

*froi*

**FS-FAST region of interest**

instructions for use

written by nick knouf

Copyright © 2003, Nicholas Knouf & MIT.

All rights reserved. This manual may be reproduced as much as you want; I'm not going to restrict your rights at all. If you post it on another website, drop me a line, okay? Or send me a postcard. I like getting mail. No lawyers were involved in the production of this work. Not tested as much as it can be; I don't have that sort of budget. Please don't sue me if it does something bad to your machine.

Set in Adobe Garamond Pro and News Gothic MT. Designed using InDesign 2.0. Created on a Macintosh and OS X.





# table of contents

|                                  |   |
|----------------------------------|---|
| requirements                     | 1 |
| getting and unpacking the source | 2 |
| installing the perl modules      | 2 |
| setup environment for froi       | 4 |

## **introduction** 5

|      |   |
|------|---|
| froi | 6 |
|------|---|

## **summary of analysis steps** 9

|                       |    |
|-----------------------|----|
| a note on terminology | 11 |
|-----------------------|----|

## **froi basics** 11

|                              |    |
|------------------------------|----|
| creating an roi              | 12 |
| a note on logical roi naming | 13 |
| editing an roi               | 14 |
| computing the roi average    | 16 |
| viewing your roi results     | 18 |
| next                         | 19 |
| combining multiple rois      | 23 |

## **further ROI tools** 23

|   |    |
|---|----|
| viewing roi overlap                             | 24 |
| mapping an roi to the surface                   | 26 |
| mapping a label to an roi                       | 27 |
| combining labels                                | 27 |
| creating and viewing a selectivity map          | 28 |
| listing rois                                    | 29 |
| creating and viewing a spatial activity profile | 30 |

## **logfiles** 33

## **roi format** 35

synopsis 37  
description 37

**mannual pages for scripts** 37

# chapter 0

## installation

Installation of froi, while not a walk in the park, should not pose difficulties to those who have installed the FS-FAST and Freesurfer distribution. Because of the connection between froi and FS-FAST there is relatively minimal configuration needed to make the tools work.

### **requirements**

froi has not been tested on a wide variety of platforms, but there is nothing that should prevent it from working in many places, including almost all Unices such as Linux, xBSD, Darwin, Solaris, and Irix. It was developed on Linux running Matlab 6.5 (R13) with perl version 5.6.1. I assume that it should work for all versions of Matlab that work with FS-FAST and with versions of perl down to 5.2 or so. If you find any version incompatibilities, please let me know.

## getting and unpacking the source

1. You can download the source from the froi website, <http://froi.sourceforge.net>.
2. Move the file to the directory just above where you want froi to live:

```
mv froi-source.tar.gz /usr/local
```

3. Unzip the distribution:  
`gunzip froi-source.tar.gz`

Next untar the archive:  
`tar xvf froi-source.tar.gz`

You will find a directory called “froi”. Within this directory is the complete distribution, including some additional modules that need to be installed for froi to work.

## installing the perl modules

To help keep the design modular, and to allow future developers a reusable framework, I created Perl modules that abstract a large amount of the code. These modules must be installed in a place that is accessible to the perl scripts that form the basis of the froi analysis.

There are three modules to install (FSFAST::Parse, FSFAST::ROI, and FSFAST::Misc) and the procedure is the same for all.

1. Unzip and untar the module source:

```
gunzip module-name.tar.gz  
tar xvf module-name.tar.gz
```

At this point you have two options. If you are the system administrator for this computer, you can do the following to install the module into your system-wide perl module directory:

2. Build the module and manual page:



```
perl Makefile.pl
make
make test
```

3. Change to root and install:

```
su root
make install
```

If, however, you are not the administrator and/or want to install the software in a non-standard place, you can do the following:

2. Tell the install script where to put the module; this location must be writable by you:

```
perl Makefile.pl PREFIX=/place/to/install/module
```

3. Build the module and manual page and install:

```
make
make test
make install
```

At this point the module will exist under `/place/to/install/module`. To let perl know the module exists, however, you have to do one more thing. You need to set the environment variable `$PERLLIB`, which differs depending on what shell you are using. For `csh` or `tcsh`, put the following in your `.environment` file:

```
setenv PERLLIB /place/to/install/module/lib/site_perl/pervar/
```

where `pervar` is your perl version, perhaps 5.6.1.

For other shells, consult the documentation to see how to set environment variables.

You will need to do this for each module in the distribution. Later releases will streamline this into a single-step process.

note: The scripts assume perl is located at `/usr/bin/perl`; if it's not there, you can create a symbolic link from `/usr/bin/perl` to your actual perl executable, or you can change the shebang lines in each perl script. Again, later releases of `froi` will automatically determine the location of perl.

## setup environment for froi

There are two final steps before you can use froi. First, you have to put the froi scripts in your path. For csh or tcsh, type (replacing the location of froi with your own location):

```
set path = ( $path /usr/local/froi/bin)
```

For other shells, consult your documentation.

Finally, you have to tell matlab where to find the scripts. In your home directory there should be a directory called “matlab”. If it doesn’t exist, create it. Within that directory, create or edit a file called “startup.m” and add the following:

```
addpath /usr/local/froi/toolbox
```

Again, replace the location of froi with your own installation directory.

At this point, logout and log back in again and you should be able to use froi!

If you have any problems, send a detailed description of the problem to [froi-bugs@sourceforge.net](mailto:froi-bugs@sourceforge.net).

# chapter 1

## introduction

Freesurfer and FS-FAST (Freesurfer Functional Analysis Stream) are software suites developed at the Massachusetts General Hospital imaging center for the analysis of functional MRI (fMRI) data. **\*\*insert freesurfer and fs-fast refs\*\*** FS-FAST is a set of tools for the analysis of slice-based fMRI data, including pre-processing (motion correction, image normalization, spatial smoothing), design specification (including gamma and finite impulse response (FIR) models), contrast specification, and statistical map creation. In a complimentary fashion, Freesurfer creates 2D surface- and 3D volume-based reconstructions of the brain. Data from the slices can then be mapped onto a previous- or current-session anatomical, or on a 2D surface, such as an inflated or flattened representation. Surfaces can also be registered into “spherical” and Talairach coordinate systems.

The advantages of FS-FAST and Freesurfer over other fMRI analysis packages are numerous. The tight integration of the two complimentary suites provides for relatively easy transformation of data from one representation to another. The majority of the programs are command-line based, allowing for easy scriptability as well as transparency of the analysis process for the user,

something that is sometimes difficult with other packages. With FS-FAST and Freesurfer there is no need for an individual subject's brain to be automatically smoothed and transformed into Talairach space; all analyses can be done in the subject's native slices.

However, FS-FAST is lacking one major feature, that being the analysis of regions of interest (ROIs). ROI analyses are an important part of functional MRI research. **\*\*insert old nancy refs\*\*** An ROI approach proceeds as follows:

- 1) Define a functionally-specific region by a “localizer” scan. For example, you might be interested in areas that respond more to faces than to objects, perhaps the fusiform face area (FFA) **\*\*insert FFA refs\*\***
- 2) Perform an experiment that tests your condition of interest; for example, you might be interested in whether or not the FFA responds differently to upright faces versus inverted faces.
- 3) Using the ROI defined in 1), look to see if there are any differences between conditions in 2).

The advantages of an ROI approach are numerous. For one, because each subject's anatomy is different, anatomical ROIs are usually not sufficient for most analyses. Defining an independent functional ROI allows your analysis to be tailored to the idiosyncrasies of the particular subject's brain. As well, since you are only looking in a small region, you do not have the problems associated with whole brain analyses. **\*\* are there problems? i think there are... \*\***

## froi

To this end, I wrote a suite of programs to do functional ROI analyses in conjunction with FS-FAST and Freesurfer called **froi**: FS-FAST ROI. The meat of the suite is to allow one to functionally define ROIs and then extract information from the ROIs, such as mean time-course (for blocked data) as well as mean and percent signal change for each condition (along with the associated standard deviations and errors). ROIs are saved in a space-efficient format and can be edited quite easily. The results of an ROI computation can be viewed using a Matlab GUI and are saved in a Matlab structure. In addition, there is an option to create tab-delimited text files of the results for importation into other programs like Excel or SPSS.

The functional definition of an ROI can be done in three ways: by  $t$  value, by

significance (sig) value, and by false discovery rate (FDR) **\*\*insert FDR refs\*\***. Defining an ROI by  $t$  or sig value is done in a simple threshold fashion; all values greater (or lower, in the case of negative values) are taken to be in the ROI. Using FDR you provide a specific false discovery rate and a hypothesis of how you think the noise is correlated, and all voxels above the FDR-calculated  $p$  value are included in the ROI.

Such ROIs should not be taken verbatim into the subsequent statistical analysis, however. Many times the ROIs will include scattered and non-contiguous voxels throughout your slices, or may contain multiple functional regions that you would like to analyze separately. After masking the image with a  $t$ , sig, or FDR threshold, you need to edit the thresholded map in order to define the appropriate ROIs. Once these ROIs are created, you are ready for further statistical analysis.

Data analysis within the ROI uses the results of the **selxavg** program of FS-FAST. **selxavg** computes the estimated deviation from baseline for each condition for each voxel in your volume. Depending on the type of analysis (either gamma-fit or FIR), you may have only one point for each condition (gamma) or a time-course (FIR). ROI analyses proceeds by averaging these data over your ROI and computing the appropriate standard deviation and standard error. These results can then be viewed or taken to a third level of analysis, such as a group analysis.

**froi** provides many other ROI functions. For example, it can combine multiple ROIs (up to three) by intersection or union. As well, it can create an overlap map of ROIs (up to three). Finally, it can create a “selectivity map” for one condition versus another, thus allowing you to see how the selectivity of voxels varies within and across slices.

Slice-based ROIs provide a simple solution to the problem of converting your data to another coordinate system: there is no problem. However, there are some times when you might want to view an ROI in a different coordinate system; for example, to look at stability across sessions or to perform a group analysis. **froi** provides scripts to simplify these tasks. It can take an ROI and convert it into a Freesurfer label while also taking a label and converting it to a **froi** ROI. As well, it can combine labels in much the same way as you can combine ROIs; however, as of the current release, it only works for two labels and only for the intersection method.











# chapter 2

## summary of analysis steps

The following steps are common to ROI analysis with **froi**:

1. Preprocess your data using FS-FAST.
2. Create an analysis.
3. Run selective averaging.
4. Run **make\_roi** to create a simple thresholded ROI using  $t$ , sig, or FDR values. Immediately after creating the thresholded ROI, you will be given the chance to edit it to your liking and save as a separate ROI.
5. Further edit your ROIs as needed using **edit\_roi**.
6. Run **compute\_roi** to compute the ROI average.
7. Run **view\_roi\_results** to see the result of the ROI computation.

For some analyses, you may want to consider one or more of the following steps:

-  Create further ROIs as needed.
-  Combine multiple ROIs together and compute an ROI average.
-  Create an overlap map for multiple ROIs and view using **display\_overlap**.
-  Compute selectivity indices and create a selectivity map for one condition versus another.
-  Transform your slice-based ROIs to volume- or surface-based labels using **roi2label**.
-  Combine your labels using **combine\_label**.
-  Convert a volume- or surface-based label to a slice-based ROI using **label2roi**.
-  Perform step 6 with your new ROI.



# chapter 3

## froi basics

This chapter aims to explain the basics of using **froi** for the analysis of functional data in association with FS-FAST. To begin, however, we need to speak the same language.

### a note on terminology

As with any new software package, there is some new terminology to be learned. **froi** is no different. Let me explain some of the terms and abbreviations you will see throughout this manual:

`analysis` -- name of an FS-FAST analysis that has been run through at least the `selxavg-sess` step.

`contrast` -- name of an FS-FAST contrast that has been run through at least the `stxgrinder-sess` step

`roianalysis` -- an FS-FAST analysis that was used to create an ROI

`func_path` -- path to your functional data. In FS-FAST parlance, this would default to `/session_directory/session_id/bold`.

`roi_path` -- path to your ROIs. By default, this is `/session_directory/session_id/bold/roianalysis_ROI`. The addition of the ROI suffix is to easily differentiate the directory in which your ROIs are stored from the original FS-FAST analysis.

`label_path` -- path to your labels created by `make_label` or `roi2label`. This path is by default `/session_directory/session_id/bold/roianalysis_ROI/label`.

## creating an roi

To create an ROI, use the **make\_roi** program. Complete documentation for this program is presented in the Appendix; I will outline a few specific cases here.

**make\_roi** requires five parameters and accepts many more; in most cases, you will give it more than the five required parameters. The necessary parameters are:

1. `-analysis analysis_name`

This is the name of a FS-FAST analysis created after `mkanalysis-sess` has been run.

2. `-contrast contrast_name`

The name of the contrast to use to create the ROI. These contrast maps are created after you have run `stxgrinder-sess`.

3. `-roi roi_name`

The name you wish to give this ROI. See the section below for suggestions on how to name your ROIs.

4. `-sf sessid_file`

The session ID filename.

5. `-df sessdir_file`

The session directory filename.

These five parameters will create a ROI that threshold your statistical map at a significance value of  $10^{-2}$ . Sometimes, however, you are interested in taking a

higher threshold or using a different statistical map. In those cases, you can give one or more of the next three parameters:

6. `-map <t | sig>`

The statistical map to use to create the ROI. The default FS-FAS<sub>t</sub> stat maps are a *t*-map and a sig-map. **make\_roi** defaults to using the sig map.

7. `-threshold value`

Value to use for the threshold.

8. `-unsigned`

This parameter will consider your statistical map to be unsigned. Thus, if you give a threshold for a *t*-map of 2, all voxels with a *t* value greater than 2 or less than -2 will be selected for the ROI.

As an alternative to thresholding the *t* or sig map, I have implemented thresholding by false discovery rate (FDR). \*\* insert ref and brief explanation of FDR \*\*

To use the FDR method, you can give one or both of the following parameters:

9. `-fdr value`

False discovery rate to use. Usually a FDR of 0.05 will give a reasonable ROI; see the references for some empirical studies.

10. `-np`

This parameter indicates that you wish to use the non-parametric calculations when using FDR. See the references for more details on when you might want to use this.

There are a few other parameters that can be given to **make\_roi** that allow you to store data in non-standard locations; only use these parameters if you know what you are doing; as well, these parameters have not been extensively tested.

## a note on logical roi naming

The method described above will create an ROI that is a simple threshold of your contrast map. Many times this will include more voxels than you actually want in your ROI. Since it is impossible for the computer to know which areas you want included in your final ROI, you will need to select a subset of the voxels that are included in this initial ROI. Thus **make\_roi** immediately starts `edit_roi`, which allows you to edit this over-broad ROI.

When naming your initial ROI, it's useful to name it based on the statistical map and threshold you are using. If you are thresholding a t map with values above 4, name your ROI (using the `-roi` parameter) as `tover4`. If you are using an FDR of 0.05, name your ROI `fdr005`. Of course these are just guidelines, but it will help simplify things down the road.

## editing an roi

Following the creation of an ROI, **make\_roi** immediately starts `edit_roi`, which can also be ran on its own from the command line. To use, `edit_roi` requires four parameters:

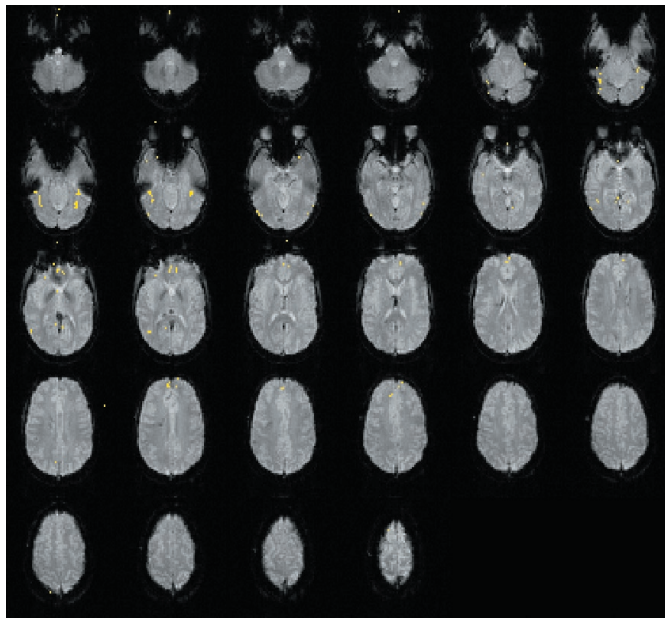
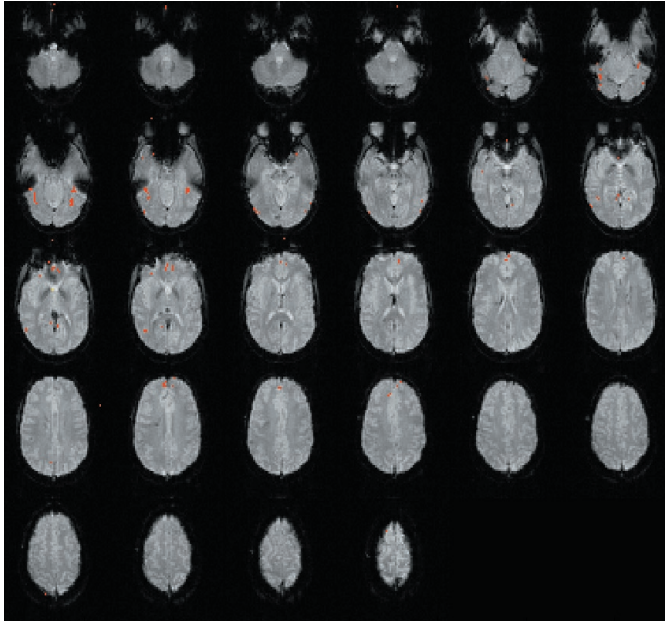
1. `-analysis`  
Name of the analysis used to create the ROI
2. `-roi`  
Name of the ROI
3. `-sf sessid`  
Name of the session ID file.
4. `-df sessdir`  
Name of the session directory file.

`edit_roi` begins with all voxels in the loaded ROI being selected (Figure 1). Selected voxels of the current ROI are shown in red, while unselected voxels of the current ROI are shown in yellow. To clear all the voxels, press "c"; to select all voxels, press "a". There are three modes in `edit_roi`; to change your mode, press "m":

1. None; do nothing
2. Select: select voxels
3. Select/Deselect; select a voxel if it is not selected, deselect a voxel if it is selected.

As well, there are a number of different brush sizes you can use, all the way from 1x1 voxels up through 11x11 voxels. To change the brush size, press "b" repeatedly until you select your desired brush size.

To simplify the creation of ROIs, you can use the brush tool to select relatively large regions of cortex and then compute the intersection of those regions with the underlying map. Thus, you do not have to be unnecessarily careful when



*Figure 1. The top image shows the creation of a ROI thresholded at a significance value of 4 with all voxels selected. The bottom image is the same with all voxels deselected.*

selecting your ROI; simply type “i” to compute the intersection (Figure 2).

To save your ROI, type “s”; the default save location is in the directory `func_dir/analysis_ROI`. Note that there is no need to add an extension for the filename; extensions are added automatically. For detailed information about the ROI format, please see the appendix.

Pressing “q” will quit `edit_roi` and warn you if you have not saved your ROI.

## computing the roi average

Once your ROI is defined, it is time to compute the average response over all the voxels in the ROI. The program that performs this computation is `compute_roi`. This program works for both event-related and blocked data, as well as for gamma fitted and finite impulse response (FIR) analyses.

`compute_roi` has similar syntax to both `make_roi` and `edit_roi`. There can be confusion in the terminology, so read the following descriptions of the parameters carefully:

1. `-analysis analysis_name`

Name of the analysis you wish to use for the ROI computation. You can use one ROI for multiple analyses; in fact, this is one of the strengths of the ROI approach. Of course the usual caveats apply, in that you must have the same slice prescription for all analyses that use the same ROI.

Thus if you wanted to compute the ROI average over the voxels in the “face-event” analysis, you would use “-analysis face-event” for this parameter.

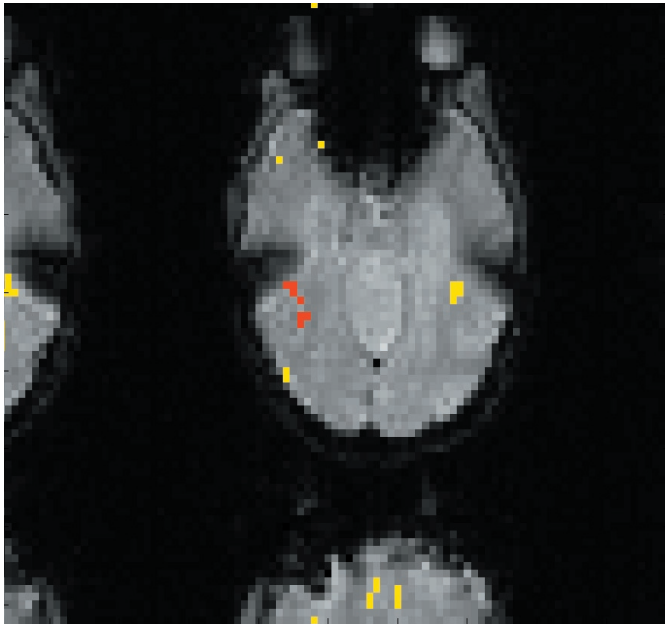
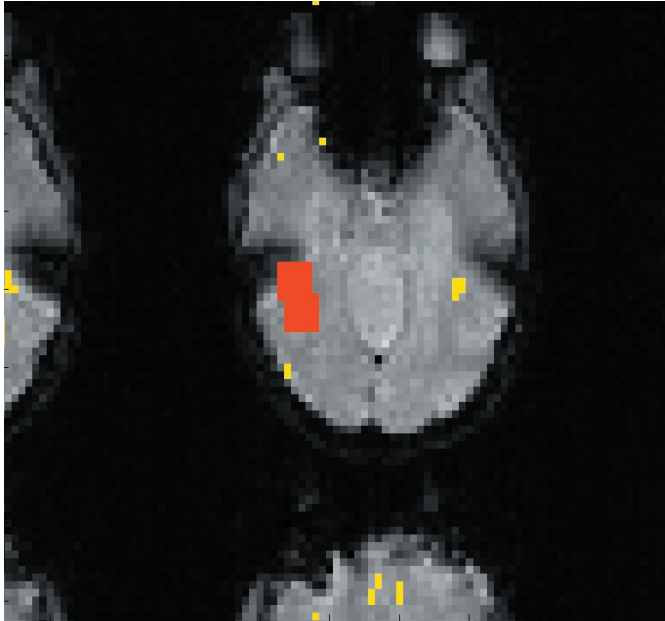
2. `-roianalysis roianalysis_name`

Name of the analysis under which you created your ROI. Note the difference between this parameter and the previous one. This corresponds to the name of the analysis used for your localizer runs, perhaps “face-blocked”. For different ROI computations across different analyses but using the same ROI, you would use the same roianalysis name.

3. `-roi roi_name`

The name of the ROI to use in the computations.

4. `-sf sessid`



*Figure 2. Creating an ROI before (top) and after (bottom) computing the intersection of the overlay (red).*

The name of the session ID file.

5. `-df sessdir`

The name of the session directory file.

`compute_roi` will produce different results based on the nature of your analysis. For example, if you have a blocked analysis, you will get the average raw time-course for your ROI in addition to amplitudes for each condition (gammafit analyses) or time-courses (event-related analyses). Note that the raw time-course is just that, raw without detrending or noise reduction. For an event-related analysis you will not get the raw time-course (which would be meaningless), but you will get the same results as for blocked data. `compute_roi` is not designed to work with the `abblocked` or `retinotopy` analyses.

All of the results of `compute_roi` are saved in a Matlab structure for easy access. Included in this structure is the results of the ROI computation, information regarding the analysis, the ROI, the number of voxels in the ROI, etc. This structure can be accessed directly in order to view the results of the computation. If you prefer, however, you can use `view_roi_results` (see next section) to see the results.

There are a few other parameters (many of them similar to **`make_roi`** and `edit_roi`) that can be passed to `compute_roi`. A useful one is `-text`; this will output the results of the ROI computation as text files for easy importation into other programs, such as Excel or SPSS.

For more information, see the `compute_roi` manual page either in the Appendix or by typing `compute_roi -man`.

## viewing your roi results

Now that your ROI has been created and the ROI averages have been computed, you probably want to see the results. The most direct way to do that is through the `view_roi_results` program (Figure 3). This will load the results of `compute_roi` and make them accessible through a Matlab graphical user interface (GUI). You will be able to view the mean response with or without standard deviation or standard error, along with percent signal change with or without standard deviation or standard error (Figure 4). If your analysis was blocked, you will also be able to see the raw time-course for each run.

The parameters for `view_roi_results` are similar to previous programs and are



probably familiar by now:

1. `-analysis analysis_name`

Name of the analysis you wish to use for the ROI computation.

2. `-roianalysis roianalysis_name`

Name of the analysis under which you created your ROI.

3. `-roi roi_name`

The name of your ROI.

4. `-sf sessid`

The name of the session ID file.

5. `-df sessdir`

The name of the session directory file.

Note that you can load other `compute_roi` structures from the `compute_roi` window. As well, any graph can be saved and/or printed.

## next

This is the end of the basic ROI analysis. There is much more that can be done with your ROIs and which will be explained in the next chapter.

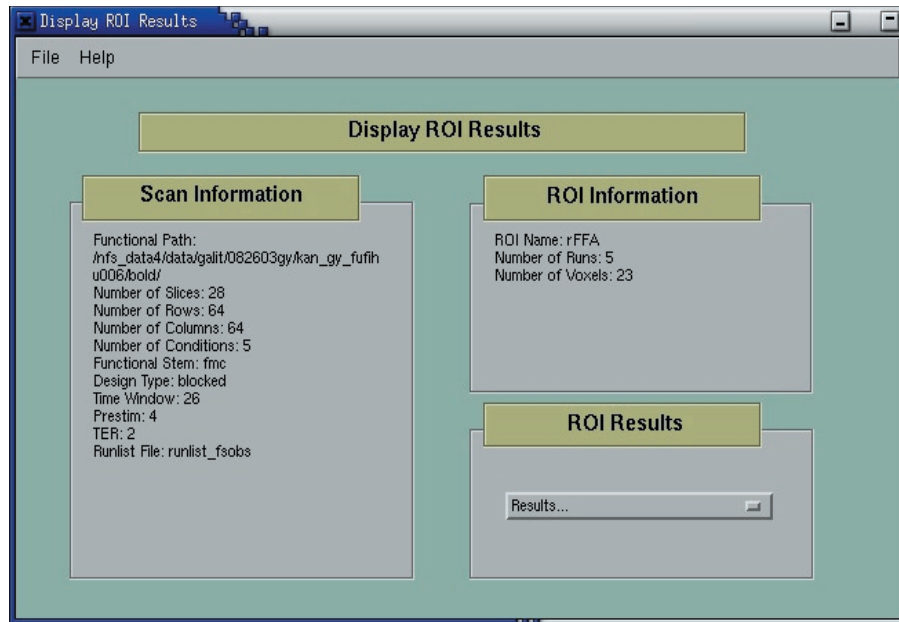
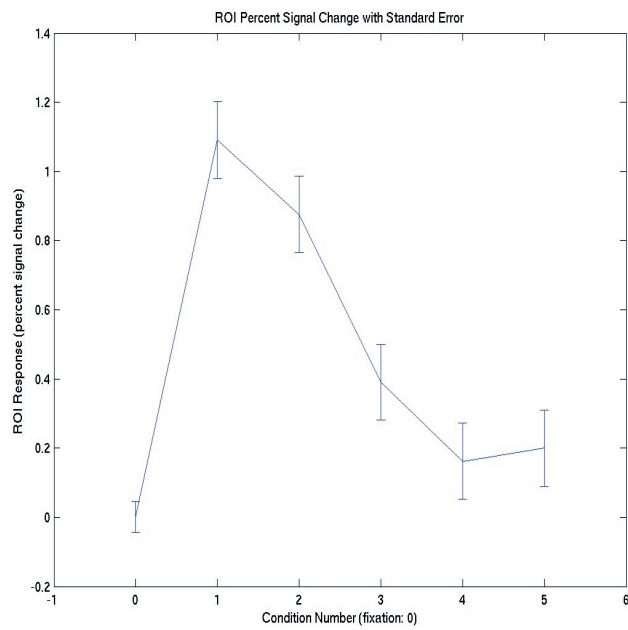
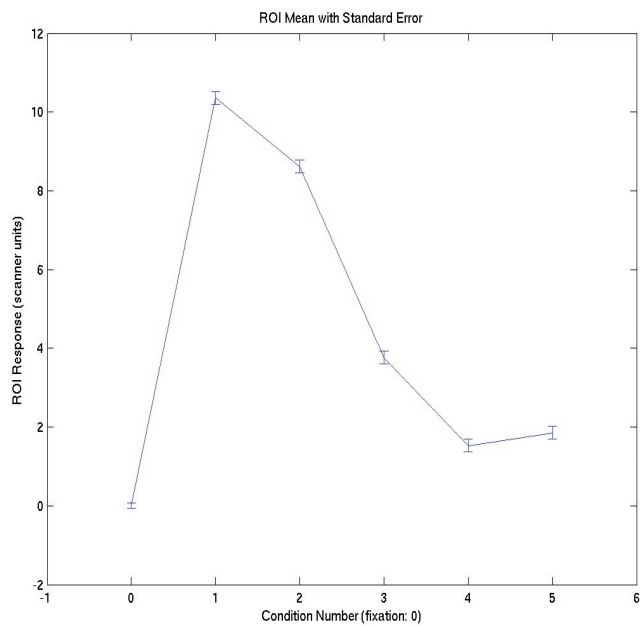


Figure 3. Graphical User Interface (GUI) for viewing ROI results. Top shows an interface giving information about the ROI and allowing the user to select different plots to display. Bottom shows an interface to view average ROI raw time-courses for blocked analyses.



*Figure 4. Sample ROI results with standard errors given in scanner units (top) and percent signal change (bottom).*



# chapter 4

## further ROI tools

### combining multiple rois

After using **froi** for a while, you will probably create a large number of ROIs. What happens if you want to take two or three ROIs, say in a similar region, and combine them together? Or, say you have an anatomical ROI that probably overlaps one of your functional ROI, and you only want to take those voxels in the functional ROI that are also in your anatomical ROI?

There exists a program called **combine\_roi** that lets you do this. Allowing up to three ROIs to be combined by either union or intersection, **combine\_roi** will output an ROI with the results of the operation. These ROIs can then be used just like any other ROI, so you can edit it, compute the ROI average, etc.

One caveat; at the moment, all of the ROIs must be from the same analysis to be used by **combine\_roi**. If you have an ROI from a different analysis, you will have to manually move it to your ROI directory.

The parameters for **combine\_roi** are intuitive:

1. -roianalysis roianalysis\_name

Name of the analysis used to create the ROI.

2. -inputroi ROI1,ROI2,ROI3

List of ROIs to combine. Note that the ROIs are separated by commas and there is no space between the comma and the next ROI. If you don't follow this format, then Bad Things will happen (where Bad Things are the program spitting out a cryptic error message and not working).

3. -outputroi roi\_name

Name of the output ROI.

4. -method <union | intersection>

Method of combining ROIs. Giving the “union” option will take voxels that appear in any input ROI, while intersection will only take the voxels that appear in all input ROIs. Thus, union is like the set operation OR while intersection is like AND.

5. -sf sessid, -df sessdir

Usual FS-FAST session ID and session directory filenames.

**combine\_roi** will chug away and produce ROIs that are the result of your desired method.

## viewing roi overlap

Say you have three conditions of interest, all of which are analyzed by an ROI approach. At some point in your analysis you're interested in whether or not these ROIs overlap, if so, where, and if not, where are they separate. Using **combine\_roi**, you can create an overlap ROI which is then viewable using a program called **display\_overlap**. The only change to the command line for **combine\_roi** is:

4. -method overlap

Take the input ROIs and create an overlap ROI.

Once **combine\_roi** is finished, you can view the results using **display\_overlap**, which has similar parameters to `view_roi_results`:

1. `-analysis analysis_name`

Name of the analysis you wish to use for the ROI computation.

2. `-roianalysis roianalysis_name`

Name of the analysis under which you created your ROI.

3. `-roi overlaproi_name`

The name of your overlap ROI.

4. `-sf sessid, -df sessdir`

The usual FS-FAST session ID files and session directory files.

The colors depend on how many ROIs there are and which areas show overlap. For two ROIs, the color key is:

blue - ROI 1

red - ROI 2

white - overlap of both ROI

while for three ROIs the color key is:

blue - ROI 1

red - ROI 2

yellow - ROI 3

purple - ROI 1 and ROI 2

orange - ROI 2 and ROI 3

green - ROI 1 and ROI 3

white - overlap of all ROIs

This color key is available also within the program by typing “c”.

As well, you can see a legend for each condition by typing “l”; thus you can see what condition maps to which ROI. Basically, ROIs are taken in the order that they appeared on the command line in **combine\_roi**.

**display\_overlap** only works with up to three ROIs; any more, and the color-map for overlap becomes exponentially large. Anybody who wishes to implement this for more than three ROIs, feel free to do so and send me a patch.

## mapping an roi to the surface

Slice-based ROIs are all well and good; they give you a subject-defined region in which to do analyses. However, there are times that you'd like to look across sessions, or perhaps look in different spaces such as spherical, Talairach, or in an average subject's brain. For that, you might be interested as to where the ROI lands in the volume or on the surface; this mapping of ROI to a surface- or volume-based label is accomplished through **roi2label**.

**roi2label** requires, like all surface- or volume-based analyses, that the subject in question have a reconstructed brain. As well, their functionals need to have been registered to this brain and there must exist a "subjectname" file with the name of the subject's reconstructed brain under the \$SUBJECT\_DIR, as defined by your Freesurfer configuration files. Therefore, before embarking on reconstructing every subject's brain in a study, ensure that you need (and are willing) to put in the time to reconstruct the brains.

**roi2label** has a relatively simple syntax with a couple pitfalls to be aware of:

1. -roianalysis roianalysis\_name

Name of the analysis under which you created the ROI.

2. -roi roi\_name

Name of the roi you'd like to map to a label.

3. -hemi <rh | lh>

Hemisphere you'd like mapped to a label.

4. -sf sessid -df sessdir

The usual session IDs and session directory files.

The main concern is to ensure that the hemisphere that you'd like to map your ROI to actually has voxels in the ROI. For example, if your ROI is for the right FFA, mapping that ROI to the left hemisphere is meaningless, as there are no left hemisphere voxels in an ROI for the right FFA. Most times such a mistake will be caught by the program and an error message will be displayed; however, if the program spits out a strange error message, be sure that you have not made such an error.

The labels are saved in a directory underneath your ROI directory named by de-



fault “label”. The label filename is of the form: roiname-hemi.label, thus for an roi rFFA and right hemisphere mapping, the label name will be rFFA-rh.label.

## mapping a label to an roi

You can also go in the reverse direction. Perhaps you have a multi-subject label that you’d like to map to individual subject’s ROIs. It’s possible to transform this label into an ROI using the program **label2roi**.

**label2roi** expects the label to be in your roi\_analysis/label directory. The name of the label should be in the format: stem-hemi.label, e.g., rFFA-rh.label. The command-line paramters are nearly identical to **roi2label**, only replacing the “-roi” parameter with “-label”:

1. -roianalysis roianalysis\_name

Name of the analysis under which you created the ROI.

2. -label label\_stem

Stemname of the label.

3. -hemi <rh | lh>

Hemisphere you’d like mapped to a label.

4. -sf sessid -df sessdir

The name of the resulting ROI will be the stem name of the label and will be saved in your roi\_analysis directory.

## combining labels

Much like with ROIs, you can also combine multiple labels together. At the moment, not as much work has been put into combining labels as with combining ROIs, to the options are quite limited: you can take two labels and combine them as one using intersection.

For more information see the **combine\_label** manual page in the appendix or by typing “combine\_label -man”.

## creating and viewing a selectivity map

Selectivity maps are another useful way to view the results of your fMRI analysis. While the topic of selectivity inevitably leads to the question of what index to use, intuitive indices can be used with excellent qualitative and quantitative results.

For our purposes a selectivity index is simply a measure of how selective a particular voxel is for a particular condition. For example, you may want to know which voxels are the most selective for faces; further, you would like to see how this selectivity drops off as a function of distance in a functionally defined region. **view\_sm** allows you to do this using one of several pre-defined selectivity indices (user-defined indices are planned for a future release). The selectivity indices included are:

1.  $(\text{preferred} - \text{non-preferred}) / (\text{preferred} + \text{non-preferred})$
2.  $(\text{preferred} - \text{non-preferred}) / (\text{baseline})$
3.  $(\text{preferred} - \text{non-preferred}) / (\text{preferred} + \text{non-preferred})$  in percent signal change
4.  $\text{abs}(\text{preferred} - \text{non-preferred}) / (\text{preferred})$  in percent signal change

Preferred and non-preferred, for lack of better terms, refer to the conditions in your selectivity index computation. For example, consider face selectivity (compared to objects). In this nomenclature, preferred would be faces and non-preferred would be objects.

aside: I understand that the terms “positive” and “non-preferred” are not entirely clear, however I’m not sure of better alternatives. Please let me know if you can think of better choices.

The actual computation of the selectivity map with **view\_sm** is quite simple and fast. The command-line parameters to **view\_sm** are:

1. -analysis analysis\_name

Name of the analysis to use to compute the selectivity.

2. -preferred #

The condition number (as defined in your paradigm file) for the “preferred” condition.

3. -non-preferred #

The condition number for the “non-preferred” condition.

4. -sf sessid -df sessdir

The usual session ID and session directory files.

This will compute a selectivity map using the first selectivity index described above. As well, it will default to showing a 90% range of the selectivity values. Since only a few voxels will show high selectivity indices, the distribution is sharply peaked. Empirical testing has determined that a value of 90% for this parameter, termed “-frac”, gives a good tradeoff between showing a decent number of indices versus being conservative in the display. You can change this parameter using:

5. -frac #

Fraction of selectivity indices to show.

To choose a different selectivity index, use:

6. -type #

Type of selectivity index to compute, where the type is given in the manpage.

The default view of the selectivity map is as a mosaic, but this can be changed by typing “s” and entering a valid slice number.

Note that **view\_sm** is rather new and has not been extensively tested.

## listing rois

Trying to remember what analyses you have done is a constant problem when dealing with large amounts of data. To help alleviate this problem, and to eliminate unnecessary digging through directories, I wrote a program called **list\_roi**. Given the name of your ROI analysis, your session ID file, and your session directory file, **list\_roi** will list all of the ROIs that have been created. For more information, and for an example, see the **list\_roi** manpage (via `list_roi -man`) or the manpages in the Appendix.

## creating and viewing a spatial activity profile

In displaying the results of fMRI analyses, most people show differences between conditions, commonly in a statistical map. However, this representation is twice-removed from the data: first there is the calculation of the difference between two conditions; second is the determination of whether or not that difference is significantly different from zero, and the subsequent transformation into a statistical value. Wouldn't it be nice, on the other hand, to look at your data directly, without these intermediate steps?

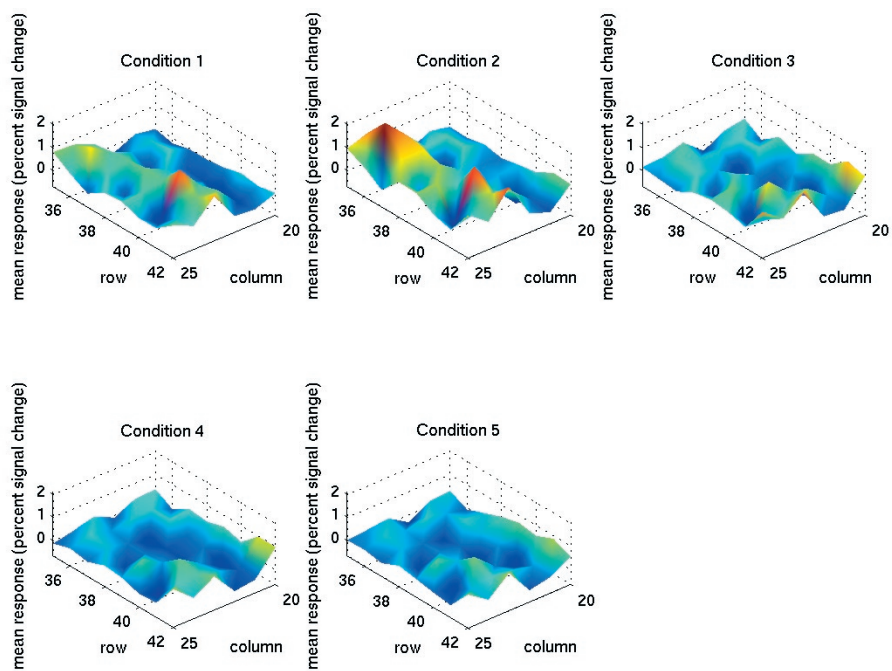
I've developed a novel but intuitive way to look at your data, tentatively called the Spatial Activity Profile (SAP). A SAP is a surface where the height at a point is directly related to the magnitude of the signal at the point. When plotted for all conditions in an experiment and on the same scale, the SAP allows you to quickly compare all conditions at once. As well, it's now possible to see where there are voxels that respond similarly to all conditions, suggesting that such voxels might be venous rather than gray matter.

**view\_mesh** implements the display of SAPs. To narrow down the region to display, it is customary to use an ROI to mask a specific portion of the brain. Note that the ROI must be rectangular, as it is presently impossible to display a SAP in matlab with non-rectangular ROIs. Thus it would be good to take a rectangular ROI around a specific functional region.

**view\_mesh** requires the following arguments:

1. **-roianalysis name**  
The name of your ROI analysis.
2. **-roi name**  
The name of your rectangular ROI.
3. **-sf sessid -df sessdir**  
The name of your session ID and session directory files, respectively.

For an example of a SAP, see Figure 4.1. The SAPs are labeled by condition; the condition number refers to the condition number used in your paradigm file. SAPs are displayed in the same scale for all conditions and plots, allowing qualitative visual comparisons.



*Figure 4.1. An example of a Spatial Activity Profile with five conditions in a region encompassing the Fusiform Face Area. Each point represents a voxel with in-plane resolution of 1.4mm. The conditions are, respectively, faces, scenes, objects, body parts, and scrambled objects.*



# appendix 1

## logfiles

Invariably, things will go wrong with your analyses and it will be your job to figure out what happened, why it happened, and what to do to fix it. To help in this task, every program in froi writes a copious amount of data to logfiles. The logfiles are saved under the directory “log” in the same directory from which you run the scripts. Naming follows a predictable pattern:

```
program_name-ROIname-subjectName.log
```

The content of the logfiles is identical to what you would see on-screen if you were to run the programs with the “-debug” parameter. Each line is given a date and time, hopefully allowing you to track down what happened when. As well, the logfiles also store the exact command-line used, thereby providing a record of what commands were run and perhaps helping to diagnose strange results.

In a future release I will add the ability for the logfiles to record matlab output, thus hopefully making the debugging process even easier.





# appendix 2

## roi format

The format of the ROI files is quite simple: text-based, space-delimited with four columns. Each row of the ROI file corresponds to a particular voxel. The columns refer to:

- 1 - slice number of the voxel
- 2 - row coordinate of the voxel
- 3 - column coordinate of the voxel
- 4 - value at the voxel

Note that columns two and three refer to row and column values, not x and y values. Not keeping this distinction in mind will cause headaches down the road; trust me, I've had them.

The fourth column, value, is normally 1, the exception being overlap ROIs, where the values denote the type of overlap at that voxel. I also reserve the right to use those values for other purposes in the future.

Planned later versions of the ROI format will require that the first line begin

with the pound character, after which will describe the dimensions of the functional volume.

# appendix x

## mannual pages for scripts

### **combine\_label**

combine\_label -- takes two labels (three to come in the future) and combines them either by set intersection or union.

#### **synopsis**

```
combine_label -subjectname subjectname -inputlabel label1,label2 -outputlabel  
output_label -method <intersection|union> [-labelpath $SUBJECTS_DIR/  
label -force -help -man -debug]
```

#### **description**

Much like with ROIs, it is sometimes useful to combine a two labels into one; for example, you might have labels for an anterior and posterior region and want to combine them into one label (label union). As well, you may have two vertices that you think overlap and you want to see if they do and to create a label that contains the overlapping vertices (label intersection). For example, this would be the case if you had an anatomical label (the fusiform gyrus) and you have a functional label (FFA) and want to know what vertices from the functional label lie within the anatomical label. `combine_label` allows the user to do this.

The default search path for labels is under `/subjectdir/funcdir/roidir/label/`. If you don't want this, provide the `-labelpath` command-line option. Note that if the directory does not exist it will be created.

The future goal is to have `combine_label` do the full suite of intersection, union, and overlap of 2-3 labels. At the moment, however, you can only do label intersection with two labels. Future work will implement the previous goal.

## required arguments

**-subjectname subjectname**

Name of the subject to which these labels belong. Even if using your own label directory (i.e., not the label directory under `$SUBJECTS_DIR`), you need to give this parameter.

**-inputlabel label1, label2**

Name of the input labels, with or without the `.label` extension. Separate the labels with a comma only, no spaces between the comma and the next label.

**-outputlabel outputlabel**

Name of the output label, with or without the `.label` extension. The label will be saved under `$SUBJECTS_DIR/subjectname/label/` unless overridden by the `-labelpath` option.

**-method intersect**

Method of label combination. `intersect` does simple set intersection and is the only method currently supported.

## optional arguments

**-labelpath \$SUBJECTS\_DIR/subjectname/label**

Path in which to look for labels, `-labelpath` defaults to the label directory under the subjectname. Change this if your labels are stored in a different location. Note that input labels will be read from this directory and the output label will be saved to this directory.

**-force**

If you attempt to run `combine_label` for an output label that has previously been created, the program will quit with an error, telling you of this situation. Use this parameter to override that behavior.

**-help**

Terse usage information.

**-man**

Complete usage information (this file).

**-debug**

Displays debugging information. Use this option to help debug unexpected behavior and when submitting a bug report.

## limitations

As described above, `combine_label` only works currently for the method “intersection” and for two labels.

## bugs

None known at the moment.

## revision

\$Id: combine\_label,v 1.3 2003/08/27 20:44:18 nknouf Exp \$

# combine\_roi

```
combine_roi -roianalysis roiAnalysisName -inputroi [ROI1,ROI2,ROI3, ...] -  
outputroi roiName -method <intersection | union | overlap> -sf sessid -df sessdir  
[-maskdir masks -funcDir bold -force -help -man]
```

Type “combine\_roi -man” for complete documentation

## description

combine\_roi allows one to combine multiple ROIs together quickly and easily. If, for example, you have ROIs for left and right FFA, and have computed ROI averages for each of those regions, but now want to consider the FFA together, irrespective of hemisphere, you can use combine\_roi with the “union” method to create a new ROI. As well, if you have an anatomical ROI and a functional ROI, you can use combine\_roi with the “intersection” method to get a new ROI that is the intersection of those two ROIs. Thus, “intersection” is like “AND” and “union” is like “OR”.

As well, combine\_roi can also be used to create an “overlap” ROI. This ROI is specially coded to show how up to three ROIs overlap. This is extremely useful if, for example, you have functionally defined three areas with different tasks and want to know if the areas overlap. To view the results, you can use the program display\_overlap .

Note that when giving the list of input ROIs to combine, the names must be separated by commas, with no space in between the comma and the next ROI. See the example below for more details.

ROIs created by combine\_roi are viewable and editable using edit\_roi .

## arguments

### required arguments

**-roianalysis roiAnalysisName**

Name of the analysis from which you computed the ROI. Do not use in conjunction with -maskdir .

**-inputROI roiList**

List of ROI names, delimited by commas, e.g., ROI1,ROI2,ROI3. Note that you cannot have a space inbetween the “,” and the next ROI in the list.

**-outputROI roiName**

Name of the ROI in which to put the output.

**-method <intersection | union | overlap >**

Name of the method to use when combining the ROIs. “intersection” will only consider a voxel to be part of the output ROI if it was in all of the input ROIs, while “union” will consider a voxel to be part of the output ROI if it was in any one (or more) of the input ROIs. Thus, “intersection” is like AND while “union” is like OR.

“overlap” is different than the other two. If you use “-method overlap”, combineROI will take the input ROIs (to a maximum of three) and compute the areas that are distinct and overlap. The order of your ROIs on the command line determines the colors that are used to determine the distinct and overlapping areas. You can display the resulting map using displayOverlap . The colors are as follows:

2 ROIs: blue - condition 1 | red - condition 2 | white - overlap of both conditions

3 ROIs: blue - condition 1 | red - condition 2 | yellow - condition 3 | purple - condition 1 and condition 2 | orange - condition 2 and condition 3 | green - condition 1 and condition 3 | white - overlap of all conditons

**-sf sessidfile**

Name of the session ID file. Note that only one session ID in the file will be used (the last one)

**-df sessdirfile**

Name of the session directory file.

## optional arguments

**-maskdir masks**

Name of the directory under the bold directory funcDir to hold your ROIs. Defaults to analysisName\_ROI , so if your analysis is called hfso, the mask directory would be called hfso\_ROI and made underneath the bold directory.

**-funcdir bold**

Name of the directory holding the functional runs. Usually “bold”, defaults to “bold”.

**-force**

If an roi named roiName already exists, combineROI will quit. Use this option to override this behavior, but be warned that it will overwrite the existing ROI.

**-help**

Provides complete argument information.

**-man**

Provides verbose usage information.

**-debug**

Provides complete debugging information.

## bugs

None known of. Send bug reports to < froi-bugs@sourceforge.net >.

## revision

\$Id: combine\_roi,v 1.8 2003/08/27 20:44:18 nknouf Exp \$

# compute\_roi

compute\_roi -analysis analysisName -roianalysis roiAnalysisName -roi roiName -sf sessidfile -df sessdirfile [-funcstem fmc -funcdir bold -outputstem roiName -maskdir analysisName\_ROI -force -help -man]

Type “compute\_roi -man” for complete documentation

## description



`compute_roi` takes a previously created ROI (commonly from `make_roi` and/or `edit_roi` ) and, with a FS\_FAST analysis, computes the average time-course over the ROI for each run in the analysis. This time course is saved as a Matlab structure in the ROI directory. Text file output (for importing into other programs) can be done using the `-text` argument.

To view the results, use `view_roi_results` .

`compute_roi` warns the user if there already exist computed and saved time-courses for this ROI; using the `-force` option will allow the user to overwrite those time courses.

NOTE: You must have the correct number of conditions for analysis `analysisName` , otherwise `compute_roi` will not work. The `fsfast` processing stream will work without the correct number of conditions being specified, but `compute_roi` will not, as it uses that value for some of its processing.

## arguments

### required arguments

**`-analysis analysisName`**

Name of a FS-FAST analysis

**`-roianalysis roiAnalysisName`**

Name of the analysis from which you created the ROI. Do not use in conjunction with `-maskdir` .

**`-roi roiName`**

Name of the ROI to use in the computations.

**`-sf sessidfile`**

Name of the session ID file. Note that only one session ID in the file will be used (the last one)

**`-df sessdirfile`**

Name of the session directory file.

## optional arguments

### **-funcdir bold**

Name of the directory holding the functional runs. Usually “bold”, defaults to “bold”.

### **-funcstem fmc**

Name of the functional stem from which to create the base. Defaults to fmc.

### **-outputstem roiName**

Name of the output stem of the computed ROI data. Defaults to the name of the ROI.

### **-maskdir masks**

Name of the directory where the ROI is saved. Only use if you saved data to a non-standard location when running makeROI . The default is to use the -roianalysis option to get the directory roiAnalysis\_ROI . Do not use in conjunction with -roianalysis .

### **-text**

If you give the -text paramter, compute\_roi will output text files containing the computed time courses and estimated ROI responses in addition to Matlab structure. This is useful for direct importing into other programs, such as Excel or SPSS. Defaults to text files not being saved.

### **-force**

If ROI time courses with names of the type analysis\_roi\_type.txt already exists (for example, if you have previously ran compute\_roi on these data), compute\_roi will quit. Use this option to override this behavior, but be warned that it will overwrite the existing time courses.

### **-help**

Provides complete argument information.

### **-man**

Provides verbose usage information.

### **-debug**

Provides complete debugging information.

## details of results

The Matlab structure will be called “ROIs” and will have a number of fields, some with information about the analysis and ROI computation, some with the results of `compute_roi`. The contents of the resulting structure (and thus, the results of the computation) depend on the analysis. For analyses with the `gammafit` parameter, you can expect the following:

Blocked

`ROIs.ROImean`: mean time course for each run. Note that this is the raw time-course before detrending (because FS-FAST does not save detrended data currently).

`ROIs.ROIhavg`, `ROIs.ROIhstd`, `ROIs.ROIhstderr`: Mean, standard deviation, and standard error, respectively, of the estimated deviation from baseline for each condition within the ROI.

`ROIs.ROIpct`, `ROIs.pctstd`, `ROIs.pctstderr`: Percent signal change, standard deviation, and standard error, respectively, for each condition.

Event-related

`ROIs.ROIhavg`, `ROIs.ROIhstd`, `ROIs.ROIhstderr`: Mean, standard deviation, and standard error, respectively, of the estimated deviation from baseline for each condition within the ROI.

`ROIs.ROIpct`, `ROIs.ROIpctstd`, `ROIs.ROIpctstderr`: Percent signal change, standard deviation, and standard error, respectively, for each condition.

For FIR analyses, the computations create the same data for both blocked and event-related analyses:

Blocked & Event-related

`ROIs.ROImean`, `ROIs.ROIstd`, `ROIs.ROIstderr`: Mean, standard deviation, and standard error, respectively, of the estimated hemodynamic response function for each condition.

`ROIs.ROIpct`, `ROIs.ROIpctstd`, `ROIs.ROIpctstderr`: Percent signal change, standard deviation, and standard error, respectively, of the estimated hemodynamic response function for each condition.

Note that for all of these results (with the exception of the average time-courses for the runs), fixation is the first condition.

A note on the directory structure: `compute_roi` expects either the `-roianalysis` or the `-maskdir` command line options. The first is used if using the default storage location for the ROIs (i.e., under the functional directory). If you give the option as `-roianalysis hfso`, `compute_roi` will expect to find the roi `roiName` under `bold/hfso_ROI`. Use the option `-maskdir` when your ROI is not saved

in this format.

## text output

The contents of the saved text files are as follows (stem: contents): mean: mean time course; std: standard deviation of “mean”; stderr: standard error of “mean”; pct: percent signal change; pctstd: standard deviation of “pct”; pctstderr: standard error of “pct”; havg: estimated max responses (only for blocked); hstd: estimated standard deviation (only for blocked); hstderr: estimated standard error (only for blocked).

## bugs

None known of. Send bug reports to < [froi-bugs@sourceforge.net](mailto:froi-bugs@sourceforge.net) >.

## revision

\$Id: compute\_roi,v 1.13 2003/08/27 20:44:18 nknouf Exp \$

# display\_overlap

`display_overlap -roianalysis analysisName -roi roiName -sf sessid -df sessdir`  
[optional arguments]

Type “`display_overlap -man`” for complete documentation

## description

`display_overlap` displays an overlap map of two or three ROIs. Such a map is created when you run `combine_roi` with the argument `-method overlap`. These overlap maps allow you to see exactly what ROIs overlap, where they overlap, and what areas are distinct for each ROI. The Matlab script provides a way to see the name of the ROIs used to create the overlap map as well as a legend of the colors. To find out more information, press `hto` get a help window.

# arguments

## required arguments

**-roianalysis analysisName**

Name of a FSEFAST analysis

**-roi roiName**

Name of the overlap ROI

**-sf sessidfile**

Name of the session ID file. Note that only one session ID in the file will be used (the last one)

**-df sessdirfile**

Name of the session directory file.

## optional arguments

**-roidir masks**

Name of the directory under the analysis roiAnalysisName to hold your ROIs. Defaults to roiAnalysisName\_ROI (e.g., for an analysis called “hfso”, the ROI directory defaults to “hfso\_ROI”).

**-base runDirectory**

Use this functional directory as the base upon which to display the overlap map. Only needs to be taken into consideration if you used different number of slices or different slice prescriptions for different runs of your scan. Defaults to the first bold directory.

**-funcDir bold**

Name of the directory holding the functional runs. Usually “bold”, defaults to “bold”.

**-funcStem fmc**

Name of the functional stem from which to create the base. See -base . Defaults to fmc.

**-help**

Provides complete argument information.

**-man**

Provides verbose usage information.

**-debug**

Provides complete debugging information.

## example

```
display_overlap -roianalysis lohfo2 -roi FFAoverlap -sf sessid -df sessdir
```

## bugs

None known of. Send bug reports to < [froi-bugs@sourceforge.net](mailto:froi-bugs@sourceforge.net) >.

## revision

```
$Id: display_overlap,v 1.6 2003/08/27 20:44:18 nknouf Exp $
```

## edit\_roi

```
edit_roi -analysis analysisName -roi roiName -sf sessid -df sessdir [optional arguments]
```

Type “`edit_roi -man`” for complete documentation

## description

`edit_roi` allows one to edit a previously created ROI. Such an ROI is commonly created from a program like `make_roi`. Multiple ROIs can be saved, as the program will use the input ROI as a template from which to save further ROIs. The program thus allows one to take a large ROI, like a “p values smaller than  $10^{-4}$  map and divide it into smaller ROIs.

edit\_roi starts Matlab with your input ROI displayed over the first run in your bold directory or the first run in your runlist (if you have defined a runlist). At first all voxels in the input ROI will be selected and in red. To erase all the selected voxels and to start from a blank ROI, press “c”. You will see that all of the voxels are now yellow, indicating that no voxels are selected for the current ROI. The yellow voxels indicate the input ROI and are used as a template when creating further ROIs.

To ease creation of ROIs, you can use a large brush size to select voxels and then press “i” to take the intersection of your brush selection and the underlying template.

There is more complete help information within the program itself; press “h” to read it.

## arguments

### required arguments

**-analysis analysisName**

Name of a fsfast analysis

**-roi roiName**

Name of the ROI to edit

**-sf sessidfile**

Name of the session ID file. Note that only one session ID in the file will be used (the last one)

**-df sessdirfile**

Name of the session directory file.

### optional arguments

**-overlay overlayName**

Name of the ROI to overlay on top of the ROI named with the -roi parameter. Most often this is used to make further edits to, e.g., an “FFA” ROI on top of a “t greater than 4” ROI.

**-roidir masks**

Name of the directory under the analysis analysisName to hold your ROIs. Defaults to analysisName\_ROI (e.g., for an analysis called “hfso”, the ROI directory defaults to “hfso\_ROI”).

**-base runDirectory**

Name of a different base directory on which to overlay the template and ROIs. Only needs to be taken into account if you used different slice prescriptions within the same scan. Defaults to the first run in the bold directory or the first run in your runlist (if you have that defined).

**-funcDir bold**

Name of the directory holding the functional runs. Usually “bold”, defaults to “bold”.

**-funcStem fmc**

Name of the functional stem from which to create the base. See -base . Defaults to fmc or the functional stem in your analysis.info file.

**-help**

Provides complete command information.

**-man**

Provides verbose usage information.

**-debug**

Provides complete debugging information.

## example

```
edit_roi -analysis blocked -roi ffa -sf sessid -df sessdir
```

## bugs

None known of. Send bug reports to <froi-bugs@sourceforge.net >.

## revision

```
$Id: edit_roi,v 1.9 2003/08/25 23:31:55 nknouf Exp $
```



# label2roi

```
label2roi -roianalysis roi_analysis_name -label label_name -hemi <rh|lh> -sf  
sessid_file -df sessdir_file [-labeldir roianalysis_ROI/label -bolddir bold -mask-  
dir roianalysis_ROI -funcstem fmc -outputstem roi-hemi -force -help -man  
-debug]
```

## description

Surface-based ROI analyses in Freesurfer are performed using labels, which are simply a list of marked vertices. Using the tools in `tksurfer`, it is quite easy to draw labels that encompass various anatomical regions. With a program like `combine_label`, you can take two labels and create either the intersection and/or union of the labels. What if, however, you'd like to see where these labels land on the slices?

`label2roi` exists to transform a surface- or volume-based label into a slice-based ROI. It uses the Freesurfer program `mri_surf2vol` to do most of its work. Note that this assumes you have done the following:  
Reconstructed the subject's brain.

Registered the current session to the previous session anatomicals that were used to reconstruct the brain.

Created a `subjectname` file, as described in the FS-FAST documentation.

One must be careful when using this program; if, for some reason, your slice prescription does not include the region within the label, the program will fail with an error.

The label filename must be in a certain format. The label filename must be of the form: `label-hemi.label`, where "label" is the name given with the `-label` flag, and "hemi" is the hemisphere given with the `-hemi` flag. For example, if you have a right hemisphere fusiform label, your label file should be called "fusiform-rh.label", and your command line arguments would be "`-label fusiform -hemi rh`". `label2roi` will create an ROI named "label.roi", so in the previous example, it would create "fusiform.roi".

## arguments

### required arguments

**-roianalysis roi\_analysis\_name**

Name of the analysis under which you created the ROI and subsequently created labels. `label2roi` will search for labels under `roidir/label`.

**-label label\_name**

Stem name of the label.

**-hemi <rh|lh >**

Hemisphere from which to map the label. Note that if you try and map a label from a hemisphere to a slice prescription that does not contain the label, the program will fail with an error.

**-sf sessid\_file**

Name of the session ID file.

**-df sessdir\_file**

Name of the session directory file.

### optional arguments

**-projfrac 0**

Fraction  $[0, 1]$  of the cortical thickness at each vertex to project along the surface normal. The default is 0. Change this to project more of your data onto the cortical surface, as the default is to sample only at the grey-white boundary.

**-labeldir roi\_analysis\_ROI/label**

Directory from which to search for the input label. Defaults to a directory called “label/” underneath your ROI analysis directory.

**-bolddir bold**

Name of the functional directory. Defaults to “bold”.

**-maskdir roianalysis\_ROI**

Name of the directory that holds the resulting ROI. Defaults to `<roianalysis_`

ROI>; set this argument if you save your data in a non-standard location. Do not use in conjunction with -roianalysis .

**-funcstem fmc**

Name of the functional stem. Defaults to “fmc”. Change this if, for example, you have separate functional volumes for smoothing and want to create ROIs from that data.

**-outputstem roi-hemi**

Name of the stem for the output label. Defaults to roi-hemi , making the output file <roi-hemi.label>.

**-force**

If you attempt to run label2roi for a ROI that has previously been created, the program will quit with an error, telling you of this situation. Use this parameter to override that behavior.

**-help**

Terse usage information.

**-man**

Complete usage information (this file).

**-debug**

Displays debugging information. Use this option to help debug unexpected behavior and when submitting a bug report.

## bugs

None known of. Send bug reports to < froi-bugs@sourceforge.net >.

## revision

\$Id: label2roi,v 1.4 2003/08/25 23:30:15 nknouf Exp \$

# list\_roi

`list_roi -roianalysis roianalysis_name -sf sessid -df sessdir [optional arguments]`

## description

When performing many ROI analyses it's sometimes hard to remember what ROIs have been created. `list_roi` exists to simply list the ROIs in a given roi analysis directory.

## arguments

### required arguments

**-roianalysis roianalysis\_name**

Name of the analysis used to create the ROIs. Do not use in conjunction with `-maskdir`.

**-sf sessidfile**

Name of the session ID file. Note that only one session ID in the file will be used (the last one)

**-df sessdirfile**

Name of the session directory file.

### optional arguments

**-maskdir mask\_dir**

Use this when you have stored your ROIs in a non-standard location.

**-help**

Provides complete argument information.

**-man**

Provides verbose usage information (this document).

### **-debug**

Provides complete debugging information.

## **revision**

\$Id: list\_roi,v 1.3 2003/08/27 20:44:18 nknouf Exp \$

## **make\_label**

```
make_label -analysis analysis_name -contrast contrast_name -hemi <rh|lh>
-sf sessid_file -df sessdir_file [-map t -threshold 2 -type signed -bolddir bold
-maskdir roianalysis_ROI -funcstem fmc -outputstem roi-map-hemi -force -
help -man -debug]
```

## **description**

Sometimes it is useful to take a ROI and convert it into a surface-based label; for that, we have `roi2label`. Other times, however, we want to easily transform a previously computed contrast into a surface-based \*.w file as well as a label. To do that, we can use `make_label`. It is required to apply a threshold to the contrast data; otherwise many more vertices will be included in the label than the research would like. The default threshold is “2”, so for t-maps this means that t values greater than 2 will be included in the label, while for a sig map, sig values greater than 2 (and thus, smaller than  $10^{-2}$ ) will be included. `make_label` defaults to using signed quantities; to use unsigned, add the argument “-type unsigned”.

To use `make_label` you must have a reconstructed brain for the subject in question, a “subjectname” file under their session directory with the name of the reconstructed subject’s name inside, and a “register.dat” file under their bold directory. This thus assumes that you have run either `autoreg-sess` or `tkregister-sess` on this subject to create the “register.dat” file. Registering and reconstructing a brain takes a fair amount of time on the part of the researcher and thus should probably only be done with long-term or repeat subjects.

There are some provisions for storing data in non-standard locations. See the options below for more information.

## required arguments

**-analysis analysis\_name**

Name of the analysis for the contrast in question.

**-contrast contrast\_name**

Name of the contrast to transform into a label.

**-hemi <rh|lh >**

Hemisphere on which to map the ROI. Note that if you try and map an ROI from a hemisphere for which there are no voxels, the program will fail, often-times without a useful error message.

**-sf sessid\_file**

Name of the session ID file.

**-df sessdir\_file**

Name of the session directory file.

## optional arguments

**-map t**

Statistical map to transform to a label. Defaults to “t”.

**-threshold 2**

Threshold to use when transforming the contrast to a label. Defaults to 2.

**-type signed**

Whether to use signed or unsigned quantities when thresholding the contrast. Defaults to “signed”.

**-projfrac 0**

Fraction [0, 1] of the cortical thickness at each vertex to project along the surface normal. The default is 0. Change this to project more of your data onto the cortical surface, as the default is to sample only at the grey-white boundary.

**-bolddir bold**

Name of the functional directory. Defaults to “bold”.

**-maskdir roianalysis\_ROI**

Name of the directory that holds the ROIs. Defaults to <roianalysis\_ROI>; set this argument if you save your data in a non-standard location. Do not use in conjunction with -roianalysis .

**-funcstem fmc**

Name of the functional stem. Defaults to “fmc”. Change this if, for example, you have separate functional volumes for smoothing and want to create ROIs from that data.

**-outputstem roi-map-hemi**

Name of the stem for the output label. Defaults to roi-map-hemi , making the output file <roi-map-hemi.label>.

**-force**

If you attempt to run roi2label for a label that has previously been created, the program will quit with an error, telling you of this situation. Use this parameter to override that behavior.

**-help**

Terse usage information.

**-man**

Complete usage information (this file).

**-debug**

Displays debugging information. Use this option to help debug unexpected behavior and when submitting a bug report.

## bugs

The “subjectname” file must contain only one line: the name of the subject. If there are one or more empty lines after the subject’s name, the program will fail.

## revision

\$Id: make\_label,v 1.4 2003/08/27 20:44:18 nknouf Exp \$

# make\_roi

```
make_roi -analysis analysis_name -contrast contrast_name -roi roi_name -sf  
sessid -df sessdir [optional arguments]
```

Type “make\_roi -man” for complete documentation.

## description

The first step in an ROI analysis is to define the ROI. make\_roi allows one to do that by thresholding a statistical map. Thresholds can be applied to either the t or significance map. As well, you can use the false discovery rate (FDR) technique to select your significance value threshold. \*\* insert FDR reference\*\*

Most of the time, however, you don't want to simply include all voxels that pass a certain threshold in an ROI. Rather, you want the voxels that pass a threshold and are clustered together in a specific anatomical region. Thus, after the creation of the thresholded ROI, make\_roi allows you to edit this ROI (using the same tools as in edit\_roi ) to select and create ROIs based on your thresholded ROI. These are the ROIs that you will most likely use in further processing.

You can get help when editing the ROI by typing “h”. Voxels in your current ROI are in red, while voxels that are not in your current ROI but are in the thresholded ROI are in yellow.

There are many options available for make\_roi that influence later processing; please read the ARGUMENTS section for more details.

## arguments

### required arguments

**-analysis analysisName**

Name of a fsfast analysis

**-contrast contrastName**

Name of a fsfast contrast



**-roi roiName**

Name you wish to give this ROI

**-sf sessidfile**

Name of the session ID file. Note that only one session ID in the file will be used (the last one)

**-df sessdirfile**

Name of the session directory file.

## optional arguments

**-map t,sig**

Stat map for contrast contrastName to load. As of writing, the code has been tested only minimally for sig maps. Defaults to sig maps.

**-fdr value**

Instead of thresholding by a simple  $t$  pvalue, threshold using False Discovery Rate (FDR). **\*\*insert ref\*\*** The default is to choose the  $p$  value that makes assumptions about the correlations of the data (that they are independent or positively dependent); these assumptions are usually valid with fMRI data. Making these assumptions gives you a more liberal pvalue than when making no assumptions. If you would like to make no assumptions add the `-np` parameter, described below. At the moment, this only applies the FDR pvalue to sig maps.

**-np**

When given, make no assumptions about the independence or dependence of your data. Add this parameter in addition to `-fdr` (described previously) if you have questions about the independence of your data.

**-threshold value**

Threshold to use when creating ROI from stat map. For both  $t$  and sig(aka,  $F$ ) maps, all voxels with  $t$  (or  $F$ ) values greater than this value will be included. Defaults to 2. See `-map`

**-base runDirectory**

When running the matlab function `editOverlay`, use this functional directory as the base upon which to overlay the mask. Only needs to be taken into consideration if you used different number of slices or different slice prescriptions for different runs of your scan. Defaults to the first bold directory.

**-funcDir bold**

Name of the directory holding the functional runs. Usually “bold”, defaults to “bold”.

**-funcStem fmc**

Name of the functional stem from which to create the base. See -base . Defaults to fmc.

**-maskdir masks**

Name of the directory under the bold directory funcDir to hold your ROIs. Defaults to analysisName\_ROI , so if your analysis is called hfso, the mask directory would be called hfso\_ROI and made underneath the bold directory.

**-(no)signed**

Determines whether to take signed or unsigned values for the thresholding. If -signed , voxels will be selected that are above (below) the threshold for positive (negative) thresholds. Use -nosigned when you don't care about the sign of the stats (i.e., when you want to take all the voxels that satisfy  $\text{abs}(\text{statistics}) > \text{abs}(\text{threshold})$ ). Defaults to -signed . See also -unsigned .

**-unsigned**

Use unsigned values for thresholding. Do not use with -signed . See also -(no)signed .

**-force**

If an roi named roiName already exists, makeROI will quit. Use this option to override this behavior, but be warned that it will overwrite the existing ROI.

**-help**

Provides complete command information.

**-man**

Provides verbose usage information.

**-debug**

Provide complete debugging information.

## bugs

The code has not been extensively tested with using the -unsigned option in

conjunction with `-fdr` .

## revision

\$Id: make\_roi,v 1.13 2003/08/27 14:57:56 nknouf Exp \$

# roi2label

```
roi2label -roianalysis roi_analysis_name -roi roi_name -hemi <rh|lh> -sf ses-  
sid_file -df sessdir_file [-labeldir roianalysis_ROI/label -bolddir bold -maskdir  
roianalysis_ROI -funcstem fmc -outputstem roi-hemi -force -help -man -de-  
bug]
```

## description

Slice-based ROIs (which are made by `make_roi` and/or `edit_roi`) are useful when you want to look at ROIs within a particular subject within a particular session; they eliminate the distortions caused by transformation into another space, such as Talairach or spherical. However, there are times when you might want to look at ROIs over multiple sessions, or wants to compare ROIs across subjects in another space. To that end, `roi2label` exists to transform a slice-based ROI into a surface-based label.

To use `roi2label` you must have a reconstructed brain for the subject in question, a “subjectname” file under their session directory with the name of the reconstructed subject’s name inside, and a “register.dat” file under their bold directory. This thus assumes that you have run either `autoreg-sess` or `tkregister-sess` on this subject to create the `register.dat` file. Registering and reconstructing a brain takes a fair amount of time on the part of the researcher and thus should probably only be done with long-term or repeat subjects.

Labels are saved under the `roi` directory of the subject, within a directory called “label”. Note that if you then want to compare labels across different sessions or subjects (for example, with the `combine_label` program), you will need to move the label(s) into a common location.

`roi2label` uses the Freesurfer program “`mri_vol2surf`” for the mapping of slice-

based data to a surface-based representation.

Note that the program will fail with an error if there are no vertices in the resulting mapping. If this occurs, ensure that you are mapping to the correct hemisphere, as this is an easy place to make a mistake.

## arguments

### required arguments

**-roianalysis roi\_analysis\_name**

Name of the analysis under which you created the ROI.

**-roi roi\_name**

Name of the ROI.

**-hemi <rh|lh >**

Hemisphere on which to map the ROI. Note that if you try and map an ROI from a hemisphere for which there are no voxels, the program will fail with an error.

**-sf sessid\_file**

Name of the session ID file.

**-df sessdir\_file**

Name of the session directory file.

### optional arguments

**-projfrac 0**

Fraction [0, 1] of the cortical thickness at each vertex to project along the surface normal. The default is 0. Change this to project more of your data onto the cortical surface, as the default is to sample only at the grey-white boundary.

**-labeldir roi\_analysis\_ROI/label**

Directory where you want the output label to be stored. Defaults to a directory called “label/” underneath your ROI analysis directory.

**-bolddir bold**

Name of the functional directory. Defaults to “bold”.

**-maskdir roianalysis\_ROI**

Name of the directory that holds the ROIs. Defaults to <roianalysis\_ROI>; set this argument if you save your data in a non-standard location. Do not use in conjunction with -roianalysis .

**-funcstem fmc**

Name of the functional stem. Defaults to “fmc”. Change this if, for example, you have separate functional volumes for smoothing and want to create ROIs from that data.

**-outputstem roi-hemi**

Name of the stem for the output label. Defaults to roi-hemi , making the output file <roi-hemi.label>.

**-force**

If you attempt to run roi2label for a label that has previously been created, the program will quit with an error, telling you of this situation. Use this parameter to override that behavior.

**-help**

Terse usage information.

**-man**

Complete usage information (this file).

**-debug**

Displays debugging information. Use this option to help debug unexpected behavior and when submitting a bug report.

## bugs

None known at the moment. “mri\_vol2surf” sometimes gives a warning of the form “voxel size not the same”; I have not tested to see if this causes problems or not. Send bug reports to <froi-bugs@sourceforge.net >.

## revision

\$Id: roi2label,v 1.6 2003/08/27 20:44:18 nknouf Exp \$

# view\_mesh

Manual information to come soon.

# view\_roi\_results

`view_roi_results -analysis lohfs0 -roianalysis lohfs0 -roi ffa -sf sessid -df sessdir`  
[optional arguments]

Type “`view_roi_results -man`” for complete documentation

## description

Running `compute_roi` to compute an ROI average creates a number of data sets, viewing of which can be somewhat tedious. This is where `view_roi_results` comes in. This script, which starts a Matlab GUI, allows one to easily view the results of an ROI computation. Thus, for blocked runs, you will be able to view the raw time course over the ROI, the estimated deviation from baseline (and errors) and the estimated percent signal change (and errors); for event-related, you will be able to view the estimated time-course (and errors) in both raw scanner units and percent signal change.

Within the GUI, click on the help menu to get more information. If, at some time when running `view_roi_results`, you want to load another data structure, you can do this from the “File...Open...” menu option.

## arguments

### required arguments

`-analysis analysisName`

Name of a fsfast analysis

**-roianalysis roiAnalysisName**

Name of the analysis from which you computed the ROI. Do not use in conjunction with `-maskdir` .

**-roi roiName**

Name of the ROI whose results you want to view.

**-sf sessidfile**

Name of the session ID file. Note that only one session ID in the file will be used (the last one)

**-df sessdirfile**

Name of the session directory file.

## optional arguments

**-maskdir masks**

Name of the directory where the ROI is saved. Only use if you saved data to a non-standard location when running `make_roi` . The default is to use the `-roianalysis` option to get the directory `roiAnalysis_ROI` . Do not use in conjunction with `-roianalysis` .

**-help**

Provides complete argument information.

**-man**

Provides verbose usage information.

**-debug**

Provides complete debugging information.

## example

`view_roi_results -analysis lohfs0 -roianalysis lohfs0 -roi ffa -sf sessid -df sessdir.`

## bugs

None known; send an e-mail to < [froi-bugs@sourceforge.net](mailto:froi-bugs@sourceforge.net) > for bug reports.

## revision

\$Id: view\_roi\_results,v 1.3 2003/08/27 20:44:18 nknouf Exp \$

## view\_sm

view\_sm -analysis analysis\_name -preferred # -non-preferred # -sf sessid -df  
sessdir [-frac # -type # -funcdir -help -man -debug]

Type “view\_sm -man” for complete documentation

## description

Selectivity indices provide a means for determining the “selectivity” of particular voxels. In this case, selectivity means a measure of how much a voxel responds to one condition (denoted here as the “preferred” condition) versus another condition (denoted here as the “non-preferred” condition). While statistical maps provide a description of areas that are active versus another condition, selectivity maps (and the corresponding indices) provide a description of how well a voxel is tuned for a particular condition.

Consistent formulas for determining selectivity are not present in the literature; however, there are some that make more sense than others. Currently two formulas are implemented, with plans for user-defined indices in the future:

1.  $(\text{preferred} - \text{non-preferred}) / (\text{preferred} + \text{non-preferred})$
2.  $(\text{preferred} - \text{non-preferred}) / (\text{hemodynamic offset of the voxel})$

In order to provide meaningful values, instead of using percent signal change or other measures (which might be below zero), view\_sm uses the hemodynamic values calculated from FS-FAST. (Thus, as an aside, these maps can only be computed from analyses that were run with a gamma fit paramter). These values are then transformed into the rescaled values which are by default centered around 1000.

view\_sm will display a selectivity map from within Matlab. There are a number



of things that can be done with the selectivity map, such as view a certain slice, view a mosaic, and view the actual selectivity indices. To get more information, press “h” for a help window.

## arguments

### required arguments

**-analysis analysisName**

Name of a fsfast analysis

**-preferred condition number**

Condition number for the “preferred” condition, i.e., the condition you are interested in viewing selectivity.

**-non-preferred condition number**

Condition number for the “non-preferred” condition, i.e., the condition to which you want to compare the selectivity of the “preferred” condition.

**-sf sessidfile**

Name of the session ID file. Note that only one session ID in the file will be used (the last one)

**-df sessdirfile**

Name of the session directory file.

### optional arguments

**-fracmask fraction**

Specify a fraction of selectivity indices to show, masking all others. Note that taking a relatively high fraction, such as 0.8 or 0.9, will not clutter the map, as most selectivity indices will be clustered in the middle of the curve. The default is 0.9.

**-type selectivity index type**

Specify a type of selectivity index to compute. There are currently two indices implemented, with plans for user-defined indices in the future.

1.  $(\text{preferred} - \text{non-preferred}) / (\text{preferred} + \text{non-preferred})$

2. (preferred - non-preferred)/(hemodynamic offset of the voxel)

The default selectivity index formula is 1.

**-roimask roi\_name**

Name of an ROI to use to mask the selectivity indices. If used, only those selectivity indices that are also in your ROI will be displayed. Note that your ROI directory must follow the normal conventions and must be named with the analysis given. As well, using the options `-frac`, `-min`, and `-max` will have no effect when using `-roimask`.

**-exclude**

Exclude all selectivity indices that fall outside of two standard deviations from the mean. This is only used when using an ROI to mask your selectivity indices. Be wary of using this, as it throws out data.

**-base runDirectory**

The name of a different base directory on which to overlay the selectivity map, this defaults to the first run in the bold directory or the first run in your runlist (if you have a runlist defined).

**-funcdir bold**

The name of the directory holding the functional runs, it is usually called "bold" and defaults to "bold".

**-help**

Provides complete argument information.

**-man**

Provides verbose usage information (this document).

**-debug**

Provides useful debugging information.

## example

```
view_sm -analysis faces -preferred 1 -non-preferred 2 -frac 0.8 -sf sessid -df
sessdir
```

## bugs

None known. If you find any, please send e-mail to < froi-  
bugs@sourceforge.net>.

## **revision**

\$Id: view\_sm,v 1.5 2003/08/27 16:04:53 nknouf Exp \$



# index

## C

### combining labels

with `combine_label` 10, 27, 37, 38, 39, 40, 53, 65

### combining ROIs

with `combine_roi` 23, 24, 25, 40, 41, 43, 48, 50

### computing ROI results

with `compute_roi` 9, 16, 18, 19, 43, 44, 45, 46, 47, 53, 65, 68, 70, 74

### creating a ROI

with `make_roi` 12, 13, 14, 16, 18, 43, 47, 50, 53, 56, 58

## D

debugging. *See* `logfiles`

## E

### editing a ROI

with `edit_roi` 9, 13, 14, 16, 18, 41, 43, 47, 50, 52, 53, 56, 58, 62, 65, 68, 70

## F

`froi` 6

## I

installation 1

## L

### listing ROIs

with `list_roi` 29, 56, 58  
`logfiles` 33

## M

### mapping a label to a ROI

using `label2roi` 10, 27, 53, 54, 55, 56

### mapping a ROI to a label

with `roi2label` 10, 12, 26, 27, 40, 47, 56, 58, 60, 65, 67, 68

## N

naming of ROIs 13

## R

ROI analysis steps 9

ROI basics 6, 11

ROI data format 35

ROI naming. *See* naming of ROIs

### ROI overlap

creating using `combine_roi` 24, 25  
viewing with `display_overlap` 10, 24, 25, 40, 43, 48, 49, 50

## S

saving ROIs. *See* ROI data

**format****selectivity maps** 28creating and viewing 28, 29, 71, 73,  
74**spatial activity profiles** 30

creating and viewing 30, 68

**V****viewing ROI results**with view\_roi\_results 9, 18, 43, 47,  
68, 69, 70