Speech**L**ab **T**oolkit

version 1.0

19 September 2003

Disclaimer

Members of SpeechLab, Department of Cognitive and Neural Systems and Boston University does not warrant or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed.

Licensing

Introduction

This document provides information on the installation and use of the **S**peech**L**ab **T**oolkit (**SLT**), a Matlab based package for performing region-of-interest (ROI) analyses of functional magnetic resonance imaging (fMRI) datasets as described in Nieto-Castanon et al (2003). *The current procedures and their parameters are tuned for our needs but the flexibility exists to modify them for yours.* While the aim of **SLT** is to perform fMRI analyses, it has several general-purpose components that can be used for other purposes.

Features

- ✓ Matlab based
- ✓ Region-of-interest (ROI) based fMRI analyses
- ✓ Integrated with SPM2
- ✓ A simple scriptable batch mode interface to SPM2
- ✓ A user interface for parcellation of cortical regions (A Matlab based CardViews clone)
- ✓ On UNIX platforms, can be used with FreeSurfer for automatic parcellation
- ✓ Minimizes duplication of data for multiple analyses

System requirements

Technically, this toolbox should work on any platform on which Matlab is supported. However, we have been using it on Linux (Redhat 8.1) and Windows 2000/XP with Matlab 6.5. Therefore, we cannot predict its performance on other configurations.

Optional components:

- ✓ FreeSurfer [http://surfer.nmr.mgh.harvard.edu]
- ✓ Visualization Toolkit (VTK) Libraries [http://public.kitware.com]

Download and Future updates

The toolbox and future updates can be downloaded from:

http://cns.bu.edu/~speech/software.php

Installation

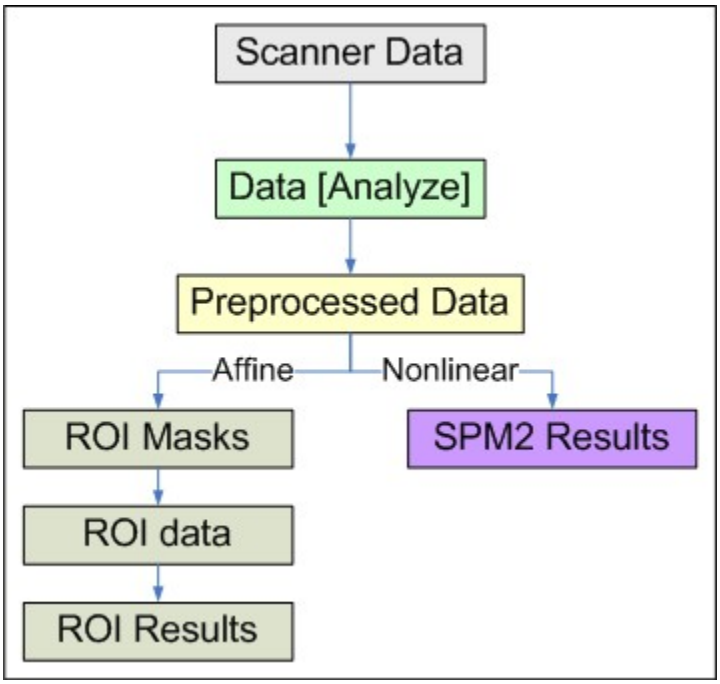
1. Unzip the software. Depending on the method of unzipping, the unzipped folder might contain another folder of the same name. In this case move the internal folder to the location you want it installed. Let us call the installation folder **SLTfolder**.
2. Add the path: **SLTfolder**/startup to the Matlab path.

Components

The **SLT** consists of several different components: some more general than others. The current composition is described in brief below.

1) **ROI Analysis Toolbox 2 (RAT2):** This is the main component of the package and includes:
   - ✓ An SPM2 batch processing system
   - ✓ ROI analyses scripts (spm_ROI)
   - ✓ Figure creation scripts for both SPM2 and spm_ROI results
   - ✓ Preprocessing scripts for ASAP
   - ✓ FreeSurfer interaction tools (UNIX only)
2) **A Segmentation and Parcellation Program (ASAP):** A graphical interface for creating ROI masks for brain images. This tool is a Matlab based clone of the program CardViews.
3) **SurfTools:** A set of computational geometry tools to work with manifold triangular meshes. In addition it provides the means for including VTK functionality in Matlab.
4) **FSTools:** A set of Matlab scripts provided by the authors of FreeSurfer for accessing and manipulating different FreeSurfer file formats.

Operational flowchart



Limitations

- ✓ Works only with Analyze format data
- ✓ Tested on block and event triggered experiments only
- ✓ fMRI analyses assumes existence of certain files:
    - o A T1 weighted slice scan
- ✓ The realignment process does not align to mean image. Assumes first image is aligned to T1 slice scan.

Analysis checklist

STEPS	SPM2	SPM_ROI	
		ASAP	FS
1. Store data in a structured manner			
2. Create experiment scripts[+]			
3. Create experiment object			
4. Preprocess data			
5. Preprocess [ASAP]	■		■
6. Create Mask [ASAP][+]	■		■
7. Create Mask [FreeSurfer][+]	■	■	
8. Extract ROI data	■		
9. Create ROI project	■		
10. Results			

[+] These particular steps require more manual interaction than the rest. While most of the other steps involve calling a function, these steps require creating and/or editing data.

Matlab command sequence

```
1. RAT2_start
2. cd(?experiment working directory?);
3. ABCD = experiment_setup_ABCD; save ABCD;
4. ABCD = roi_preprocess_subjects(ABCD); save ABCD;
   When a new subject is added use the following:
   4.1.  roi_preprocess_subjects(ABCD,length(ABCD.subject));
   4.2.  ABCD = roi_preprocess_subjects(ABCD,[],'both',[0 0
      0 0 0]);

5. ASAP_start
6. roi_RAT2ASAP(ABCD);
7. Perform ASAP edits
8. roi_RAT2ASAP(ABCD,[],[0 1]);
9. freesurfer_start

10. roi_RAT2FS(ABCD,'ABCD',[],[1 1 1 0 0 0 0 0]);
11. Perform FreeSurfer edits
12. roi_RAT2FS(ABCD,'ABCD',[],[0 0 0 1 1 1 1 1]);

13. Modify roi_setup_ABCD if needed.
14. ABCD = roi_setup_ABCD(ABCD); save ABCD;
15. ABCD = roi_create_ROI_data(ABCD); save ABCD;
   When a new subject is added use the following:
   15.1. roi_create_ROI_data(ABCD,length(ABCD.subject));
   15.2. ABCD = roi_create_ROIdata(ABCD,[],[0 0]);

16. mkdir('ABCD.spm2');cd('ABCD.spm2');
17. roi_fixedfx_subjects(ABCD);
```

```
roi_spmFigures(ABCD);
mkdir('ABCD.roi');cd('ABCD.roi');
spm_ROI_subjects([],'ABCD_filename');
spm_ROI_model;
results = spm_ROI_results;
mkdir('ABCD.figures');cd('ABCD.figures');
roi_Fig_create(results);
```

Checklist description

1. **Store data in structured manner**

Properly organizing the directory/folder structure of the data can help speed implementing the object creation scripts. Names of subject and series directories that are indexed and linearized (e.g. 'subject.001', 'structuralseries.003') are more convenient than a non-indexable name (eg. MKYYAAZX). As an example, the directory structure used in our lab is shown below.

[**** INSERT FIGURE HERE ***]

The directory structure is then specified in a template that is then used to access data locations in the experiment scripts described below.

2. **Create experiment scripts [15-60 minutes]**

This is the single most important step of this processing stream and involves the creation of 5 Matlab scripts and a matrix/cell array of matrices.

- experiment_setup_ABCD.m
- subject_setup_ABCD.m
- design_setup_ABCD.m
- contrast_setup_ABCD.m
- roi_setup_ABCD.m
- runinfo_ABCD.mat

On most occasions, one will find that this process involves modifying the demo scripts or a previously created script using some simple guidelines. These guidelines are dictated by 5 different Matlab object classes: @experiment, @subject, @design, @session and @roisession. These scripts can be created before scanning any subjects and modified appropriately as new subjects are added to the experiment.

2.1. Experiment_setup_ABCD.m: [See experiment_setup_demo.m]

This script is responsible for the "business" end of the functional analysis. The file creates and fills an experiment object. If written properly, this is the only script that will need to be modified as new subjects are added to the experiment. Based on the directory structure in our lab, modifiable details are described below. This script calls the remaining scripts (except roi_setup_ABCD.m) to fill in the various fields of the experiment structure.

- ✓ Define the fields of dirTemplate: This global variable informs subject_setup_ABCD.m where to acquire subject specific information from.
- ✓ Set basic field information (see script comments)
- ✓ Set number of subjects.
- ✓ Define for each subject the session numbers that contain functional data, e.g.: sessidx={[4:9],[5:10],[3:7,9]}; where each vector represents the valid runs for each of the 3 subjects in this study
- ✓ Create Boolean cell array representing which runs are valid. In subject_setup_ABCD, this information will be used to determing invalid

trials. For each invalid run, there will be lengthPerRun (or volumespersission in subject_setup) invalid trials. A concatenated array of all trials with size equal to total # of runs*lengthPerRun is created with indices corresponding to valid runs = 0 and indices corresponding to invalid runs = 0. This allows the invalid runs to be used for preprocessing and thrown out for the functional analysis.

- ✓ Set run order of the runs for each subject e.g.: runorder = {[4 1 2 5 3],[1 5 2 3 4],[5 1 2 4 3], [1 2 5 3 4]}; Assuming that the set of runs for every subject were the same and the only difference was the order of the runs, this variable contains that information. However, if the information about run order is embedded in the runinfo variable (see runinfo_ABCD.m) then the runorder can be left sequential or empty.

2.2. Subject_setup_ABCD.m [See subject_setup_demo.m]

This script adds run specific information per subject to the experiment object. The information added includes filenames for structural, hiresolution and functional images and information to build the design matrix. For most experiments this file can be left unmodified (choose the appropriate subject_setup relevant to your experiment). The information required by this file comes from 3 sources: experiment_setup_ABCD.m, design_setup_ABCD.m and the information contained in the runinfo variable saved in a Matlab data file pointed to by the field 'runinfo' in the design object. Depending on how runinfo is created (see roi_setup_ABCD.m below), this file may need to be modified.

2.3. Design_setup_ABCD.m [see design_setup_demo.m]

In this file we set more parameters specific to the functional runs. This involves setting appropriate fields of the @design object as defined below:

- ✓ dsgn.type ['block','trigger','event'] Setting this field defines the type of scanning paradigm used.
- ✓ dsgn.TR [in seconds] specifies the repetition time, or the time to acquire a single scan
- ✓ dsgn.volumespertrigger specifies the number of volumes acquired after triggering the scanner.
- ✓ dsgn.eventspersession specifies the total number of events that were presented to the subject during a single run.
- ✓ dsgn.condnames {'speaking','rest'} specifies the names of different parametric conditions
- ✓ dsgn.blocklength [30,15] specifies the length of each condition block. Can be a scalar if all blocks are of the same duration, or a vector otherwise.
- ✓ dsgn.detrend specifies whether or not to use a detrending covariate in the design matrix
- ✓ dsgn.runinfo specifies the name of the Matlab file containing the runinfo variable
- ✓ dsgn.runinfotype = 1; % 1 - per subject, 2 - per experiment

The following units are as defined by the spm_fMRI_design file. Since the processing functions provide a wrapper around spm, these values need to be set in the design_setup_ABCD.m file.

- ✓ dsgn.xBF_name specifies the type of basis set used
- ✓ dsgn.xBF_length specifies the length in durations of seconds the length of the basis function
- ✓ dsgn.xBF_order specifies the order of the basis set
- ✓ dsgn.xBF_T specifies the total number of functional slices collected
- ✓ dsgn.xBF_UNITS specifies the time units in terms of 'scans' or 'secs'. This is the same time units that are used for blocklength above.
- ✓ dsgn.xX_K_HParam: set to total length of a run
- ✓ dsgn.xVi_form: either 'none' or 'AR(1)_+_w'. The second option allows for serial correlations and would normally choose this for block design. May need to set to 'none' if run into memory problems. This can be overcome by performing random effects analysis. Also please use spaces instead of the underscores in ‘AR(1)_+_w’.

The following design parameters are specific to the ROI analyses stream.

- ✓ dsgn.roiSmoothFWHM specifies intra region smoothing extent
- ✓ dsgn.L2_X specifies the Level 2 design matrix. Can be left empty
- ✓ dsgn.whitening [1 or 0] specifies whether or not to whiten the data.
- ✓ dsgn.dataReductionType ['SVD' or ‘FFT’] specifies the type of data reduction to use
- ✓ dsgn.dataReductionLevel [3-15]; specifies the number of data reduction components kept
- ✓ dsgn.useSPHcoords [0 or 1] specifies whether to use spherical coordinates or not during ROI analyses. This depends on whether the mask data was generated using FreeSurfer.

2.4. Contrast_setup_ABCD.m [see contrast_setup_demo.m]

Defines the desired contrasts for the study, the contrast name, and the test to be performed ('T', of 'F”). The length of each contrast vectors must match the number of items in the dsgn.condnames (see design_setup_ABCD.m above). The contrasts are defined as a struct array that is assigned to the contrasts field of the experiment object. An example contrast is defined below:

```
contrast(1).c      = [1 1 1 -1 -1 -1 0 0 0 0];
contrast(1).name   = ['condA-condB'];
contrast(1).stat   = 'T';
```

2.5. Roi_setup_ABCD.m [see roi_setup_demo.m]

The script creates an roisession object which consists of information from the subject session and creates a new mask field which contains a pointer to the ROI mask file. Either the freesurfer-generated (default) or ASAP-generated masks can be used. For mask data generated FreeSurfer one should be able to use the default script provided in the package. For data generated with another source, the template script needs to be modified to point at the

location of the mask data.

2.6. Runinfo_ABCD.mat

The runinfo_* file contains a variable runinfo that defines the order of functional events/blocks within each of the functional runs. There a several ways of creating this variable but the manner demonstrated below should take care of all possibilities that the scripts can handle.

Per subject per run per condition with overlapping conditions

runinfo{subjectID}{runID}(:,conditionID)

This consists of a matrix per subject per run where each column is a condition and the number of rows equals the number of blocks, events or triggers (**BET**s). The values are Boolean in each column with a 1 in the row that corresponds to a **BET** execution. This structure handles all designs that we have utilized in our studies. If you find something that is represented by this structure please let us know.

3. **Create experiment object [a few minutes]**

Now that the pertinent files have been set up, the analysis can be started.

3.1. Upon launching MATLAB, run **RAT2_start** to set up all the paths for functional analysis (use **freesurfer_start** if you are going to use FreeSurfer at some stage).

3.2. Then add to the paths for the scripts that have been created above:

addpath('Path_to_your_scripts')

e.g.: addpath ~/software/scripts/satra/demo01/

3.3. Use the directory containing all your subject folders as your working directory. [This is not necessary if you used the directory prefix fields of dirTemplate.]

3.4. execute: ABCD_object = experiment_setup_ABCD;

3.5. save ABCD_project ABCD_object [This object can be saved at any location. Since the object contains hardwired paths to your data, the data cannot be moved from the current location. If the data is moved, the object will need to be recreated. If the data was already processed using **roi_preprocess**, then it does not need to be processed again]

These 5 steps need to be executed each time you add a new subject to the experiment.

4. **Preprocess data [1-2 hrs per subject][See roi_preprocess for details]**

In this step all the processing required to carry out standard SPM2 analyses is performed. This process includes realignment, coregistration, normalization (both affine and nonlinear) and smoothing (nonlinear normalization).

Execute: ABCD_object = **roi_preprocess_subjects**(ABCD_object);

Since this process is time consuming, once it has been completed, the paths to the various processed data can be regenerated without reprocessing. This implies that even if the ABCD_object is corrupted or deleted by mistake, the pointers can be regenerated without processing the whole data set again. The command for pointer generation is as given below.

```
ABCD_object = roi_preprocess_subjects(ABCD_object,[],'both',[0 0 0 0 0]);
```

Please remember to save the object each time you carry out this step.

Save: save ABCD_project ABCD_object

5. **Preprocess [ASAP] [15-20 minutes per subject]**

Preprocessing for ASAP consists of taking the affine transformed structural image file (typically w*.img in spm2) created during the preprocess step and running it through some segmentation/brain extraction routines. This can be performed on a single structural image using the **roi_asap_preprocess** command or on the ABCD_object using **roi_RAT2ASAP** function. The roi_RAT2ASAP function simply iterates through all the brains and processes them individually using roi_asap_preprocess. In addition it has the option of generating the mask file automatically from the ASAP project file, once all the regions have been created.

6. **Create Mask [ASAP] [0.5-1 day per subject manual]**

Mask creation involves loading the *.spt file generated during the preprocess[ASAP] step into ASAPP. This can be done via the command line or using the ASAPP graphical interface. Call ASAP_start to set up the appropriate paths. Call ASAPP to launch the interace. The process of creating a mask involves 5-6 steps. Since ASAP is a CardViews clone, these steps are similar but not necessarily identical.

6.1. Contour edit [optional]
6.2. Segmentation edit
6.3. Sulcal drawing
6.4. Node labeling
6.5. Parcellation
6.6. Region labeling

Once these steps have been completed, a mask can be extracted from the project file.

7. **Create Mask [FreeSurfer] [1-2 hours manual, 12 hours computer]**

The batch aspect of this process is carried out by the function **roi_RAT2FS** (see the file for details). The function converts the affine normalized scan to FreeSurfer and then launches a set of FreeSurfer shell scripts to perform the various stages of FreeSurfer processing as described in the output of:

recon-all --help

This command can be carried out in various stages. This is needed as

FreeSurfer requires manual editing in the middle of the process. For details of each individual stage, see the help of roi_RAT2FS. The final stage of the process classifies the surface into regions and converts it back to a mask used in roi analyses.

8. **Extract ROI data [15-30 minutes per subject]**

First, modify **roi_setup_ABCD** script to point to the location of the mask data and then execute the script to add the relevant fields to the roisession structure inside ABCD_object. [Remember to call RAT2_start/freesurfer_start followed by adding the scripts directory to the path].

```
ABCD_object = roi_setup_ABCD(ABCD_object);
```

Once the pointers to the mask files have been embedded, the next step is to extract the data from the functional runs corresponding to the regions defined by the masks. This is done with **roi_create_ROI_data** [See the scripts help for details].

```
ABCD_object = roi_create_ROI_data(ABCD_object);
```

Save the expt object.

9. **Create ROI project [less than a minute]**

Once the ROI data has been extracted, we need to create an ROI project. Ideally this will be eliminated in the future. Currently, this is the process we use to combine two separate branches of the toolkit. Create a directory called ABCD.roi and switch to it as the working directory. Call the following function:

```
spm_ROI_subjects(sids, 'ABCD_object filename');
```

The first parameter selects which subjects' to perform ROI analyses on. The second parameter provides the location of the Matlab file storing the ABCD_object. This function creates an spm_ROI project file that stores information from for conducting ROI analysis.

10. **Results**

With the first 9 steps completed the stage is set for estimating the model and evaluating the contrasts.

10.1. **SPM2 results**

10.1.1. Create a new directory, say, ABCD.spm2 and make that the current directory. Load the ABCD_object and run the following to perform the SPM anlysis:

roi_fixedfx_subjects(ABCD_object).

The function will perform design creation, model estimation, and contrast evaluation. With only ABCD_object as input to the function, analysis will be done on all subject data currently in the object. A subset of subjects can also be analyzed by adding a parameter which contains a vector of subjects, e.g., roi_fixedfx_subjects(ABCd_object, [1:3]). See the help for roi_fixedfx_subjects for details.

10.1.2. Once the contrasts have been evaluated one can create SPM results figures for all the contrasts using the function:

roi_spmFigures(ABCD_object)

Currently, the various parameter values for creating the figures are embedded in m-file itself. These will be taken out an made part of the @experiment object.

10.2. **ROI results**

10.2.1. To perform the analyses call **spm_ROI_model** (???). This will launch a gui for you to select the *.roi project file created earlier. Once you select the file it will evaluate the model for each subject and compute the contrasts.

10.2.2. Once the contrasts have been computed, you call **spm_ROI_results** to evaluate results across subjects. Without any arguments, the function will ask the user for various options:

10.2.2.1. how to perform the across subject analyses

10.2.2.2. which ROIs to use

10.2.2.3. which contrasts to evaluate

10.2.2.4. how to plot the results

10.2.3. In addition to the above plotting routine, if one uses the SpeechLab ROI structure additional plotting routines are provided (**roi_Fig_create**). The output of the spm_ROI_results function is a structure that can be provided as an argument to these routines to plot pseudo-brain ROI analyses results as shown in the figure below.

Appendix A: Help files

Help files for the various commands are listed here. You can also get this help by typing help command_name at the Matlab prompt

1. roi_preprocess_subjects

```
function expt = roi_preprocess_subjects(expt,sid,type,FLAG)
% EXPT = ROI_PREPROCESS_SUBJECTS(EXPT) preprocesses each
% subject specified in the EXPT structure to perform realignment,
% coregsitration, normalization (both affine and full) and
% smoothing (in case of full normalization). It returns the updated
% experiment structure containing information about preprocessed
% files. The data is created in subfolders called 'affine' and
% 'full' at the location of the original data.
%
% EXPT = ROI_PREPROCESS_SUBJECTS(EXPT,SID) allows one to specify
% which subjects' should be preprocessed. If SID is left empty all
% subjects are preprocessed.
%
% EXPT = ROI_PREPROCESS_SUBJECTS(EXPT,SID,TYPE) specifies whether
% the normalization is only 'affine' or 'full' (nonlinear) or
% 'both' (default).
%
% EXPT = ROI_PREPROCESS_SUBJECTS(EXPT,SID,TYPE,FLAG) controls which
% step of the process to execute. FLAG is a 5 element boolean
%
vector:[doRealign,doCoregister,doANormalize,doFnormalize,doSmooth].
% Each of these flags determine whether the process should be
% performed (1) and/or whether the file pointers should be updated
% (1/0).
%
% See also: ROI_REALIGN_SUBJECT, ROI_COREGISTER_SUBJECT,
% ROI_AFFINENORMALIZE_SUBJECT, ROI_FULLNORMALIZE_SUBJECT,
% ROI_SMOOTH_SUBJECT, @EXPERIMENT, @SESSION, EXPT_SETUP_DEMO
```

2. roi_RAT2ASAP

```
function expt = roi_RAT2ASAP(expt,prefix,sid,FLAGS)
% EXPT = ROI_RAT2ASAP(EXPT) takes the affine normalized structural
% image information for each subject stored in the EXPT object and
% preprocesses it for use with ASAP.
%
% ROI_RAT2ASAP(EXPT,SID) specifies which subjects' to preprocess
% for ASAP using a vector SID. If left empty all subjects are
% processed.
%
% ROI_RAT2ASAP(EXPT,SID,FLAGS) controls the two steps of this
% routine. Currently FLAGS is a 3 element boolean vector:
% [doPreprocess, doCreateMask, doForceMask]. The first flag
% converts the data into an ASAP project file. The second step
% takes an ASAP project file and converts it to an ROImask. The
% third option forces a recreation of the maskfile even if it does
% exist.
%
```

```
% See also: ROI_ASAP_PREPROCESS, SAP_COMMAND
```

3. roi_RAT2FS

```
function maskfile = roi_RAT2FS(expt,prefix,sid,FLAGS)
% MASKFILE = ROI_RATFS(EXPT,PREFIX) takes the structural image
% information for each subject stored in the EXPT object and
% creates a FreeSurfer directory prefixed by the provided PREFIX as
% (PREFIX.NN). The process starts with converting the structural
% image into FreeSurfer format and ends with converting a
% classified FreeSurfer surface into volume mask whose filename is
% returned in MASKFILE. The first two arguments are compulsory.
%
% ROI_RAT2FS(EXPT,PREFIX,SID) specifies which subjects' to process
% in FreeSurfer using a vector SID. If left empty all subjects are
% processed.
%
% ROI_RAT2FS(EXPT,PREFIX,SID,FLAGS) provides a finer grained
% control on every step of the process. Currently FLAGS is an 8
% element boolean vector: [doCreateDir, doConvertImg, doPreprocess,
% doCreateSurf, doStage3, doStage4a, doStage4b, doFS2RAT]. Each
% flag is described below.
% doCreateDir  = Creates freesurfer subject directory
% doConvertImg = Convert Analyze image to FreeSurfer .COR format
% doPreprocess = pre-manual  editing (mc, tal, norm, strip, seg,
stage2)
% doCreateSurf = with manual editing (fill, tess, sm1, inf1)
% doStage3     = runs the topology fixer (and stage2)
% doStage4a    = make final surfaces
% doStage4b    = spherical morph and auto-classifier
% doFS2RAT     = convert surface back to mask
%
% See also: ROI_FS2RAT
```

4. roi_create_ROI_data

```
function expt = roi_create_ROI_data(expt,sid,FLAG)
% FUNCTION EXPT = ROI_CREATE_ROI_DATA(EXPT) extracts ROI data from
% the functional runs for all subjects based on the masks stored in
% the roisession structure (see roi_setup_expt.m for details). The
% extracted data is stored in the same directory where the mask
% file is generated/stored and is indexed by subject name and
% session.
%
% EXPT = ROI_CREATE_ROI_DATA(EXPT,SID) allows one to specify which
% subjects' data should be extracted. If SID is left empty all
% subjects' data are extracted.
%
% EXPT = ROI_CREATE_ROI_DATA(EXPT,SID,FLAG) controls the process of
% data extraction. FLAG is a 2 element boolean vector:
%                [doExtract, doSmooth]
% The first element controls data extraction. In the event that one
% has already generated the data, leaving this flag unset (0)
% prevents reextraction of data. However, the fields of the EXPT
% structure are filled with the appropriate data pointers.
```

```
% [TODO: doSmooth is a redundant flag currently. In the future, it
% will provide within the region smoothing. The second flag is
% ignored if the first flag is set to 0.]
%
% See also: @EXPERIMENT, ROI_SETUP_DEMO
```

5. roi_fixedfx_subjects

```
function roi_fixedfx_subjects(expt,sid,doFlag)
% ROI_FIXEDFX_SUBJECTS(EXPT) is a wrapper around SPM that takes the
% information about the experiment and subjects embedded in the
% EXPT object and performs 3 operations: SPM fMRI design creation,
% model estimation and contrast evaluation.
%
% ROI_FIXEDFX_SUBJECTS(EXPT,SID) allows one to specify which
% subjects' should be considered as part of the analysis. An empty
% value of SID performs the steps on all the subjects.
%
% ROI_FIXEDFX_SUBJECTS(EXPT,SID,DOFLAG) controls which steps of the
% process to perform. DOFLAG is a string input that takes the
% following options: 'all'[default], 'design', 'estimate',
% 'contrast'.
%
% See also: ROI_FMRI_DESIGN, ROI_ESTIMATE, ROI_CONTRAST
```

6. roi_spmFigures

```
function roi_spmFigures(varargin)

% This function creates and saves spm style figures for each contrast
% you've specified.
% Inputs:
% ========================================================================
% usage: roi_spmFigures(expt,[valid])
% expt : The expt object for your study (i.e. RAT2 processing stream)
%
% valid : Optional argument, a vector of length length(expt.contrasts)
%         specifying which ontrasts to draw and save (i.e. a one in
%         valid(n) to draw contrast n.  If not specified, draw ALL contrast
%         images!
%
% Below are the defaults for figure generation.  If you change these here,
% they will be set in each spmParams struct.  You can override these for
% each individual contrast figure below.
% Description:
% ========================================================================
% pAdj :        p value error correction (none, 'FWE' family wise error
%               correction, 'FDR' false discovery rate)
% pThresh :     cutoff for significant voxels (SPM defaults to 0.05 for
%               error corrected (FWE|FDR), 0.001 for none.
% xThresh :     extent threshold (minimum # voxels in a cluster)
% brighten :    controls aesthetics of rendered 'blobs'.  Use NaN for spm96
%               style yellow/red blobs, 1 for red blobs (no brightening)
%               down to 0 for a lot of brightening of red blobs.
% template :    which brain template to render on.  Choose 'spm96' for old
%               spm brain including medial surface, 'smooth' for a smoothed
%               average brain, 'single' for a typical single brain.
% tabulate :    1 to create and save summary tables of active clusters, 0
%               for no tables (just brain images)
```

```
% subdir   :    name of a directory in the cwd you wish to create and place
%               the figures in.  Can not override this per contrast!
% filename :    Do not enter a default here.  The default is to trim out
%               white spaces from your contrast name, and prefix with
%               'cImg_' (contrast image).  You can override this in each
%               individual contrast if you like, but be careful!
```

7. spm_ROI_subjects
8. spm_ROI_model
9. spm_ROI_results
10. roi_ Fig_create

Appendix B: Other tools

SpeechLab specific information

Howto define sesscov

TODO

1.Provide a feature comparison with CardViews