with new ones:
    o2 = canlab_results_fmridisplay(d, o2, 'cmaprange', [.3 .45], 'splitcolor', {[0 0 1] [.3 0 .8] [
```

Visualization_functions.**cl_line_plots**(*clusters*)

Usage

```
    cl_line_plots(clusters)
```

purpose: ???

Input clusters

cluster object

Output lineH

handle

Visualization_functions.**cl_overlap**(*in1*, *in2*)

Usage

```
    cl_overlap(in1,in2)
```

purpose: compute overlap of two clusters

Input in1

cluster object

in1

cluster object

Output ov_mat

matrix of overlaps

Visualization_functions.**close_non_spm_graphics_figures**()

Usage

```
close_non_spm_graphics_figures()
```

purpose: closes all figure windows that are not the SPM orthviews window

Input

Output

Visualization_functions.**cluster_barplot**(*P*, *clusters*, *varargin*)

Usage

```
[clusters, subcl] = cluster_barplot(P, clusters, varargin)
```

this function not only plots, but separates subclusters using pca / cluster_princomp.m based on pattern across all conditions, covariance (not correlation),

Inputs

P: cell array of strings containing image file names (data extracted from these)

clusters:

[opt] - 'subclusters': to get sub-clustering based on pca and clustering of voxels

[opt] - cell array: of strings with condition names

[opt] - 'split': value is 1: to split into 2 plots (first half and last half of P)

value is 2: to plot individual subjects over bars

[opt] - 'center': center parameter values in plot (subtract row means) this gives closer to correct "within subjects" error bars and may be used when the overall parameter values have no meaning

[opt] - 'covs':

followed by between-subject covariates (e.g., behavioral regressors) plots remove these before plotting means and std. errors

[opt] - 'max': to make plots based on max z-values within region for each dataset P not compatible with 'split' and 'center' (ignores these commands) right now, special for inhib - see also inhib2_cluster_barplot (good function)

[opt] - 'indiv': to threshold based on individual t-statistics

FOLLOW with cell array of t-images – usually, there will be one cell, with images for each subject in rows, to define voxels for each ss.

BUT Tnames can be the same length as contrast images, one t-img per subject per contrast, if desired.

Outputs

clusters: clusters struture, with BARPLOT substructure added

subcl: substructure contains data extracted and image file names

This program uses XYZmm millimeter coordinates in clusters to find voxels So clusters and data files may have different dimensions.

Examples

```
cluster_barplot(EXPT.SNPM.P(4:6), clusters(2:3))
cluster_barplot(EXPT.SNPM.P(7:12), clusters(2:3), {'ObjE' 'AttE' 'InteractE' 'ObjI' 'AttI' 'Inte
[clusters, subclusters] = cluster_barplot(EXPT.SNPM.P(17:24), clusters, 'subclusters', 'split')
```

```
    RS2_8vs2_placeboCP = cluster_barplot(EXPT.SNPM.P([8 10 12 14
    16]), RS2meta, 'indiv', T);
```

See Also mask2clusters.m, a simpler version that just extracts clusters from a mask file.

Visualization_functions.**cluster_cutaways**(*clusters*, *textprefix*, *mycolor*, *whichcuts*, *varargin*)

groups clusters to image on brains by the first letter in whichcuts

Usage

```
    O = cluster_cutaways(clusters,textprefix,mycolor,whichcuts,[coords for center of cuts],[revx])
```

groups clusters to image on brains by the first letter in whichcuts therefore, if you enter 'y' for whichcuts, the program will select clusters with similar y values (w/i 15 mm) and image them on the same brain. you need files called brain_render_T1.img/hdr on the path these should be scalped anatomicals for the brain image.

you would also need 'brain_render_T1.mat' for the transparent brain surface, if you changed this script to get the transparent brain surface.

if it cuts from the wrong side, try 'revx' as input into tor_3d.m or enter anything as 2nd var arg.

Inputs

clusters: cluster object to display on brain cuts

textprefix:

prefix for saving images to files

mycolor:

cluster colors - 3 el. vector, single letter, or string of letters

enter letter or letter string in single quotes. e.g., O.clcol = 'yrgb';

whichcuts: letter string (no quotes) - which axes to cut along - xyz are choices

Optional Inputs **bestc** best cut, 3-element vector of coordinates, default [0 0 0]

revx text string to reverse x cut direction, enter 1 to do it.

Outputs

O: output returned from renderCluster_ui.m

Examples

```
    O = cluster_cutaways(clusters,'myoutname','y','yzx',[0 0 0],[revx])
```

Uses renderCluster_ui.m

Visualization_functions.**cluster_image_shape**(*cl*, *varargin*)

Usage

```
    [hpatch, cl] = cluster_image_sphere(cl or [k x 3 list of mm coords], varargin)
```

Images spheres at cluster centers Combine with addbrain and cluster_tools('connect3d') or cluster_nmdsfig_glassbrain to get 3-D plots of connected blobs

Optional Inputs {'color', 'colors'}, mycolor = varargin{i+1};

'radius', myradius = varargin{i+1};

Outputs

hpatch: patch handles

cl: new cl with sphere coordiates in XYZmm and XYZ

Examples

```
function [hpatch, cl] = cluster_image_sphere(cl)

% With optional arguments:
[hpatch, cl] = cluster_image_sphere(cl, 'color', 'b', 'radius', 10)
[hpatch, cl] = cluster_image_sphere(cl, 'color', {'r' 'g' 'b' etc}, 'radius', 10)
```

Example Given an MNI coordinate, plot a sphere on a brain surface

```
my_mm_coord = [40, 46, 22]';
create_figure('surface')
cl = [];
cl.XYZmm = my_mm_coord;
cl.mm_center = my_mm_coord';
V = spm_vol(which('brainmask.nii'));
cl.M = V.mat;
[hpatch, cl] = cluster_image_sphere(cl, 'color', 'g', 'radius', 10)
p = addbrain;
set(p, 'FaceAlpha', 1);
axis image
view(135, 30);
lighting gouraud;
lightRestoreSingle;
material dull;

% Turn xyz mm coordinates into clusters and image them

my_mm_coord = [40 46 22; 50 26 40; 45 36 50; 60 12 0]
[hpatch, cl] = cluster_image_sphere(my_mm_coord, 'color', 'b', 'radius', 4);
```

Visualization_functions.**cluster_image_sphere**(*cl*, *varargin*)

Usage

```
[hpatch, cl] = cluster_image_sphere(cl or [k x 3 list of mm coords], varargin)
```

Images spheres at cluster centers Combine with addbrain and cluster_tools('connect3d') or cluster_nmdsfig_glassbrain to get 3-D plots of connected blobs or surface_cutaway

Optional Inputs {'color', 'colors'}, mycolor = varargin{i+1};

'radius', myradius = varargin{i+1};

Outputs patch handles, and new cl with sphere coordiates in XYZmm and XYZ

Example Usage

```
function [hpatch, cl] = cluster_image_sphere(cl)

% With optional arguments:
[hpatch, cl] = cluster_image_sphere(cl, 'color', 'b', 'radius', 10)
[hpatch, cl] = cluster_image_sphere(cl, 'color', {'r' 'g' 'b' etc}, 'radius', 10)
```

Examples Given an MNI coordinate, plot a sphere on a brain surface

```
my_mm_coord = [40, 46, 22]';
create_figure('surface')
cl = [];
cl.XYZmm = my_mm_coord;
cl.mm_center = my_mm_coord';
V = spm_vol(which('brainmask.nii'));
cl.M = V.mat;
[hpatch, cl] = cluster_image_sphere(cl, 'color', 'g', 'radius', 10)
p = addbrain;
set(p, 'FaceAlpha', 1);
axis image
view(135, 30); lighting gouraud; lightRestoreSingle; material dull;
```

Example: Turn xyz mm coordinates into clusters and image them

```
my_mm_coord = [40 46 22; 50 26 40; 45 36 50; 60 12 0]
[hpatch, cl] = cluster_image_sphere(my_mm_coord, 'color', 'b', 'radius', 4);
```

Visualization_functions.**cluster_orthviews**(*varargin*)

Usage

```
cluster_orthviews(inputs in any order)
```

This function uses spm_orthviews to display activation blobs from a clusters structure on a canonical structural brain image. Multiple clusters may be plotted in multiple colors, and blobs may be added to an existing orthviews figure.

Inputs clusters structures

colors cell array, e.g., {[0 0 1] [1 0 0] [0 1 0]} if no colors cell array, uses Z- or t-scores to color map

Optional Inputs

'add': to suppress making new orthviews

'copy': to copy to smaller subfigure with empty axis beside it

'unique': to display in unique colors for each cluster

'overlay': followed by name of image, to use a custom anatomical overlay

'bivalent': to plot increases in solid colors specified by colors cell {1} and {2}

OR, if no colors entered, use hot/cool map in spm_orthviews_hotcool_colormap

Options if you specify colors (addColouredBlobs)

'trans': to make blobs transparent

'solid': to make them solid

Options for using spm's color map (addBlobs)

'blue': display in blue split color map instead of default red-yellow

'handle': followed by integer for which orthviews window to use (default = 1)

Visualization_functions.**cluster_orthviews_classes**(*cl*, *classes*, *overlay*, *myview*, *domontage*, *varargin*)

Usage

```
colors2 = cluster_orthviews_classes(cl,classes,overlay,myview,domontage, [orthview axis], [color
```

Makes montage and cluster orthviews of classes of clusters in different colors (hard coded colors right now).

Inputs

cl: clusters structure with regions

classes: integers with classes (i.e., networks) to be color-coded on plot

overlay: anatomical underlay image

myview: if entered, shows centers on plot

domontage: if non-zero, make montages of networks

[orthview axis]: Optional; integer for which orthviews axis to use, 1:kk

Output Cell of colors, in order, for use in other functions

Examples

```
classes = c.ClusterSolution.classes;
overlay = EXPT.overlay;
cluster_orthviews_classes(cl,c.ClusterSolution.classes, EXPT.overlay, 'saggital', 0);
cluster_orthviews_classes(cl,c.ClusterSolution.classes, EXPT.overlay, 'axial', 0);
cluster_orthviews_classes(cl,c.ClusterSolution.classes, EXPT.overlay, 'coronal', 0);
cluster_orthviews_classes(cl,c.ClusterSolution.classes, EXPT.overlay, [], 0);
```

Visualization_functions.**cluster_orthviews_montage**(*spacing*, *myview*, *varargin*)

Usage

```
[slices_fig_h, slice_mm_coords, slice_vox_coords, axis_handles] = cluster_orthviews_montage(spac
```

Runs on top of spm_orthviews, creates montages from current orthviews display, whatever it is

Examples

```
cluster_orthviews_montage(6, 'coronal');   % 6 mm spacing
cluster_orthviews_montage(10, 'sagittal', 'range', [-10 10]); % 10 mm spacing sag view with only
cluster_orthviews_montage(12, 'axial');    % 12 mm spacing, axial view
```

additional options: enter AFTER overlay:

- 'whichorth', whichorth = varargin{i+1}; varargin{i:i+1} = [];
- 'onerow', doonerow = 1; varargin{i} = [];
- 'range', followed by [min max] mm coords for slices
- 'xhairs', xhairs = 1; turn on cross-hairs on slice plot

used in cluster_orthviews_classes

Visualization_functions.**cluster_orthviews_overlap**(*mask1*, *mask2*, *varargin*)

Plot two clusters on the orthviews, and their intersections for activations and deactivations in intermediate colors

Usage

```
cluster_orthviews_overlap(mask1, mask2, [colors cell])
```

Inputs **mask1** path to nifit image

mask2 path to nifit image

Optional Inputs **colors** cell array of RGB colors like: {[1 0 0] [0 0 1] [1 1 0] [0 1 1]}

Visualization_functions.**cluster_orthviews_overlap2**(*masks*, *varargin*)

Plot blobs on the orthviews, and their intersections for activations and deactivations in intermediate colors

Usage

```
cluster_orthviews_overlap2(masks, ['colors', colors cell], ['surface'], ['negative'] )
```

Positive effects only!

:Inputs

masks image_vector or fmri_data object

:Optional Inputs

colors

'colors',{[RGB],[RGB],..} cell array of RGB triplets

surface 'surface',[0/1] default =1

negative 'negative' string to only use negative clusters, otherwise only positive clusters

nomontage 'nomontage',[0/1] don't do montage, default do a montage

Visualization_functions.**cluster_orthviews_overlap_3colors**(*mask1*, *mask2*, *mask3*, *varargin*)

Plot three clusters on the orthviews, and their intersections for activations and deactivations in intermediate colors

Usage

```
cluster_orthviews_overlap_3colors(mask1, mask2, mask3, ['colors', colors cell], ['surface'] )
```

Positive effects only!

Inputs mask1 mask2 mask3 are either image names (preferred!) or clusters structures. for clusters structures, need to add .dim field and you need the 2010 object-oriented code in the canlab repository.

Visualization_functions.**cluster_orthviews_showcenters**(*cl*, *myview*, *overlay*, *xhairs*, *sliceorder*, *bgcolor*, *varargin*)

Usage

```
slices_fig_h = cluster_orthviews_showcenters(cl, myview, [overlay], [xhairs], [order slices flag

cluster_orthviews_showcenters(cl, 'coronal');
cluster_orthviews_showcenters(cl, 'sagittal');
cluster_orthviews_showcenters(cl, 'axial');
```

used in cluster_orthviews_classes

Visualization_functions.**cluster_surf**(*varargin*)

Surface plot of clusters on a standard brain

Usage

```
[suface_handle,colorchangestring] = cluster_surf(varargin)
```

Inputs

CLUSTERS: clusters structures, as created in tor_extract_rois.m

COLORS:

cell array of colors for each cluster: {[1 0 0] [0 1 0] [0 0 1]}

if number of colors specified is greater than number of clusters structures entered, n+1 and n+2 colors are overlap of 2 and overlap of all clusters, respectively.

SURFACE MAT FILE:

file name of mat file containing brain surface vertices and faces

as created with isosurface.

SPECIAL SURFACE KEYWORDS: special string: 'bg' 'hipp' (hcmp,thal,amy) number of mm to plot from surface (mmdeep)

- Special keywords for sets of surfaces are: left, right, bg, limbic, cerebellum, brainstem
- Other keywords: 'left' 'right' 'amygdala' 'thalamus' 'hippocampus' ' 'midbrain' 'caudate' 'globus pallidus' 'putamen' 'nucleus accumbens'

  'hypothalamus' 'cerebellum'

EXISTING SURFACE HANDLE(S): handles for surface patches, created, e.g., with addbrain.m. This lets you be very flexible in the surfaces you image onto.

'colorscale':

This scales colors by Z-scores of voxels if used

- Uses input color, unless also used with 'heatmap'
- Z scores should be in ROW vector
- use with 'normalize' to scale Z-scores between -1 and 1
- will also create transparent effects, mixing blob color with existing surface color in linear proportion to Z-scores

'heatmap':

Map Z-scores to surface colors

- Used WITH or instead of 'colorscale'
- Blobs can have a range of colors
- Use with REFERENCE RANGE option below to control scale
- solid colors entered as input will be ignored
- use with 'colormaps' option below to be flexible in which color maps you apply.
- if 'colorscale' is also used, will produce transparent blobs.

REFERENCE RANGE: reference Z-scores range, [zmin_act zmax_act zmax_negact zmin_negact], e.g., [0 5 -5 0], use only with 'heatmap' option to get refZ from clusters, try:

```
clZ = cat(2,clusters.Z);
refZ = [min(clZ(clZ > 0)) max(clZ) min(clZ(clZ < 0)) min(clZ)];
```

'colormaps':

- followed by custom [colors x 3] matrices for positive colors and negative colors.

- matlab can create some: e.g., colormap summer, jet, etc. others can be created with colormap_tor.m

color [0 1 1] (cyan) is reserved for the overlap color btwn cluster sets.

Examples

```
P = 'C:\tor_scripts\3DheadUtility\canonical_brains\surf_single_subj_T1_gray.mat';
cluster_surf(tcl,acl,P,10,{[0 1 0] [1 0 0]},'colorscale','heatmap')

or P = h (surface handle) to use current surface in figure, and refZ
cluster_surf(tcl,acl,h,[3 5 -5 -3],10,{[0 1 0] [1 0 0]},'colorscale','heatmap')
```

More examples

```
cluster_surf(cl,2,'heatmap');     % brain surface.  vertices colored @2 mm
cluster_surf(cl,2,'bg','heatmap');    % heatmap on basal ganglia
cluster_surf(cl,5,'left','heatmap');  % heatmap on left surface @5 mm
cluster_surf(cl,2,'right','heatmap');

% A multi-color, multi-threshold display on the cerebellum
colors = {[1 1 0] [1 .5 0] [1 .3 .3]};
tor_fig;
sh = cluster_surf(cl{3},colors(3),5,'cerebellum');
cluster_surf(cl{2},colors(2),5,sh);
cluster_surf(cl{1},colors(1),5,sh);

% Custom colormaps:
create_figure('Brain Surface'); cluster_surf(cl, 2, 'heatmap','left');

poscm = colormap_tor([.2 .2 .4], [1 1 0], [.9 .6 .1]);  %slate to orange to yellow
negcm = colormap_tor([0 0 1], [0 .3 1]);  % light blue to dark blue
create_figure('Brain Surface'); cluster_surf(cl, 2, 'heatmap', 'colormaps', poscm, negcm, 'left'

% Single-color transparent map (green):
cluster_surf(cl, 2, {[0 1 0]}, 'colorscale', p3(2), 'normalize');
```

See Also addbrain, img2surf.m, surface() methods for objects, cluster_cutaways

Visualization_functions.**cluster_surf_batch**(*varargin*)

Usage

```
surf_handles = cluster_surf_batch(varargin)
```

Examples

```
% Single-map visualization
P2 = threshold_imgs('rob_tmap_0001.img',tinv(1-.005,12),15,'pos');
cluster_surf_batch(P2);
surf_handles = cluster_surf_batch(cl,{[1 0 0]},cl2);

% Two maps with overlap
surf_handles = cluster_surf_batch(cl,{[1 0 0] [0 1 0] [1 1 0]},cl2);
```

Visualization_functions.**cluster_surf_batch2**(*varargin*)

Usage

```
surf_handles = cluster_surf_batch(varargin)
```

Examples

```
% Single-map visualization
P2 = threshold_imgs('rob_tmap_0001.img', tinv(1-.005, 12), 15, 'pos');
cluster_surf_batch(P2);
surf_handles = cluster_surf_batch({cl cl2}, {[1 0 0]});

% Two maps with overlap
surf_handles = cluster_surf_batch(cl, {[1 0 0] [0 1 0] [1 1 0]}, cl2);
```

Visualization_functions.**colormap_tor**(*lowcolor*, *hicolor*, *varargin*)

Usage

```
newcolormap = colormap_tor(lowcolor, hicolor, [midcolor], [midcolor2], etc.)
```

Create a new colormap of your choosing.

Examples

```
colormap_tor([.2 .2 .6], [1 1 0]);  % slate to yellow
colormap_tor([.9 .5 .2], [1 1 0]);  % orange to yellow
colormap_tor([.8 .1 .1], [1 1 0], [.9 .6 .1]);  %red to orange to yellow
colormap_tor([.2 .2 .4], [1 1 0], [.9 .6 .1]);  %slate to orange to yellow
```

Visualization_functions.**compare_filtered_t**(*anatP*, *varargin*)

Usage

```
compare_filtered_t(anatP,P1,P2, etc...)
```

Examples

```
compare_filtered_t([],'rob_tmap_filtered_0001.img','rob_tmap_filtered_0002.img')

Threshold spm T images and display them together in SPM orthviews
threshold_spm_t(.005,22,0,'pos')
compare_filtered_t([],'rfx0009/spmT_filtered_0002.img','rfx0011/spmT_filtered_0002.img', ...
'rfx0013/spmT_filtered_0002.img','rfx0015/spmT_filtered_0002.img','rfx0017/spmT_filtered_0002.im
```

Visualization_functions.**compare_slice**(*ovlP*, *sxyz*, *sz*, *VOL*, *varargin*)

Usage

```
compare_slice(ovlP,sxyz,sz,VOL)
```

Inputs

ovlP: is name of overlay image file (anatomical)

sxzy: is a cell array, where each cell is a list of

VOXEL coordinates in 3 x n matrix

sz: is a cell array, where each cell contains the z values corresponding to sxyz. This is used to create pseudocolor on the plots.

VOL: is a structure containing the field M, where M is the SPM99-style mapping matrix from voxel to mm space. This is used to map sxyz values onto the overlay image.

VOL must also contain field dim, which has the voxel dims of the results image for all sxyz. All sxyz must have the same dimensions!

`Visualization_functions.`**`conf_region`**(*X*, *varargin*)

Usage

```
[ax,ci,cen,ub,lb,S,e,lam,Fm,Fc,F,pval, msb] = conf_region(X,[doplot])
```

alternative conf region multivariate based on Johnson & Wichern, 4th ed., p. 236 2D (3D)

Outputs

ax: axes of confidence region, scaled to be half-length of confidence hyperellipsoid

ci: length of axes of conf region (diagonal matrix), axes in reverse order of importance as in ouput of eig

cen: center of region (means of variables)

ub: upper boundary coordinates for region, rows are dims (vars), cols index coordinates last coordinate is on axis of greatest variation ("reverse order"), from output of eig

lb: lower boundary coordinates to plot, try: plot(ub(1,:),ub(2,:),'bo'); hold on; plot(lb(1,:),lb(2,:),'ro')

S: covariance matrix

e: eigenvectors

lam: eigenvalues

Fm: degrees of freedom multiplier, p(n-1) / n(n-p)

Fc: critical F value based on alpha level

F: F value for test - probably not quite right

pval: p value for test - probably not quite right

To plot

Either enter 1 or color as a 2nd argument, or do it yourself:

```
[ax,ci,cen,ub,lb,S,e,lam] = conf_region(X);
b = pinv([X(:,1) ones(size(X,1),1)] ) * X(:,2);
theta = atan(b(1)) - pi/2;
[h,h2] = plot_ellipse(cen(1),cen(2),theta,ci(1,1),ci(2,2));
```

There is also an example for rendering individual subject conf regions

Examples

```
N = 250;
X = mvnrnd([1 2], [1 .6; .6 1], N);
[ax,ci,cen,ub,lb,S,e,lam] = conf_region(X);
b = pinv([X(:,1) ones(size(X,1),1)] ) * X(:,2);
theta = atan(b(1)) - pi/2;
%create_figure('test');
[h,h2] = plot_ellipse(cen(1),cen(2),theta,ci(1,1),ci(2,2));
```

```
% plot standard deviation - for boostrapping, or for individual cases
[h,h2] = plot_ellipse(cen(1),cen(2),theta,ci(1,1)*sqrt(N),ci(2,2)*sqrt(N));
```

Example Render individual subject conf regions

```
X = x(wh, [3 1]);
dfe = size(X, 1) - size(X, 2);   % df
alph = .1;  % .1 for 90%, .05 for 95%
tcrit = tinv(1 - alph, dfe);
ci = ((lam) .^ .5) .* tcrit;
b = pinv([X(:,1) ones(size(X,1),1)] ) * X(:,2);
theta = atan(b(1)) - pi/2;
[h,h2] = plot_ellipse(cen(1),cen(2),theta,ci(1,1),ci(2,2));

set(h, 'Color', 'k', 'LineWidth', 1);
set(h2, 'FaceColor', [.5 1 0]);
```

Visualization_functions.**create_figure**(*tagname*, *varargin*)

Usage

```
f1 = create_figure(['tagname'], [subplotrows], [subplotcols], [do not clear flag])
```

checks for old figure with tag of tagname, clears it if it exists, or creates new one if it doesn't

Visualization_functions.**errorbar_horizontal**(*varargin*)

Error bar plot

Usage

```
errorbar_horizontal(X,Y,L,U)
```

plots the graph of vector X vs. vector Y with error bars specified by the vectors L and U. L and U contain the lower and upper error ranges for each point in Y. Each error bar is L(i) + U(i) long and is drawn a distance of U(i) above and L(i) below the points in (X,Y). The vectors X,Y,L and U must all be the same length. If X,Y,L and U are matrices then each column produces a separate line.

```
errorbar_horizontal(X,Y,E)
% or
errorbar_horizontal(Y,E)
```

plots Y with error bars [Y-E Y+E].

```
ERRORBAR(...,'LineSpec')
```

uses the color and linestyle specified by the string 'LineSpec'. The color is applied to the data line and error bars while the linestyle and marker are applied to the data line only. See PLOT for possibilities.

```
errorbar_horizontal(AX,...)
```

plots into AX instead of GCA.

```
H = errorbar_horizontal(...)
```

returns a vector of errorbarseries handles in H.

Examples To draws symmetric error bars of unit standard deviation

```
x = 1:10;
y = sin(x);
```

```
    e = std(y)*ones(size(x));
    errorbar(x,y,e)
```

If means(:, 1) is x-values, means(:, 2) is y-values, stds(:, 1) is error on x, and stds(:, 2) is error on y, then:

```
    lineh = errorbar_horizontal(means(:, 1), means(:, 2), stds(:, 1));
    linehx = errorbar(means(:, 1), means(:, 2), stds(:, 2));
```

Visualization_functions.**errorbar_width**(*h*, *x*, *interval*)

Work with errorbar.m: Adjust the width of errorbar

Usage

```
    errorbar_width(h, x, interval)
```

Inputs

h: errorbar graphic handle

x: vector x, which is used in errorbar

interval: e.g., [-.1 .1] or [0 0]

Examples you can see this output in

http://wagerlab.colorado.edu/wiki/doku.php/help/core/figure_gallery

```
    x = 1:5; % x values
    y = [32 40 55 84 130]; % mean
    e = [6 6 6 6 6]; % standard error of the mean

    create_figure(y_axis);
    set(gcf, 'Position', [1   512   268   194]);
    col = [0.3333    0.6588    1.0000];
    markercol = col-.2;

    h = errorbar(x, y, e, 'o', 'color', 'k', 'linewidth', 1.5, 'markersize', 7, 'markerfacecolor', c
    hold on;
    sepplot(x, y, .75, 'color', col, 'linewidth', 2);
    errorbar_width(h, x, [0 0]); % here

    set(gca, 'xlim', [.5 5.5], 'linewidth', 1.5);

    try
       pagesetup(gcf);
       saveas(gcf, 'example.pdf');
    catch
       pagesetup(gcf);
       saveas(gcf, 'example.pdf');
    end
```

Visualization_functions.**fill_area_around_points**(*x*, *y*, *borderscale*, *color*)

Usage

```
    h = fill_area_around_points(x, y, borderscale, color)
```

fills area around list of coordinates in a color using spline interpolation and other stuff. designed for cluster imaging in nmdsfig figures.

Example

```
x = randn(3,1);
y = randn(3,1);
figure; plot(x,y,'k.');
h = fill_area_around_points(x, y, .2, 'r');
```

Visualization_functions.**getVertexColors**(*xyz*, *v*, *actcolor*, *varargin*)

Usage

```
[c, alld] = getVertexColors(xyz, v, actcolor, [basecolor], [mind], 'vert', [xyz2], [actcolor2],
```

given a point list of XYZ mm coordinates (3 columns) and a list of vertices in an isosurface, returns FaceVertexCData color values for brain near points and brain not near points. c is vertex color specification, 3 columns indicating RGB values

Inputs

xyz: a 3-vol list of vertices to color

v: can be a matrix of vertices or a handle to a patch object containing vertices if it's a handle, this function sets the color to interp and the FaceVertexCData to the color matrix c

actcolor: [r g b] activation color

basecolor: [r g b] baseline color - optional.

mind: optional - min distance to color vertex Vertices within mind of an xyz coordinate will be colored

****colorscale** optional. followed by vector of values by which to multiply input color these are scaled to be between .3 and one.

if entered, this will make the colors vary by, for example, Z score so Z-scores are an acceptable input.

cscale should be in the same coordinate order as xyz

for ADDITIONAL clusters, repeat the 'colorscale', Z argument pair in the function call

YOU CAN ALSO pass true RGB values for each xyz coordinate in: 'colorscale', rgblist, IF cscale is a 3-vector, it specifies the ACTUAL colors, and is not scaled to .3 - 1

Following basecolor and mind:

additional xyz coordinate lists, with syntax: vert', xyz2 [your xyz input], [r g b] color for xyz plot

also, you can enter 'ovlcolor' followed by [r g b] for overlaps between xyz sets colors will ONLY appear in the overlap color if they share actual coordinates in common, not necessarily if surface vertices are within the specified distance from both sets of coords.

Examples

```
% to get a good brain surface, try this:
figure
p = patch('Faces', faces, 'Vertices', vertices, 'FaceColor', [.5 .5 .5], ...
        å  'EdgeColor', 'none', 'SpecularStrength', .2, 'FaceAlpha', 1, 'SpecularExponent', 200)
lighting gouraud;
camlight right
axis image;
myLight = camlight(0, 0);
set(myLight, 'Tag', 'myLight');
set(gcf, 'WindowButtonUpFcn', 'lightFollowView');
```

```
    lightfollowview
    drawnow
```

by Tor Wager August 25, 2002 ..

Visualization_functions.**get_cluster_volume**(*clIn*)

Usage

```
    clOut = get_cluster_volume(clIn)
```

Adapted from:

FORMAT clusters = ihb_getClusters

Get cluster information (use [SPM,VOL,xX,xCon,xSDM] = spm_getSPM;)

Visualization_functions.**glassbrain_avi**(*fps*, *len*, *clusters*)

Usage

```
    M = glassbrain_avi(fps,len,clusters)
```

works with cluster_surf.m assumes a gray brain where no activation, or creates one

SEE ALSO cluster_surf_movie.m, which works well too.

Visualization_functions.**imageCluster**(*varargin*)

Usage

```
    [out,cl] = imageCluster(arguments as specified below)
```

Images a cluster isosurface on an existing 3D head or brain plot

works with tsu (Talairach Space Utility) main window as current figure old: clusters = getappdata(gcf, 'clusters');

Inputs (in any order): keyword followed by input argument

'cluster': followed by cluster to image, from SPM or TSU.

'getclusters': no other args necessary - starts gui for cluster selection function returns all clusters. select with clusters(i)

'getfigclusters': get clusters from TSU main figure. must be current figure.

'figure': create a new figure to image color on

'color': followed by color value - either text or vector

'alpha': followed by transparency value for cluster surface, 0-1 1 is opaque, 0 is completely transparent

Control of smoothing

'heightthresh':

followed by cutoff threshold post-smooth, in percentage of min Z value in cl

- enter a number between 0 and 1

'fwhm': followed by smoothing kernel FWHM (Gaussian)

'kernelsize': followed by box size for kernel support (5 5 5 is default)

Outputs

out: Patch handle

cl: cluster struct

Uses cl.XYZmm and cl.voxSize

Works in Matlab 5.3, but with no transparency.

Example

```
p(i) = imageCluster('cluster',region2struct(r(i)),'color',colors{i},'alpha',1, 'fwhm', 1.2, 'hei
view(135, 30); lighting gouraud; lightRestoreSingle; axis image; camlight right;
```

Visualization_functions.**imageCluster_block**(*varargin*)

Usage

```
[out,cl] = imageCluster_block(input arguments)
```

Images a cluster isosurface on an existing 3D head or brain plot

works with tsu (Talairach Space Utility) main window as current figure old: clusters = getappdata(gcf, 'clusters');

Inputs (in any order): keyword followed by input argument

'cluster': followed by cluster to image, from SPM or TSU.

'getclusters': no other args necessary - starts gui for cluster selection function returns all clusters. select with clusters(i)

'getfigclusters': get clusters from TSU main figure. must be current figure.

'figure': create a new figure to image color on

'color': followed by color value - either text or vector

'alpha': followed by transparency value for cluster surface, 0-1 1 is opaque, 0 is completely transparent

Outputs

out: Patch handle

cl: cluster struct

Visualization_functions.**image_histogram**(*P*, *varargin*)

Usage

```
h = image_histogram(P,[method(string)],[range])
```

eliminates 0, NaN voxels from either image red then blue

Inputs

P: is one or two images in string array

Methods: 'def'

range: optional 3rd input, range of values to include in histogram

Example

```
h = image_histogram('p_Omnibus.img','def',[0 1-eps]);
```

Visualization_functions.**line_plot_multisubject**(*X*, *Y*, *varargin*)

Plots a scatterplot with multi-subject data, with one line per subject in a unique color.

Usage

```
[han, X, Y] = line_plot_multisubject(X, Y, varargin)
```

Inputs

X and Y: are cell arrays, one cell per upper level unit (subject)

varargin:

'n_bins: pass in the number of point "bins". Will divide each subj's trials into bins, get the avg X and Y per bin, and plot those points.

'noind': suppress points

'subjid': followed by integer vector of subject ID numbers. Use when passing in vectors (with subjects concatenated) rather than cell arrays in X and Y

'center': subtract means of each subject before plotting

'colors': followed by array size N of desired colors. if not passed in, will use scn_standard_colors

'MarkerTypes': followed by char string. if not passed in, uses 'osvd^<>ph' by default

'group_avg_ref_line': will make a reference line for the group avg

Outputs

han: handles to points and lines

X, Y: new variables (binned if bins requested)

Examples

Visualization_functions.**lineplot_columns**(*dat_matrix*, *varargin*)

Usage

```
out = lineplot_columns(dat_matrix, varargin)
```

Default values (see below for how to change) - w = 3; % width - color = 'k'; % color - wh = true(size(dat_matrix)); % which observations - x = 1:size(dat_matrix, 2); % x values - marker = 'o'; % marker style - linestyle = '-'; % line - markersize = 8; % markersize - markerfacecolor = [.5 .5 .5]; % face color - dowithinste = 0; % enter 'within' to get within-ss ste - atleast = 1; % 'atleast' followed by n for at least n valid obs to plot - doshading = 0; % shaded vs. error-bar plots - CIs95 = 0; % 95% CI's. Does not work with within subj error

Inputs

dat_matrix: is usually a rectangular matrix with rows = observations, columns = variables.

It can also be a cell array with column vectors in each cell, for unequal numbers of observations, but then the rows will not be the same observations across variables.

Optional Inputs followed by values:

{'w', 'color', 'x', 'marker', 'linestyle', 'markersize', 'markerfacecolor', 'wh'}

Keywords

{'within', 'dowithinste'}, 'shade', 'atleast', 'CIs' <- for 95% CI's

Examples

```
out = lineplot_columns(dat_matrix, 'w', 3, 'color', 'r', 'markerfacecolor', [1 .5 0], 'wh', ispa
out = lineplot_columns(dat_matrix, 'w', 3, 'color', [0 1 0], 'markerfacecolor', [1 .5 0], 'withi
out = lineplot_columns(dat_matrix, 'w', 3, 'color', 'b', 'markerfacecolor', [0 .5 1], 'within');

% shaded error regions
out = lineplot_columns(hotopen, 'color', 'r', 'marker', 'none', 'w', 1, 'shade');
```

Visualization_functions.**make3Davi**(*varargin*)

```
mov = make3Davi([opt] Options_Structure)
```

Makes an avi movie file called head3d[x].avi or whatever you specify.

Options

O.name: 'output_name.avi';

O.fps: frames per second

O.length: length of movie in s

O.H: cluster handles in vector (for adjusting transparency with time)

O.timecourse{i}: cell array of time courses for each cluster

O.timeres: resolution, in s, of timecourse data

O.azOffset: azimuth value to offset, positive = move clockwise

O.elOffset: elevation value to move through, positive = inf to superior

O.zoom: zoom value to end up with

O.add2movie: add to existing movie - enter mov structure in this field

O.closemovie: 1 or 0, close the movie afterward or not.

Blank fields for az, el, zoom, timecourse indicate that these functions should not be performed This script spirals up, right, and in 36 degrees

Notes: my indeo5 one wouldn't work in media player also only seems to work if you add clusters before head isosurfaces for Mac OS X / UNIX, choose "no compression"

start with the image in the location you want to zoom in on but with no zoom.

The movie FINISHES at the current axis locations

Example "Surface tour"

```
O = struct('name','all4_union_bg_surf.avi','fps',10,'length',6,'azOffset',180,'zoom',1,'elOffset
view(90,0);lightfollowview;O = struct('name','all4_union_bg_surf.avi','add2movie',mov,'fps',10,'
view(0,90);lightfollowview;O = struct('name','all4_union_bg_surf.avi','add2movie',mov,'fps',10,'
mov = close(mov);
```

Visualization_functions.**make3Davi_uncompressed**(*varargin*)

Usage

```
mov = make3davi([opt] Options_Structure)
```

Makes an avi movie file called head3d[x].avi or whatever you specify.

Options

O.name: 'output_name.avi';

O.fps: frames per second

O.length: length of movie in s

O.H: cluster handles in vector (for adjusting transparency with time)

O.timecourse{i}: cell array of time courses for each cluster

O.timeres: resolution, in s, of timecourse data

O.azOffset: azimuth value to offset, positive = move clockwise

O.elOffset: elevation value to move through, positive = inf to superior

O.zoom: zoom value to end up with

O.add2movie: add to existing movie - enter mov structure in this field

O.closemovie: 1 or 0, close the movie afterward or not.

Blank fields for az, el, zoom, timecourse indicate that these functions should not be performed This script spirals up, right, and in 36 degrees

Notes my indeo5 one wouldn't work in media player also only seems to work if you add clusters before head isosurfaces

Start with the image in the location you want to zoom in on but with no zoom.

Visualization_functions.**make_figure_into_orthviews**()

Usage

```
[hh1,hh2,hh3,hl,a1,a2,a3] = make_figure_into_orthviews
```

Copies a surface rendering or glass brain into three separate view panels, one saggital, one axial, and one coronal

returns handles to objects in each view and hl light handles

Examples

```
[hh1,hh2,hh3,hl,a1,a2,a3] = make_figure_into_orthviews;
axes(a1)
text(-55,60,70,'L','FontSize',24);text(55,60,70,'R','FontSize',24);
axes(a3)
text(-55,60,70,'L','FontSize',24);text(55,60,70,'R','FontSize',24);
```

Visualization_functions.**makelegend**(*names*, *colors*, *makefig*)

```
makelegend(names,colors,[decimal places if numeric entries for names])
```

Inputs

names: must be cell array of names OR a vector of numbers (i.e., thresholds) that will be converted to text

colors: can be cell array of text or rgb values, or matrix of [r g b] values

Examples

```
han = makelegend({'red' 'green' 'blue'}, {'r' 'g' 'b'});

han = makelegend([.001 .005 .01], {[1 1 0] [1 .5 0] [.7 .3 .3]});
```

Visualization_functions.**map_data_to_colormap**(*datavaluesets*, *poscm*, *negcm*, *varargin*)

Usage

```
actcolor = map_data_to_colormap(datavaluesets, poscm, negcm, varargin)
```

Given sets of data values (each cell is a row vector of data values, e.g., z-scores) and color maps for positive and negative values, returns mapped colors for each data value in order. These colors can be used for direct plotting.

Inputs

input 1: data values (e.g., z-scores). k data value sets, in cells. Each cell contains row vector of data values

input 2/3: color maps [n x 3] for positive and negative values

input 4: optional: fixed range of data defining max and min colors

Examples

```
poscm = colormap_tor([0 0 0], [1 1 0]);
negcm = colormap_tor([0 0 1], [0 0 0]);
Z = randn(40, 1)';
actcolors = map_data_to_colormap({Z}, poscm, negcm)
[Z' actcolors{1}]
```

Tor Wager, Sept. 2007 ..

Visualization_functions.**mask2surface**(*P*, *varargin*)

```
[hPatch,outP,FV, cl, myLight] = mask2surface(P)
```

Input

P: a file name for a mask file, OR a clusters structure

Inputs

hPatch: handle to surface patch

outP: file name, with path, of mat file containing faces and vertices

FV: isosurface

cl: clusters structure with coords and meshgrid info

varargin: suppress lighting (0) or do lighting (1)

varargin{2}: color for patch objects

uses get_cluster_volume, adapted from Sergey Pakhomov

use with cluster_surf.m to map activations onto surfaces: cluster_surf(clusters1,clusters2,outP,10,{[0 1 0] [1 0 0]})

or use getvertexcolors.m to map colors onto surface using hPatch.

Visualization_functions.**mdsfig**(*varargin*)

```
[f1,hh] = mdsfig(pc,names,classes,linemat,cordir)
```

Create the plot with stimulus coordinates

Inputs

pc: is objects x dimensions

clus: is a vector of object classes

names: is cell array of names for rows of pc (objects), or empty ([])

sigm: is optional matrix of 1, -1, and 0 entries signifies which pairs to connect with lines positive elements are solid lines, negative elements are dashed

Examples

```
mdsfig(pc,clus,names,sigmat);

x = randn(5,2);
y = [1 1 1 2 2]';
c = eye(5); c(1,2) = 1; c(1,4) = -1; c(2,3) = 1;
mdsfig(x,y,[],c)
```

Example2 using data output from mvroi.m

```
DAT.coords = CLUSTER.Gs(:,1:2); DAT.names = CLUSTER.names; DAT.classes =
CLUSTER.classes; DAT.lines = DATA.CORRELS.AVGSTATS.sigmat;
figure;
mdsfig(DAT.coords,DAT.names,DAT.classes,DAT.lines);
```

Visualization_functions.**mdsfig_3d**(*X*, *names*, *clus*, *linemat*, *cordir*)

```
mdsfig_3d(X,names,clus,linemat,cordir)
```

3-D MDS figure

Subfunction of mdsfig used if 3+ dims are available

Visualization_functions.**mea_visualise**(*plotmat*, *xaxis*, *yaxis*, *caxis*)
This program allows the visualisation of 3D images in separate subplots

Visualization_functions.**montage_clusters**(*ovl*, *clusters*, *varargin*)

Usage

```
fig_handle = montage_clusters(ovl, clusters, varargin)
```

Varargin (in any order) = a) additional clusters structures b) cell array of colors (text format), must be ROW vector {'r' 'g'} etc...

if length of color string is longer than number of clusters inputs, additional colors are interpreted as '2-intersection' and 'all-intersection' colors, in that order. This overrides single string argument (c, below) for color input

3. **single string color argument for overlaps (intersections)** plots intersections of ANY TWO clusters right now. also color for plotting points, if entered. use 'nooverlap' as an input argument to suppress this.
4. [n x 3] matrix of points to plot on map
5. text labels for points, must be cell array in COLUMN vector
6. **single number, 1/0 for whether to plot overlapping coordinates in overlap colors** default is 1.

g) color limit vector [min max] indicates color mapping for blobs rather than

solid colors. This will do hot/cool mapping

Intersections of 2 colors are magenta, and ALL colors are yellow unless otherwise specified try:

```
CLU = clusters2clu(clusters);
spm_orthviews('AddColouredBlobs', 1, CLU.XYZ, CLU.Z, CLU.M, [1 0 0])
```

Visualization_functions.**montage_clusters_maxslice**(*ovl*, *clusters*, *varargin*)

Varargin (in any order) = a) additional clusters structures b) cell array of colors (text format), must be ROW vector {'r' 'g'} etc...

if length of color string is longer than number of clusters inputs, additional colors are interpreted as '2-intersection' and 'all-intersection' colors, in that order. This overrides single string argument (c, below) for color input

3. **single string color argument for overlaps (intersections)** plots intersections of ANY TWO clusters right now. also color for plotting points, if entered. use 'nooverlap' as an input argument to suppress this.
4. [n x 3] matrix of points to plot on map
5. text labels for points, must be cell array in COLUMN vector
6. **single number, 1/0 for whether to plot overlapping coordinates in overlap colors** default is 1.
7. **color limit vector [min max] indicates color mapping for blobs rather than** solid colors. This will do hot/cool mapping

Intersections of 2 colors are magenta, and ALL colors are yellow unless otherwise specified

This maxslice version does 1) Not create a new figure, and 2) finds max slice through clusters to create single slice image.

Useful for displaying next to timecourse plots (for example) try:

```
CLU = clusters2clu(clusters);
spm_orthviews('AddColouredBlobs',1,CLU.XYZ,CLU.Z,CLU.M,[1 0 0])
```

Visualization_functions.**montage_clusters_medial**(*ovl*, *clusters*, *varargin*)

Varargin (in any order) = a) additional clusters structures b) cell array of colors (text format), must be ROW vector {'r' 'g'} etc...

if length of color string is longer than number of clusters inputs, additional colors are interpreted as '2-intersection' and 'all-intersection' colors, in that order. This overrides single string argument (c, below) for color input

3. **single string color argument for overlaps (intersections)** plots intersections of ANY TWO clusters right now. also color for plotting points, if entered. use 'nooverlap' as an input argument to suppress this.
4. [n x 3] matrix of points to plot on map
5. text labels for points, must be cell array in COLUMN vector
6. **single number, 1/0 for whether to plot overlapping coordinates in overlap colors** default is 1.
7. **color limit vector [min max] indicates color mapping for blobs rather than** solid colors. This will do hot/cool mapping

Intersections of 2 colors are magenta, and ALL colors are yellow unless otherwise specified

try:

```
CLU = clusters2clu(clusters);
spm_orthviews('AddColouredBlobs',1,CLU.XYZ,CLU.Z,CLU.M,[1 0 0])
```

Visualization_functions.**montage_clusters_points**(*ovl*, *clusters*, *XYZpts*, *varargin*)

```
ph = montage_clusters_points(ovl,clusters,XYZpts,varargin)
```

Inputs

varargin: additional clusters structures

XYZpts: XYZ mm coordinates of points to plot

ph: point handles

Visualization_functions.**montage_clusters_text**(*ovl*, *clusters*, *varargin*)

Inputs

varargin: additional clusters structures this function puts text cluster numbers on cluster centers

color: cell array of text strings indicating colors {'r' 'g'} etc...

Visualization_functions.**montage_clusters_text2**(*cl*)

Visualization_functions.**montage_image_Worsley**(*image_name*, *varargin*)

```
data = montage_image_Worsley(3D or 4D image name)
```

Make a compact montage of some images Designed by Keith Worsley for pca_image.m Adapted by Tor Wager, Feb 2008

Limits and Colormap are designed for component loadings between [-1 1]

Examples

```
create_figure('Montage');
montage_image_Worsley('test_run1_pca.img');

data = montage_image_Worsley(imgname, 'pcacov')  % changes scaling and colormap
data = montage_image_Worsley(imgname, 'pcacov', [1 3 5])  % show only
                                                    volumes 1, 3, 5 in image(s)
```

Visualization_functions.**movie_of_slice_timeseries**(*imgs*, *slicenumber*, *moviename*, *orientation*)

Inputs

imgs: Image names

slicenumber: which slice to view (in volume)

moviename: something like: 'slice10_timeseries.avi'

orientation: 'axial' or 'sagittal'

Examples

```
slicenumber = 10;
moviename = 'slice10_timeseries.avi';
```

Visualization_functions.**movie_stillframes**(*numframes*, *mov*)
Add still frames to a movie

Visualization_functions.**multi_threshold**(*P*, *type*, *df*, *varargin*)

```
cl = multi_threshold(P,type,df,[overlay image name])
```

F contrast: df = xSPM.df;

type = 'F' or 'T'or 'none'

Visualization_functions.**mvroi_mdsfig_plot2**(*CLUSTER*, *SPEC*, *sigmatavg*, *sigmatdif*, *titlestr*)

```
mvroi_mdsfig_plot2(CLUSTER,SPEC,DATA.CORRELS.AVGSTATS.sigmat_uncorrected,DATA.CORRELS.DIFSTATS.s
```

Example

```
mvroi_mdsfig_plot2(DATA.CLUSTER,DATA.SPEC,DATA.CORRELS.AVGSTATS.sigmat_uncorrected,DATA.CORRELS.
```

Visualization_functions.**mvroi_mdsfig_plot_sepstates**(*CLUSTER*, *CORRELS*, *SPEC*)

```
mvroi_mdsfig_plot_sepstates(CLUSTER,CORRELS,SPEC,DATA.CORRELS.AVGSTATS.sigmat_uncorrected,DATA.C
```

Plots SEPARATE STATES

Example

```
mvroi_mdsfig_plot2(DATA.CLUSTER,DATA.SPEC,DATA.CORRELS.AVGSTATS.sigmat_uncorrected,DATA.CORRELS.
```

Visualization_functions.**mvroi_mdsfig_plugin2**(*CLUSTER*, *SPEC*, *sigmatavg*, *sigmatdif*)
mvroi_mdsfig_plugint2(CLUSTER,SPEC,DATA.CORRELS.AVGSTATS.sigmat_uncorrected,DATA.CORRELS.DIFSTATS.sign

Visualization_functions.**mvroi_plot_firs**(*DATA*, *r*)

```
[h,t,w] = mvroi_plot_firs(DATA,region)
```

Examples

```
DATA.SPEC.firnames = {'Antic (C)'  'Pain (C)' 'Response (C)' 'Antic (P)' 'Pain (P)' 'Response (P
DATA.SPEC.firconditions = [1 4]
```

Visualization_functions.**nmdsfig**(*pc*, *varargin*)

```
f1 = nmdsfig(pc,[opt. inputs in any order])
```

Create a 1-D or 2-D plot with stimulus coordinates

reserved keywords, each followed by appropriate input:

- case 'classes', clus = varargin{i+1};
- case 'names', names = varargin{i+1};

- case 'sig', sigmat = varargin{i+1};
- case 'thr', thr = varargin{i+1};
- case 'legend', legmat = varargin{i+1};
- case 'sig2', sigmat2 = varargin{i+1};a sigmat2 can be thresholded at multiple values in thr
- case 'colors', colors = varargin{i+1};
- case 'sizes', sizes = varargin{i+1};
- case 'sigonly' plot regions with significant connections only
- case 'nolines', do not plot lines (lines plotted by default, but only if sigmat is entered)

'linethickness', followed by matrix of line thickness values

NOTE: can enter sig matrix with non-zero values equal to line thickness and use for both sig and linethickness inputs % but thickness values should be scaled to integers for line thickness

Creates a figure only if f1 output is requested

'fill', fill in areas around groups

pc: is objects x dimensions

clus: is a vector of object classes

names: is cell array of names for rows of pc (objects), or empty ([])

sig: is optional matrix of 1, -1, and 0 entries signifies which pairs to connect with lines positive elements are solid lines, negative elements are dashed

- Can be a series of t-maps in 3-D array

[opt] threshold vector of critical t-values, e.g., [2.2 5.4] If used, enter t-maps in sigmat

[opt] a 2nd sigmat, if entered, will plot dashed lines instead of solid ones. This is used by cluster_nmdsfig to plot interactions between covariance and behavioral scores

Figure creation

If existing fig with tag 'nmdsfig', activates Otherwise, if fig handle requested as output, creates or if not, uses current figure.

Examples c is output of cluster_nmdsfig

```
sizes = sum(c.STATS.sigmat);
f1 = nmdsfig(c.GroupSpace,'classes',c.ClusterSolution.classes,'names',c.names,'sig',p_vs_c_heat.

% add length legend
f1 =
nmdsfig(c.GroupSpace,'classes',c.ClusterSolution.classes,'names',c.names,'sig',p_vs_c_heat.sig,'
'sizes',sizes,'sizescale',[4 12],'lengthlegend',c.r);

% Auto size scaling based on number of connections:
f1 =
nmdsfig(c.GroupSpace,'classes',c.ClusterSolution.classes,'names',c.names,'sig',c.STATS.sigmat,'l

f1 =
nmdsfig(c.GroupSpace,'classes',c.ClusterSolution.classes,'names',c.names,'sig',p_vs_c_heat.sig,'
```

SEE ALSO cluster_nmdsfig

Visualization_functions.**nmdsfig1D**(*pc*, *clus*, *names*, *varargin*)

```
f1 = nmdsfig(pc,clus,names)
```

Create the plot with stimulus coordinates

Inputs

pc: is objects x dimensions

clus: is a vector of object classes

names: is cell array of names for rows of pc (objects), or empty ([])

sigm: is optional matrix of 1, -1, and 0 entries signifies which pairs to connect with lines positive elements are solid lines, negative elements are dashed

Examples

```
nmdsfig(pc,clus,names,sigmat);

x = randn(5,2);
y = [1 1 1 2 2]';
c = eye(5); c(1,2) = 1; c(1,4) = -1; c(2,3) = 1;
nmdsfig(x,y,[],c)
```

Visualization_functions.**nmdsfig_fill**(*varargin*)

Purpose: to take information about objects in multidimensional space and draw colored contours around them. Used within nmdsfig.m

Usage:

1) If c is a structure from cluster_nmdsfig, which is compatible with nmdsfig_tools, then:

```
hh = nmdsfig_fill(c)
```

Fields used are classes, groupSpace, and colors (see code for details)

2) Pass in arguments directly:

```
hh = nmdsfig_fill(classes, positions, colors)
```

classes is n x 1 vector of group assignments (or all ones for one group)

positions is n x 2 matrix of x, y coordinates colors is a cell containing as many colors as classes, {'r' 'g' 'b' ...} or {[1 0 0] [0 1 0] [0 0 1] ...}

Examples

```
load nmdsfig_output
hh = nmdsfig_fill(c)
set(findobj('Type','Line'), 'Color', 'k')
```

Visualization_functions.**nmdsfig_legend**(*X*, *r*)

X is stim coords, r is correlation coefficient matrix nmds figure should be current fig.

nmdsfig_legend(c.ClusterSolution.X,c.r) get x limit and dims of current nmds fig.

Visualization_functions.**plot3d**(*X*, *names*, *clus*, *linemat*, *cordir*)

define scaling %%%%%%%

Visualization_functions.**plotDesign**(*ons*, *rt*, *TR*, *varargin*)

Usage

```
[X,d,out,handles] = plotDesign(ons,rt,TR,varargin)
```

simple function to plot a design plots regressors and color-coded onset times as little sticks, with RT represented as height of the stick

Inputs

ons: a cell array of onset times in s

OR a delta indicator function matrix Event durations (durs) will ONLY work with cell array inputs

- optional *

The second column of each cell of ons can be a series of event durations for each event.

rt: is a cell array of rts or other parametric modulator for each onset event (or empty if no values)

TR: repetition time for sampling, in s

Optional Inputs

returns the model matrix (X) and the delta function d

optional arguments

1. y offset for plotting rts, default = 2
2. vector of epoch durations in sec for each trial type, default is events

'yoffset': followed by yoffset for plotting rts; default is auto scale

'durs': followed by durations in sec, either:

Constant duration

Vector of one duration for each event type Cell array of one duration per trial * Note: You can also add duration to ons input instead; see ons above for more info *

{'color', 'colors'}: followed by cell array of colors

'samefig': keep on same figure

'basisset': followed by name of basis set

'overlapping': Default is to plot separate lines in separate vertical positions. To plot overlapping in same location, enter this.

Examples plot epochs of different lengths stored in conditions(*).stimlength

```
[X3,d] = plotDesign(evtonsets,[],1,2,cat(2,conditions.stimlength));
```

See Also onsets2fmridesign

Visualization_functions.**plot_correlation**(*X*, *Y*, *varargin*)

Usage

```
handles = plot_correlation(X,Y,varargin)
```

plots robust or OLS simple or partial correlations replaces prplot and plot_correlation_samefig

Inputs

X: is matrix of columns of interest plus nuisance default is to plot partial effect of 1st column

Y: is one or more columns of data

Optional Inputs

'robust': robust IRLS plot

'noprint': suppress text output

'doquad': quadratic term; not tested, may not work

'col': followed by column of interest

'labels': followed by cell array of text labels for each obs.

'colors': followed by cell array of colors for each column of Y

'ylabel': followed by y-axis label string

'xlabel': followed by x-axis label string

'weights': followed by weights that override any computed ones

Examples Plot robust partial corr. 2 of X against col. 17 of Y,

controlling for other X

```
figure;
h = plot_correlation(X,Y(:,17),'col',2,'robust','ylabel','Brain
                     data','xlabel','Order effect');
```

Plot Col. 1 of X vs. Y in red squares

```
figure;
h = plot_correlation(X,Y(:,17),'robust','colors',{'rs'});

tor_fig;
```

Visualization_functions.**plot_correlation_samefig**(*xvec*, *yvec*, *varargin*)

Usage

```
[r,infostring,sig,h] = plot_correlation_samefig(xvec,yvec,[textlabs],[color],[doquad],[dorobust]
```

Inputs

xvec: x vector

yvec: y vector

varargin string of text labels

Optional Inputs

doquad: flag for quadratic correlations as well!

dorobust: remove n outliers from data, using Min Cov Determinant (MCD) Rousseeuw, P.J. (1984), "Least Median of Squares Regression," Journal of the American Statistical Association, Vol. 79, pp. 871-88 outliers calculated using IRLS (robustfit.m) do not work well. you enter n

empty variable arguments are OK, defaults will be used

Examples

```
    figure; [r, infos] = plot_correlation_samefig(x, y, [], 'ko', 0, 1);

    % for text labels only, try:
    plot_correlation(beh1,mri1,highlow,'w.');
```

Visualization_functions.**plot_dx_hrfs**(*EXPT*, *clusters*, *varargin*)

Usage

```
    EXPT = plot_dx_hrfs(EXPT,clusters,[dolegend],[dosave],[dosmooth],[doindiv])
```

Inputs uses EXPT.FIR and clusters

If not found, creates:

EXPT.FIR.regsofinterest = trial types EXPT.FIR.mcol = colors

Seems to be a problem with showing the brain slice when it makes the legend as well! Weird bug. Optional argument turns legend off.

Optional Inputs (all defaults are zero)

dolegend: 1 on, 0 off

dosave: 1 saves tiff files in timecourse_plots subdir, 0 does not

dosmooth: n smooths hrfs and re-calculates st. errors, 0 plots saved values stored in clusters.HRF.HRF and .STE

doindiff: plot low vs. high groups of individuals (indiv diffs)

See also

- extract_dxbeta_data.m, plot_dx_hrfs_indiffs.m

Examples

```
    % has some smoothing (0 weight @ 3 time pts)
    plot_dx_hrfs(EXPT,cl(1),0,1,3);
```

Visualization_functions.**plot_ellipse**(*x*, *y*, *theta*, *a*, *b*)

PLOT_ELLIPSE

Usage

```
    [h,h2]=plot_ellipse(x,y,theta,a,b)
```

This routine plots an ellipse with centre (x,y), axis lengths a,b with major axis at an angle of theta radians from the horizontal.

Visualization_functions.**plot_error**(*varargin*)

Plot a matrix with shaded error area around it

Usage

```
    [line_handle patch_handle] = plot_error(varargin)
```

Examples

```
    % plots matrix Y against x-values in X, where Y is a matrix with
    % each row representing a signal. The shaded area represents the
    % standard error across the columns of Y.
```

```
    PLOT_ERROR(X, Y)

    % plots the data matrix Y versus its index.
    PLOT_ERROR(Y)

    % uses external data for the error areas. In this case, Y is assumed
    % to be a mean timeseries already. Y and errorData must be vectors
    % of the same length.
    PLOT_ERROR(..., 'errorData', errorData)

    % The following example indicates whether to handle NaNs in the data.
    % If set, plot_error will use nanmean, nanstd, etc. Off by default.
    PLOT_ERROR(..., 'allowNaNs', [0|1])

    % The following example plots the line according to the designated
    % ColorSpec string, and shades the error area by the color of the line
    PLOT_ERROR(..., colorSpecString)

    % The following example plots into the axes designated by the AX
    % axes handle.
    PLOT_ERROR(AX, ...)

    % The following example returns the handle of the main line object
    [line_handle patch_handle] = PLOT_ERROR
```

Visualization_functions.**plot_horizontal_line**(*y*, *color*)

function han = plot_horizontal_line(y,color)

Visualization_functions.**plot_hrf_model_fit**(*m*, *TR*, *bf*, *varargin*)

Usage

```
    plot_hrf_model_fit(m,TR,bf,[stim input function for epoch, in 1 s resolution])
```

if m is an hrf curve sampled at 1 s,

plot_model_fit(m,1,'fir');

plot_model_fit(m,1,'hrf');

Visualization_functions.**plot_joint_hist_contour**(*z*, *xbins*, *ybins*, *color*, *varargin*)

plot a 95% 2-D density region for a 2-D histogram

Usage

```
    h = plot_joint_hist_contour(z, xbins, ybins, color, ['confval', confval], ['maxalpha', maxalpha]
```

Inputs

z: 2-D histogram values, counts in bins. See joint_hist.m

confval: optional input; [0 - 1], retain this proportion of values in confidence region

maxalpha: optional; [0 - 1], maximum transparency

Outputs

h: handle to graphical contour object

z: thresholded z matrix of counts

Examples

```
z = joint_hist(nnmfscores{i}{j}(:, 1),nnmfscores{i}{j}(:, 2), 50, 'noplot');
h = plot_joint_hist_contour(z, [0 0 1]);
```

Visualization_functions.**plot_matrix_cols**(*X*, *varargin*)

Usage

```
han = plot_matrix_cols(X, [method], [x-values], [colors cell], [linewidth], [axis limits])
```

Plot line plots showing each column of a matrix as a vertical or horizontal line

```
han = plot_matrix_cols(X)
```

Optional Inputs

method: plot_matrix_cols(X, 'horiz')

plot_matrix_cols(X, 'vertical')

x-values: plot_matrix_cols(X, 'horiz', x_in_secs)

colors: plot_matrix_cols(X, 'horiz', [], {'r' 'g' 'b' 'y' 'm'})

Linewidth: plot_matrix_cols(X, 'horiz', [], {'r' 'g' 'b' 'y' 'm'}, 2)

axis limits: plot_matrix_cols(X, 'horiz', [], [], [], [0 10])

legacy 'method' string: within denoising, plot_matrix_cols(X, 'denoising') to make red plots

Visualization_functions.**plot_vertical_line**(*x*, *color*)

Visualization_functions.**prplot**(*yy*, *X*, *k*, *varargin*)

Partial residual plot of y ~ X for column k Partial residual plot of one column of X against y. Uses IRLS estimation to downweight outliers if you enter a 4th argument.

Usage

```
[r,str,sig,ry,rx,h,rr] = prplot(y,X,col,[dorobust],[colors])
```

Inputs

y if y contains multiple columns, different colors and symbols will be used, with a separate regression for each.

colors e.g., {'ro' 'bs' 'gd' 'y^' 'cv' 'mx'}

Visualization_functions.**renderCluster_ui**(*varargin*)

Usage

```
O = renderCluster_ui([opt] O)
```

This ui is set to render on the default single_subj_T1, in default colors More flexibility is available if you use the functions.

main functions used:

- tor_3d.m - images head with cutaway views
- imageCluster.m - images a cluster isosurface
- mni_TSU.m and tor_ihb_TalSpace.m - to get clusters these and related functions are part of Talairach Space Utility written by Sergey Pakhomov, 2001 modified very slightly by Tor Wager to not convert to Talairach Space and use MNI coordinates instead.

Use of the TSU functions for getting clusters require SPM99.

Output to workspace Isosurface handles are in p, for head isosurfaces, and cH, for the cluster isosurface

D = image data, Ds = smoothed data, hdr = img header

Add more clusters by using:

```
cH(2) = imageCluster('cluster',clusters(i));
```

O = option structure with fields

dohead: y/n add head surface

head: filename or 'default' for default canonical brain

dobrain: y/n/filename add (transparent) brain surface, y for default brain or enter filename

get_from: workspace/file/TSU/TSUfigure/none get clusters from here if workspace, enter clusters in O.clusters

which_cl: vector of clusters to image (from list)

clcol: cluster colors - 3 el. vector, single letter, or string of letters enter letter or letter string in single quotes. e.g., O.clcol = 'yrgb';

whichc: letter string (no quotes) - which axes to cut along - xyzw are choices

addtext: y/n add text to clusters

textfield: field in cluster structure containing text

textcol: character code (r, b, g, etc.) for color of text

bestCoords: coordinates in mm to define x,y, and z cuts

revx: text string to reverse x cut direction, enter 1 to do it.

for solid brain rendering without the scalp, where dohead gives you a brain image, use brain_render_T1.img

see cluster_cutaways.m for an easy-to-use version.

for transparent brain rendering of a set of clusters, try:

```
figure('Color','w');
O = struct('dohead','n','dobrain','y','get_from','workspace', ...
            'clusters',clusters, ...
            'which_cl',1:length(clusters),'whichc','y','bestCoords',[0 0 0],'clcol','y','addtext
            'head','single_subj_T1');
renderCluster_ui(O)
```

Visualization_functions.**renderCluster_ui4**(*O*)

Usage

```
O = renderCluster_ui4(O)
```

Inputs O

O.head: name of head to use, no .img extension

O.surf: 'y' or 'n': image surface

O.sets: number of sets

O.getfrom: cell array of where to get clusters from: 'workspace' 'file' etc.

O.varname: cell array of variable names

O.color: cell array of colors - strings or vectors

O.numbers: 'y' or 'n': add numbers to plot

also need individual fields named contents of varname, which contain clusters

Visualization_functions.**roi_contour_map**(*dat*, *varargin*)

Draw a pattern map of one slice (either saggital, axial, or coronal) that shows the most voxels, or the slice that you specify (e.g., x = #). You can also draw outlines for the significant voxels from a statistical test.

Usage

```
info = roi_contour_map(dat, varargin)
```

Inputs

dat: dat can be fmri_data, statistic_image (to mark significant voxels), and region objects. If your dat is the "region" object, please add 'cluster' as an optional input. You can display two pattern maps for the purpose of comparison by putting additional columns of data in cell array.

An example of displaying two pattern maps:

```
dat{1} = region('img1.nii');
dat{2} = region('img2.nii'); )
```

Optional Inputs

'cluster': When the data is a region object, you need this option.

'sig': This option outlines significant voxels. To use this option, data in a format of statistic_image with a "sig" field should be given.

'colorbar': display colorbar under the plot.

'use_same_range': When you display two pattern maps, this option uses the same color range for the two maps.

'surf': surface plot rather than voxel-by-voxel mapping.

'xyz': When you want a specific view and slice, you can use this option with 'coord'. (1:x - saggital view, 2:y - coronal view, 3:z - axial view)

'coord': With 'xyz' option, this specifies the slice displayed.

'notfill': Default is to fill in the blank voxels using a black color. With this option, you can color the blank voxels with the white color.

'whole': Default is dividing the data into contiguous regions and show only one region that has the most voxels. This option akes this function not to divide into contiguous regions.

'colors' or 'color': you can specify your own colormap.

'contour': Not fully implemented yet.

Outputs

info: information about the display with the following fields.

info.dat:

- [2x30 double] (xyz mesh)
- Z: [1x30 double] (z values)
- xyz: 3 (1:x-saggital, 2:y-coronal, 3:z-axial)
- xyz_coord: -2 (slice coordinate; in this case, z = 3)
- region_idx: 1

Examples you can also see the same example and output in

http://wagerlab.colorado.edu/wiki/doku.php/help/core/figure_gallery

```
mask{1} = 'dACC_hw_pattern_sl6mm.nii';
mask{2} = 'dACC_rf_pattern_sl6mm.nii';
for i = 1:2, cl{i} = region(mask{i}); end
info = roi_contour_map([cl{1} cl{2}], 'cluster', 'use_same_range', 'colorbar');
```

Visualization_functions.**scn_export_papersetup**(*minsize*)

Usage

```
scn_export_papersetup([opt: min size in pixels, default = 400])
```

set paper size for current figure so that print to png or tiff looks as it should (as it does on-screen)

Visualization_functions.**scn_standard_colors**(*varargin*)

Create a set of unique colors in a standardized order.

Usage

```
colors = scn_standard_colors(100)
```

Optional input: minimum number of colors to generate

Repeats after 36 colors unique colors for each blob

Visualization_functions.**selective_average_interactive_view_init**(*imgs*, *onsets*, *scans_per_sess*, *TR*, *hp_length*, *t*, *basepts*, *plotstes*)

Usage

```
selective_average_interactive_view_init(imgs, onsets, 20, 1:2, 1);
```

Initialize point-and-click data extraction and plotting of selective averages

Inputs

imgs: cell array of image files, one cell per subject

imgs is cell array of time series image names for each subject

onsets: cell array of onsets

Each cell contains its own cell array, with one cell per condition.

t: time points in average to estimate

basepts: indices of baseline points to subtract from individual averages

plotstes: plot standard errors: 1 or 0

Visualization_functions.**sepplot**(*x*, *y*, *prop*, *varargin*)

Draw a shorter line plots between points. To see a figure example, please visit http://wagerlab.colorado.edu/wiki/doku.php/help/core/figure_gallery.

Usage

```
h = sepplot(x, y, prop, varargin)
```

Inputs

x, y: The function plots vector Y against vector X

prop: The proportion of the lines: prop can be between 0 and 1

Optional Inputs Enter keyword followed by variable with values

'color': followed by color (e.g., 'color', [.5 .5 .5]) (default = black)

'linewidth': followed by a number for linewidth (e.g., 'linewidth', 2) (default = .5)

'linestyle': linestyle, e.g., followed by '-', '–', ':' (default = '-')

Output

h: graphic handles for lines

Examples you can see the output in

http://wagerlab.colorado.edu/wiki/doku.php/help/core/figure_gallery

```
x = 1:5; % x values
y = [32 40 55 84 130]; % mean
e = [6 6 6 6 6]; % standard error of the mean

create_figure(y_axis);
set(gcf, 'Position', [1   512   268   194]);
col = [0.3333    0.6588    1.0000];
markercol = col-.2;

h = errorbar(x, y, e, 'o', 'color', 'k', 'linewidth', 1.5, 'markersize', 7, 'markerfacecolor', c
hold on;
sepplot(x, y, .75, 'color', col, 'linewidth', 2);
errorbar_width(h, x, [0 0]);

set(gca, 'xlim', [.5 5.5], 'linewidth', 1.5);

try
    pagesetup(gcf);
    saveas(gcf, 'example.pdf');
catch
    pagesetup(gcf);
    saveas(gcf, 'example.pdf');
end
```

Visualization_functions.**shepardplot**(*D*, *ntest*, *varargin*)

Usage

```
[Y,obs,imp,stress] = shepardplot(D, [ntest], [k dims to save])
```

Outputs

Y: is stimulus locations (cols are dims, rows are stimuli) = eigenvectors of scalar product matrix from cmdscale

obs: is vector of observed distances

imp: is vector of implied distances

squareform(obs) or (imp) = full matrix of distances

Visualization_functions.**sphere_roi_tool**(*varargin*)

Usage

```
clusters = sphere_roi_tool('mask','mask.img','bilat',1)
cl = sphere_roi_tool('mask',EXPT.mask,'bilat',0,'overlay',EXPT.overlay);
```

A tool for building a number of spherical ROIs masked with some other mask (e.g., gray matter). High-level end-user function.

Inputs arguments are in string - value pairs, in any order

mask: mask image for spheres, default: which('scalped_avg152T1_graymatter.img');

bilat: make all rois bilateral, default is 1

radius of sphere is defined on a region-by-region basis

write: followed by name of image to write out

afterwards, try eliminating few-voxel regions:

```
cl(find(cat(1,cl.numVox)<10)) = [];
```

and naming the clusters:

```
clusters = cluster_names(clusters);
```

draw on existing regions:

```
cl = sphere_roi_tool('mask',EXPT.mask,'bilat',0,'overlay',EXPT.overlay,'add');
```

use existing clusters

```
cl = sphere_roi_tool(cl,'mask',mask,'bilat',0,'overlay',ovl,'add');
```

Visualization_functions.**spm_orthviews_change_colormap**(*lowcolor*, *hicolor*, *varargin*)

Usage

```
cm = spm_orthviews_change_colormap(lowcolor, hicolor, midcolor1, midcolor2, etc.)
```

Create a new split colormap of your choosing and apply it to the spm_orthviews figure.

Examples

```
spm_orthviews_change_colormap([.2 .2 .6], [1 1 0]);  % slate to yellow
spm_orthviews_change_colormap([.9 .5 .2], [1 1 0]);  % orange to yellow
spm_orthviews_change_colormap([.8 .1 .1], [1 1 0], [.9 .6 .1]);  %red to orange to yellow
spm_orthviews_change_colormap([.2 .2 .4], [1 1 0], [.9 .6 .1]);  %slate to orange to yellow
cm = spm_orthviews_change_colormap([0 0 1], [1 1 0], [0 .5 1], [0 .5 .5], ...
[0 1 .5], [0 1 0], [.5 1 0]);
cm = spm_orthviews_change_colormap([0 0 1], [1 1 0], [.5 0 1], [.5 .5 1], ...
[1 .5 1], [1 .5 .5], [1 .5 0]);
```

Visualization_functions.**spm_orthviews_hotcool_colormap**(*t*, *thr*)

Usage

```
cm = spm_orthviews_hotcool_colormap(t, thr)
```

Create split-level colormap with hot colors for positive values and cool colors for negative ones

Apply this colormap to the spm Graphics window (or any figure with the tag 'Graphics')

Inputs

t: range of input statistic values (doesn't have to be t-values)

tthr:

threshold more extreme than which (+/-) colors should be used

enter a positive value

Designed to work on blobs added using spm_orthviews 'addblobs' feature

Examples for using 'clusters' in torlab format

```
% Threshold an image to get clusters:
[dat, volInfo, cl] = iimg_threshold('test_statistic.img', 'thr', 3.1440, 'k', 20);

% Display the clusters using spm_orthviews and apply the new color map:
cluster_orthviews(cl); cm = spm_orthviews_hotcool_colormap(cat(2,cl.Z), 3);
```

Visualization_functions.**spm_orthviews_name_axis**(*name*, *axisnum*)

Usage

```
spm_orthviews_name_axis(name, axis#)
```

put names on spm_orthviews axes

Visualization_functions.**spm_orthviews_showposition**()

Usage

```
handles = spm_orthviews_showposition;
```

plots x, y, and z coordinates on SPM orthviews figure

bring context structure variable into this script. contains handles, etc.

Visualization_functions.**spm_orthviews_white_background**()

Usage

```
spm_orthviews_white_background
```

Set up orthviews to draw with a white background and softened grayscale brain

No input arguments. Updates spm's global variable st

Uses SPM_OV_BLACK2WHITE.M

Visualization_functions.**spm_ov_black2white**(*varargin*)

Plugin for spm_orthviews to change the black background and dark edges to a white background with softer gray edges, for pub. quality figures

if st (a global variable) st.plugins has 'black2white' added, and the st.vols{i} struct has a field called 'black2white', then this will be called.

To initialize, add this code to the calling function:

```
    st.vols{1}.black2white = 1;
    bwexist = strfind(st.plugins, 'black2white')
    bwexist = any(cat(2, bwexist{:}))
    if ~bwexist
        st.plugins{end+1} = 'black2white';
    end
```

soften edges; 0 is no softening, a range is more softening

Visualization_functions.**standardMRIlighting**(*option*, *handles*)

Usage

```
    myLight = standardMRIlighting(option,handles)
```

Inputs

option: 'full' - all lighting adjustments

'reflectance' - ambient strength and reflectance only

handles: [isosurfaceHandle isocapsHandle]

Visualization_functions.**surf_plot_tor**(*data_matrix1*, *xvals*, *yvals*, *xname*, *yname*, *zname*, *varargin*)

Stylized surface plot of one or two surfaces

Usage

```
    [data_matrix12, xvals2, yvals2, data_matrix22] = surf_plot_tor(data_matrix1, xvals, yvals, xname
```

Examples See classify_search_script3...in meta-analysis classification

```
    surf_plot_tor(corrc_mean, mya, mys, 'Activation feature cutoff', 'Sensitivity feature cutoff', '
```

xvals are columns, yvals are rows!

Visualization_functions.**surface_cutaway**(*varargin*)

Make a specialized cutaway surface and add blobs if entered

Usage

```
    surface_handles = surface_cutaway(varargin)
```

Optional Inputs With no inputs, this function creates a vector of surface handles and returns them in surface_handles

'cl': followed by clusters structure or region object with blob info

'surface_handles': followed by vector of existing surface handles to plot blobs on

'ycut_mm':

followed by y-cutoff in mm for coronal section.

- if absent, shows entire medial surface

'existingfig':

Use existing figure axis; do not create new one

- if 'handles' are passed in, this is irrelevant

'pos_colormap':

followed by colormap for positive-going values

- n x 3 vector of rgb colors
- see colormap_tor or matlab's colormap

'neg_colormap': followed by colormap for negative-going values

'color_upperboundpercentile': followed by 1-100 percentile threshold; see Color values below

'color_lowerboundpercentile': followed by 1-100 percentile threshold; see Color values below

Output

surface_handles: vector of surface handles for all surface objects

Color values

Creates color scale using color-mapped colors, thresholded based on percentile of the distribution of scores in .Z field.

•To change the upper bound for which percentile of Z-scores is mapped to the highest color value, enter 'color_upperboundpercentile', [new percentile from 1-100]

•To change the lower bound value mapped to the lowest color value, enter 'color_lowerboundpercentile' followed by [new percentile from 1-100]

Examples

```
% Create new surfaces:
% r is a region object made from a thresholded image (see region.m)
surface_handles = surface_cutaway('cl', r, 'ycut_mm', -30);

% Image blobs on existing handles:
surface_handles = surface_cutaway('cl', r, 'handles', surface_handles);

% Do it step by step:
p = surface_cutaway();
p = surface_cutaway('cl', r, 'surface_handles', p);

% Use custom colormaps (you can define 'pos_colormap' and 'neg_colormap'):
poscm = colormap_tor([.5 0 .5], [1 0 0]);  % purple to red
p = surface_cutaway('cl', r, 'surface_handles', p, 'pos_colormap', poscm);
p = surface_cutaway('cl', r, 'ycut_mm', -30, 'pos_colormap', poscm);

% use mediation_brain_surface_figs and re-make colors
all_surf_handles = mediation_brain_surface_figs([]);
surface(t2, 'cutaway', 'surface_handles', all_surf_handles, 'color_upperboundpercentile', 95, 'c
```

Visualization_functions.**talairach_clusters**(*xyz*, *L*, *str*, *varargin*)

Usage

```
cl = talairach_clusters(xyz,L,str,[color, e.g., 'r'])
```

Saves and/or visualizes contiguous clusters given coordinates from the Carmack Talairach atlas and a textlabel (e.g., "Amygdala") for coordinates to retrieve

Examples

```
    load talairach_info L5 x y z; xyz = [x y z];
    cl = talairach_clusters(xyz,L5,'Amygdala','y'); save cl_amy cl

    % OR

    load
    cl = talairach_clusters(cl,[],[],'y');
```

get clusters structure given an XYZ mm list, a list of labels L in a cell array, and a string to match labels to

Complete example of making a limbic figure (some parts shown)

```
    load talairach_info L5 L3 x y z; xyz = [x y z];
    cl = talairach_clusters(xyz,L5,'Amygdala','y');
    cl = talairach_clusters(xyz,L3,'Caudate','b');
    cl = talairach_clusters(xyz,L5,'Putamen','g'); save cl_putamen cl
    cl = talairach_clusters(xyz,L5,'Lateral Globus Pallidus','c'); save cl_glo1 cl
    cl = talairach_clusters(xyz,L5,'Medial Globus Pallidus','c'); save cl_glo2 cl
    axis off
    set(gcf,'Color','w')
    h = lightangle(0,30);
    material dull
    h2 = lightangle(0,-30);
    h3 = lightangle(90,-30);
```

Visualization_functions.**timeseries_prplot**(*y*, *X*, *cols*, *varargin*)

Usage

```
    [r,bb,f] = timeseries_prplot(y,X,cols,varargin)
```

Plots timeseries data (y') against fitted response (X)

Inputs

y: y is n x 1 data points, X is n x k model matrix of predictors

y is adjusted to remove all effects OTHER THAN those columns of X specified in cols, creating a partial residual vector y'.

X: The fitted response to X(:,cols) is plotted against y' to graphically assess the effect of particular columns of X on y.

varargin includes two optional arguments:

1. a vector of trial onsets (to shade in plot)
2. length of elements to shade after trial onset.

Outputs

r: partial residuals

bb: betas

f: partial fitted response

Examples

```
    timeseries_prplot(Yfla,X,[2 4],x2,18);
```

to average across sessions 1 and 2:

Visualization_functions.**tor_3d**(*varargin*)

Usage

```
[D,Ds,hdr,p,coords,X,Y,Z] = tor_3d(varargin)
```

made to use single_subj_T1.img from SPM99

Options

'data': followed by image data, must also use 'hdr'; data is full image volume

'hdr': followed by hdr structure (use read_hdr)

'coords': 3 element row vector of x,y,z coordinates at which to cut away

'figure': create a new figure to plot on

'whichcuts': followed by incisions; choices are x y z w (whole)

example:'whichcuts','xyz' (xyz is default) order of output handles (p(i)) is 'wyzx'

New: Special methods:

- 'coronal slice right'
- 'coronal slice left'
- 'coronal slice'

'filename': followed by filename of image data to load, in single quotes

should be analyze img format, without the .img extension

cluster imaging assumes neurological orientation, but should work anyway.

'revx': reverse x cut direction, so cut in from left instead of right

'topmm': topmm = varargin{i+1};

'intensity_threshold': percentile of data above which is considered in-object higher = more sparse object

Outputs

D: img data

Ds: smoothed data

hdr: header

p: image handles

coords: coordinates

X, Y, Z: are the millimeter, origin centered reference frame for the head isosurface

Examples

```
[D,Ds,hdr,p,coords] = tor_3d('figure','data',D,'hdr',hdr,'whichcuts','yzx');
[D,Ds,hdr,p,coords] = tor_3d('figure');
[D,Ds,hdr,headhandle,coords] = tor_3d('whichcuts','z', 'coords', [-Inf
-Inf Inf], 'filename', 'T1_face_exemplar', 'intensity_threshold', 80); set(gcf, 'Color', 'w'); a

% Special slice example:
[D,Ds,hdr,handle,coords]  = tor_3d('whichcuts', 'coronal slice right', 'coords', [0 12 0], 'topm
lightRestoreSingle
set(handle(1), 'FaceColor', [.5 .5 .5])
```

Visualization_functions.**tor_fill_steplot**(*dat*, *color*, *varargin*)

Usage

```
[h,t] = tor_fill_steplot(dat,color,[robust flag],[p-thresh],[x vector],[covs no interest])
```

Plots a mean vector (mean of each column of dat) surrounded by a fill with standard err bars

If dat has 3 dimensions, then the diff between dat(:,:,1) and dat(:,:,2) is used as the difference for computing standard err (as in repeated measures)

if behavior is entered as optional argument, removes it before plotting lines. Also returns adjusted output in d, dat

Optional: robust flag (1/0), robust IRLS

Examples

```
tor_fig;
tor_fill_steplot(dat,{'b' 'r'},0,.05,secs);
```

Visualization_functions.**tor_ihb_GetClusterSet**()

Usage

```
bIsEmpty = ihb_GetClusterSet
```

Function to select set of cluster (SPM.mat) and select one cluster from this set

Return value == 1 if cluster set is nonempty and one cluster selected 0 otherwise

Visualization_functions.**tor_ihb_GetClusters**()

Usage

```
clusters = ihb_getClusters
```

Get cluster information (use [SPM,VOL,xX,xCon,xSDM] = spm_getSPM;)

Output

clusters: array of structs with fields:

Common to all clusters in the set

isSpmCluster:

1 if got from SPM data; 0 - if constructed by symmetrical or intersection

title: title for comparison (string) (SPM.title)

hThreshold: height threshold (SPM.u)

voxSize: voxel dimensions {mm} - column vector (VOL.VOX)

Specific for every cluster

name: name of the cluster

numVox: number of voxels in cluster

Z: minimum of n Statistics {filtered on u and k} (1 x num_vox)

XYZmm: location of voxels {mm} (3 x num_vox)

pVoxelLev: corrected p for max value in cluster

pClustLev: corrected p for given cluster (cluster level)

Talariach volume

xTal: x Talariach coordinates ready for contourslice & isosurface

yTal: y Talariach coordinates ready for contourslice & isosurface

zTal: z Talariach coordinates ready for contourslice & isosurface

vTal: Talariach volume values ready for contourslice & isosurface

xMin, yMin, zMin, xMax, yMax, zMax - bounding box in mm for Talariach

Visualization_functions.**tor_ihb_TalSpace**()

Usage

```
tor_ihb_TalSpace
```

Main function to view SPM99 cluster's (contours & 3D) in Talariach space

Graphic objects and their 'Tags':

type handle Tag Description

figure hFigMain ihb_TalSpaceMain_fig Main figure figure hFig3D ihb_TalSpace3D_fig 3D view figure figure hFigLeg ihb_TalSpaceLeg_fig Legend figure

axes hAxis3D ihb_axesTalSpace3D Axes for 3D view light hLight ihb_lightTalSpace3D Light object for 3D view

patch ihb_cntAxial contours patch handles patch ihb_cntFrontal in 3D view window patch ihb_cntSaggitalL patch ihb_cntSaggitalR

patch ihb_surfaceCluster isosurface patch

uicontrol ihb_CursorCoord coordinates under cursor uicontrol ihb_ClusterLevelProb cluster probability uicontrol ihb_VoxelLevelProb extr. probability uicontrol ihb_VolumeInVox cluster volume uicontrol ihb_ClusterPopUp select cluster combo box

uicontrol ihb_clr_Orig pushbuttons for uicontrol ihb_clr_Symm color change in uicontrol ihb_clr_Surface legend window uicontrol ihb_clr_Axial uicontrol ihb_clr_Frontal uicontrol ihb_clr_Saggital

line ihb_LineOriginal lines for original clusters line ihb_LineSymmetrical lines for symmetrical clusters

uimenu ihb_VoxSizeMenu111 uimenu ihb_VoxSizeMenu222 uimenu ihb_VoxSizeMenu444

uimenu ihb_RendererMenuZbuffer uimenu ihb_RendererMenuOpenGL

uimenu ihb_TransparencyMenu uimenu ihb_TransparencyMenuOpaque uimenu ihb_TransparencyMenuLow uimenu ihb_TransparencyMenuMedium uimenu ihb_TransparencyMenuHigh

uimenu ihb_Axes3DMenuClusterOnly uimenu ihb_Axes3DMenuWholeBrain

uimenu ihb_LineWidthMenuThin uimenu ihb_LineWidthMenuNormal uimenu ihb_LineWidthMenuThick

Application data (name used with setappdata the same as handle or variable name

for hFigMain:

axes hAxAxial axial axes hAxFront frontal axes hAxSagL left saggital axes hAxSagR right saggital strucure array clusters array of selected clusters integer indClusterToView index of currently viewed cluster in clusters array bool drawSymLS == 1 if draw symmetrical on left saggital bool drawSymRS == 1 if draw symmetrical on right saggital

for hAxis (where hAxis is one of hAxAxial, hAxFront, hAxSagL, hAxSagR): (see ihb_LoadSlice for detail)

strucure array sliceInfo array of Talarich slice information (ihbdfl_ax_info etc.) integer array rngArray array of local indexes in sliceInfo integer sliceIndex index of slice in the local range array rngArray double sliceDist slice distance from origin string axType axis type: 'ax' 'fr' 'sl' 'sr' 1x2 vector xLim limit values for X direction 1x2 vector yLim limit values for Y direction

Default information in ihb_TalSpaceDfl.mat file (see ihb_ResetDefaults)

To get default variable with name var use:

load('ihb_TalSpaceDfl.mat', 'var');

To save:

save('ihb_TalSpaceDfl.mat', 'var', '-append');

Figures tags for windows created by SPM during call spm_getSPM

ihbdfl_spm_fig_Interactive ihbdfl_spm_fig_SelFileWin ihbdfl_spm_fig_ConMan

Minimal size of clusters to draw

ihbdfl_min_size_to_draw

Sizes of voxels used in Talariach space

ihbdfl_tal_x_vox ihbdfl_tal_y_vox ihbdfl_tal_z_vox

Sizes of main window and axis for ihb_TalSpace

ihbdfl_main_width ihbdfl_main_height ihbdfl_main_gap ihbdfl_main_x ihbdfl_main_y ihbdfl_main_z ihbdfl_main_info_h

Talariach slices information

ihbdfl_ax_info ihbdfl_fr_info ihbdfl_sl_info ihbdfl_sr_info

Axis limits to draw slice

ihbdfl_xLimD ihbdfl_yLimD ihbdfl_zLimD ihbdfl_xLimI ihbdfl_yLimI ihbdfl_zLimI

3D view Renderer ('zbuffer' or 'OpenGL') 3D view transparency (available only for OpenGL

ihbdfl_renderer ihbdfl_transparency

Axes for 3D view type. If ihbdfl_bAxesIsClusterOnly == 1 - axes have cluster range 0 - axes have brain range

ihbdfl_bAxesIsClusterOnly

Colors defaults

ihbdfl_color_clOrig ihbdfl_color_clSymm ihbdfl_color_surface ihbdfl_color_cntAx ihbdfl_color_cntFr ihbdfl_color_cntSag

Line width

ihbdfl_line_width

Visualization_functions.**tor_ihb_UpdateClusterTalVoxSize**(*clIn*, *nTotalVox*, *nTotalProcessed*)

Usage

```
clOut = ihb_ChangeTalVoxSize(clIn)
```

Inputs

clIn: initial input cluster (structure see ihb_GetClusters)

nTotalVox: total nmb of voxel in all clusters (used for waitbar)

nTotalProcessed: total nmb of voxel in already processed clusters (used for waitbar)

Output

clOut: output cluster with new Talariach related fields

It is assumed that waitbar already exists

Load current voxel size ..

Visualization_functions.**tor_polar_plot**(*vals*, *colors*, *names*, *varargin*)

Make polar line plot(s)

Usage

```
hh = tor_polar_plot(vals, colors, names, ['nofigure'])
```

Inputs

vals: cell array, one cell per plot

in each cell, matrix of observations x variables plots one line for each variable.

names: is cell array, one cell per plot

contains cell array of names for each condition

Optional Inputs

'nofigure': suppress figure

'nonneg': make all values non-negative by subtracting min value from all values in series (plot)

'nofill': Do not fill in polygons

Output

hh: Handles to line objects

Examples

```
tor_polar_plot({w+1}, {'r' 'b'}, setnames(1))
```

Note: Dark grey inner line is zero point if nonneg option is used. Otherwise, zero is the origin.

Visualization_functions.**violinplot**(*Y*, *varargin*)

Simple violin plot using matlab default kernel density estimation

This function creates violin plots based on kernel density estimation using ksdensity with default settings. Please be careful when comparing pdfs estimated with different bandwidth!

Differently to other boxplot functions, you may specify the x-position. This is particularly usefule when overlaying with other data / plots.

Input

Y: Data to be plotted, being either n x m matrix. A 'violin' is plotted for each column m, OR 1 x m Cellarry with elements being numerical colums of nx1 length.

varargin:

xlabel: xlabel. Set either [] or in the form {'txt1','txt2','txt3',...}

facecolor=[1 0.5 0]: FaceColor: Specify abbrev. or m x 3 matrix (e.g. [1 0 0])

edgecolor='k': LineColor: Specify abbrev. (e.g. 'k' for black); set either []," or 'none' if the mean should not be plotted

linewidth=2: Linewidth for boundary of violin plot

facealpha=0.5: Alpha value (transparency)

mc='k': Color of the bars indicating the mean; set either []," or 'none' if the mean should not be plotted

medc='r': Color of the bars indicating the median; set either []," or 'none' if the mean should not be plotted

bw=[];:

Kernel bandwidth, prescribe if wanted. %If b is a single number, b will be applied to all estimates %If b is an array of 1xm or mx1, b(i) will be applied to column (i).

'x': followed by x position for center(s) of plots

'nopoints': don't display dots

'pointsize': you can define point size.

'pointcolor': if you want to use different color for points, you can use this option.

'weights': You can add weights in the same format of the data. It will use weighted version of kdensity function, show weighted mean, and weighted point sizes. Be careful: if some data points have NaNs, this will remove those data points from the plot.

Outputs

h: figure handle

L: Legend handle

MX: Means of groups

MED: Medians of groups

bw: bandwidth of kernel

pointloc: point locations on violin plot This has x, y, idx as subfields. idx indicates the order of the data in the original input.

Example1 (default)

```
disp('this example uses the statistical toolbox')
Y=[rand(1000,1),gamrnd(1,2,1000,1),normrnd(10,2,1000,1),gamrnd(10,0.1,1000,1)];
[h,L,MX,MED]=violinplot(Y);
ylabel('\Delta [yesno^{-2}]','FontSize',14)
```

Example2 (specify facecolor, edgecolor, xlabel)

```
disp('this example uses the statistical toolbox')
Y=[rand(1000,1),gamrnd(1,2,1000,1),normrnd(10,2,1000,1),gamrnd(10,0.1,1000,1)];
violinplot(Y,'xlabel',{'a','b','c','d'},'facecolor',[1 1 0;0 1 0;.3 .3 .3;0 0.3 0.1],'edgecolor'
           'bw',0.3,...
           'mc','k',...
           'medc','r--')
ylabel('\Delta [yesno^{-2}]','FontSize',14)
```

Example3 (specify x axis location)

```
disp('this example uses the statistical toolbox')
Y=[rand(1000,1),gamrnd(1,2,1000,1),normrnd(10,2,1000,1),gamrnd(10,0.1,1000,1)];
violinplot(Y,'x',[-1 .7 3.4 8.8],'facecolor',[1 1 0;0 1 0;.3 .3 .3;0 0.3 0.1],'edgecolor','none'
           'bw',0.3,'mc','k','medc','r-.')
axis([-2 10 -0.5 20])
ylabel('\Delta [yesno^{-2}]','FontSize',14)
```

Example4 (Give data as cells with different n)

disp('this example uses the statistical toolbox')

```
Y{:,1}=rand(10,1);
Y{:,2}=rand(1000,1);
violinplot(Y,'facecolor',[1 1 0;0 1 0;.3 .3 .3;0 0.3 0.1],'edgecolor','none','bw',0.1,'mc','k','
ylabel('\Delta [yesno^{-2}]','FontSize',14)
```

Visualization_functions.**wani_pie**(*X*, *varargin*)

Draw a little better pie chart

Usage

```
h = wani_pie(X, varargin)
```

Inputs

X: a vector

Optional Inputs Enter keyword followed by variable with values

'cols' or 'colors': colors N x 3 {default: using colors from microsoft office} OR cell array of 3-element colors (no text input for colors!)

'notext': no text for percentage {default: false}

'fontsize': font size for percentage {default: 15}

'hole': add a hole in the middle of the pie chart {default: no hole}

'hole_size': specify the size of the middle hole {default: 5000}

'outline'

'outlinecol'

'outlinewidth'

Output

h: graphic handles

Examples

```
% data
X = rand(10,1);
h = wani_pie(X, 'notext', 'hole')

savename = 'example_pie.pdf';

try
    pagesetup(gcf);
    saveas(gcf, savename);
catch
    pagesetup(gcf);
    saveas(gcf, savename);
end
```

Visualization_functions.**xval_lasso_brain_permutation_histogram**(*stats*)

Plot histograms and get permutation test-based p-value for xval_lasso_brain output structure

xval_lasso_brain_permutation_histogram(stats)

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

@

c

d

f

h

i

m

p

r

s

v

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z