

Journal of Statistical Software

October 2011, Volume 44, Issue 14. *http://www.jstatsoft.org/*

arf3DS4: An Integrated Framework for Localization and Connectivity Analysis of fMRI Data

Wouter D. Weeda
University of Amsterdam

Frank de Vos
University of Amsterdam

Lourens J. Waldorp
University of Amsterdam

Raoul P. P. P. Grasman
University of Amsterdam

Hilde M. Huizenga
University of Amsterdam

Abstract

In standard fMRI analysis all voxels are tested in a massive univariate approach, that is, each voxel is tested independently. This requires stringent corrections for multiple comparisons to control the number of false positive tests (i.e., marking voxels as active while they are actually not). As a result, fMRI analyses may suffer from low power to detect activation, especially in studies with high levels of noise in the data, for example developmental or single-subject studies. Activated region fitting (ARF) yields a solution by modeling fMRI data by multiple Gaussian shaped regions. ARF only requires a small number of parameters and therefore has increased power to detect activation. If required, the estimated regions can be directly used as regions of interest in a functional connectivity analysis. ARF is implemented in the R package **arf3DS4**. In this paper ARF and its implementation are described and illustrated with an example.

Keywords: fMRI, spatial model, functional connectivity.

1. Introduction

The main focus of functional magnetic resonance imaging (fMRI) analyses has been the localization of brain regions that are active during a particular task. This analysis is usually performed using *massive univariate testing* of voxels: testing each voxel separately for activation. A problem with this approach is that it requires ad-hoc multiple comparison corrections to control false positives (i.e., marking voxels as active while they are actually not), and that these corrections are often conservative (Nichols and Hayasaka 2003), or require stringent assumptions about the data (Worsley *et al.* 1996).

Conservativeness is especially problematic in cases where data are very noisy. For example, in developmental studies children often show increased levels of variability in brain responses (Samanez-Larkin and D'Esposito 2008), leading to smaller chances of detecting activation. But conservativeness can even be problematic in 'standard' studies when there are small differences in activation levels between conditions, or when there are few replications (sessions or blocks).

In recent years the focus is not only on localization, but also on how brain regions interact during task (and non-task) performance, so-called connectivity analysis (Biswal *et al.* 1995; Friston *et al.* 1997; Bassett and Bullmore 2006; Honey *et al.* 2009; Sporns 2010, 2011; Smith *et al.* 2011). For localization, and especially for connectivity analysis, it is important that all active brain regions (i.e., regions of interest, ROIs) are identified (Eichler 2005; Waldorp *et al.* 2011). Here the problem of conservativeness is especially problematic as failing to localize ROIs can bias the connectivity analysis.

To address these problems we developed a framework termed activated region fitting (ARF, Weeda *et al.* 2009, 2011). The ARF framework is based on the observation that fMRI activity measured in the brain has a spatial extent of several millimeters and that this activity is spatially smooth (Hartvig 2002). In standard univariate analysis this observation is neglected as each voxel is treated as independent from any other. Inherently, this leads to loss of information and results in less power to detect activated voxels. Incorporating the spatial smoothness between voxels (i.e., voxels close to each other in space behave similarly) can therefore be advantageous, see for example Bowman (2005); Penny and Friston (2003); Lukic *et al.* (2007); Xu *et al.* (2009). In ARF this is accomplished by assuming that every active brain region can be described by a Gaussian shaped model. An entire fMRI volume can then be described by multiple 'parameterized' Gaussian regions of activation. The Gaussian shape was chosen because: (i) it has a plausible shape; a highly active center with activity diminishing farther away from the center, (ii) has a relatively simple parameterization, and (iii) is flexible enough to model regions of different shapes and sizes. This parameterization allows for hypotheses on the location of an active region, the spatial extent of an active region, and the amplitude of an active region.

FMRI analysis using ARF has several advantages. The main one is that ARF has increased power to detect activation as the number of parameters in the spatial model is low compared to the number of voxels it describes. In Weeda *et al.* (2009) and Weeda *et al.* (2011) it was shown in simulations that ARF has increased power over standard methods (Bonferroni, false discovery rate, Genovese *et al.* 2002, and a cluster-size threshold, Forman *et al.* 1995). Also, when applied to empirical data of a verbal feedback experiment (Christoffels *et al.* 2007) and a go/no-go experiment (van Gaal *et al.* 2010) an increase in power over standard methods was observed.

Another advantage of the ARF method is that the spatial model can be directly used as ROIs in a connectivity analysis (Weeda *et al.* 2011). This allows researchers to identify which brain regions covary, without having to specify ROIs a-priori. Traditionally ROIs are either defined a-priori based on anatomical regions, or based on the thresholded outcome of a general linear model (GLM) analysis. When ROIs are based on anatomical regions this requires that the constituents of the network are already known, which is often not the case. When ROIs are based on GLM outcomes connectivity analysis can be hindered by conservativeness of the multiple comparison correction procedure, therefore missing regions in the network. Since ARF has increased power (Weeda *et al.* 2009, 2011), it gives a better estimation of the regions

in the network. Also, parameters in the spatial model allow for direct selection of the voxels within a ROI: assigning higher weight to voxels in the center of a region. This alleviates the need for post-hoc procedures to calculate ROI summary statistics. ARF thus integrates the localization of brain regions and the analysis of the functional connections between these regions in one framework.

The ARF framework is implemented in R (R Development Core Team 2011) in the package **arf3DS4** (Weeda 2011). The overview of this paper is as follows. First, the ARF method is described in more detail. Second, the **arf3DS4** package will be explained. Third, the application of ARF will be illustrated with an example dataset.

2. Activated region fitting

The method of activated region fitting was developed to increase the power of detecting active brain regions by capitalizing on the smooth spatial pattern inherently present in fMRI data. In ARF this smooth pattern is modeled by fitting a spatial model consisting of multiple Gaussian shaped regions to fMRI data. By fitting models with different complexity (i.e., different numbers of Gaussian shaped regions) ARF determines a model that best describes the data while keeping the number of parameters in the model low. Subsequently, hypothesis tests are performed on the parameters of the optimal model, allowing to test hypotheses on location, spatial extent and amplitude of each region. The active regions can also be used as ROIs in a connectivity analysis. By using each active region as a ROI, ARF uses the model parameters to estimate trial-by-trial activity. Correlating these trial-by-trial estimates of different regions gives the functional connectivity estimates (i.e., covariation of regions) between regions.

Since the details of the ARF methods are described fully in Weeda *et al.* (2009) and Weeda *et al.* (2011), we limit ourselves here to the most important functions of the method. We will describe in detail the data requirements, the spatial model, model selection, and connectivity estimation.

2.1. Data

An ARF analysis is performed on unthresholded outcomes of a standard GLM analysis (Friston *et al.* 1995). To maintain flexibility this analysis can be performed in any fMRI analysis package that supports export of fMRI data files in the NIfTI format, e.g., **FSL** (Smith *et al.* 2004), **SPM** (Friston *et al.* 2007), or R packages **AnalyzeFMRI** (Marchini and Lafaye de Micheaux 2011; Bordier *et al.* 2011) or **fmri** (Tabelow and Polzehl 2011a,b). This allows researchers to perform the basic fMRI analysis steps (e.g., registration, motion correction, temporal filtering) and the GLM modeling steps in their preferred analysis package (see for example Huettel *et al.* 2008; Jezzard *et al.* 2001; Lazar 2008, for an introduction to GLM modeling). For the ARF method it is advised to *not* use spatial filtering (i.e., to *not* smooth the data), as smoothing can impose an artificial Gaussian structure on the data, even in pure noise cases, and therefore will result in overfitting. Standard, the output of a GLM analysis is a 3 dimensional volume of β values with their associated standard errors (usually converted to t or z values). For localization of active regions ARF requires several independent outcomes of a GLM analysis. This can be different runs of an event-related design, or different blocks

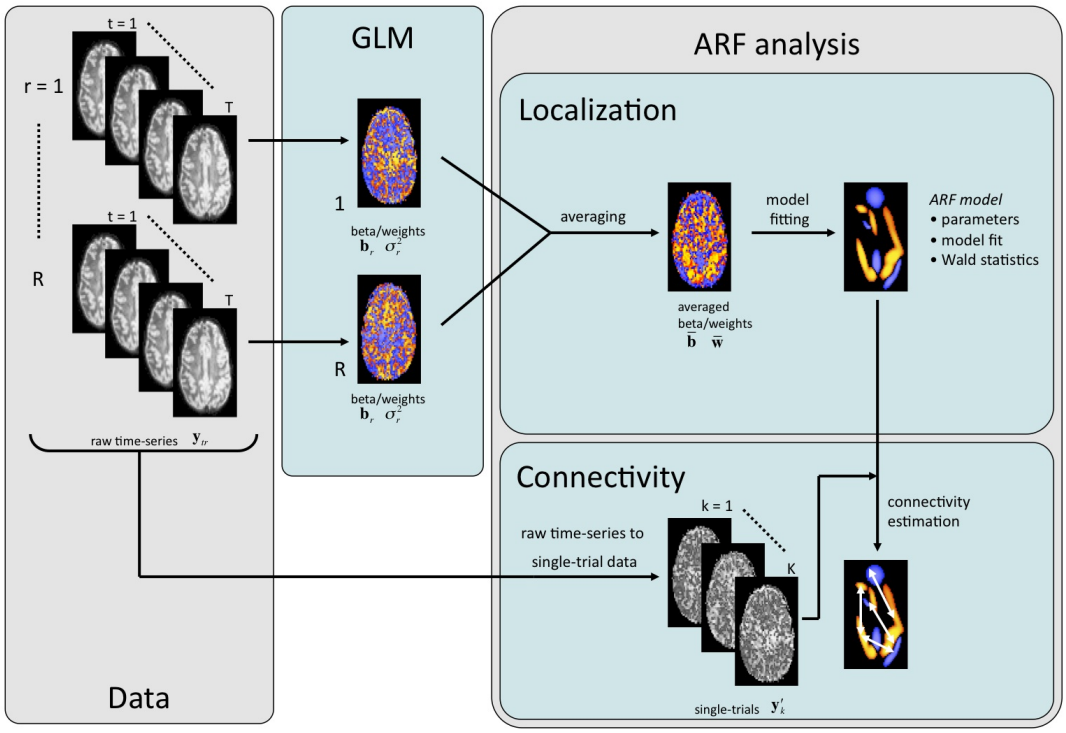


Figure 1: Overview of ARF analysis. First, data $\mathbf{y}$ (with $t = 1, ..., T$ timepoints) of multiple runs ($r = 1, ..., R$) are analyzed using standard GLM analysis. The unthresholded estimates ($\mathbf{b}$ and σ^2) are then averaged (creating $\overline{\mathbf{b}}$ and $\overline{\mathbf{w}}$, respectively) and used to estimate the ARF model. The model estimates are used to test hypothesis regarding location, spatial extent and amplitude of regions. Functional connectivity estimates can be obtained by: (i) estimating single trials $\mathbf{y}'$ (with $k = 1, ..., K$ trials) from the raw time series and (ii) correlating trial-by-trial activity for each region in the model.

in a blocked design[1]. When estimating functional connectivity ARF requires in addition the time series data on which the GLM was performed.

Let $\mathbf{y}_{tr}$ be a $(N \times 1)$ vector of measurements of time-point $t = 1, ..., T$ of run $r = 1, ..., R$ with N indicating the number of voxels in a volume. On these data (Figure 1, left panel) a GLM analysis is performed. Let the outcomes of this GLM analysis (Figure 1, middle panel) for each run be the $(N \times 1)$ vectors $\mathbf{b}_r$ (β values) and σ_r^2 (squared standard errors of β)[2]. The fitting of the ARF spatial model is performed on the averaged data (Figure 1, top-right panel) over the runs:

$$\overline{\mathbf{b}} = \frac{1}{R} \sum_{r=1}^{R} \mathbf{b}_r \tag{1}$$

[1]For purposes of conciseness we will use the term 'run' for an independent measurement (be it a run or a block).

[2]Note that the outcomes can also be input as t values with the squared standard errors set to 1.

with associated average variances:

$$\overline{\mathbf{w}} = \frac{1}{R^2} \sum_{r=1}^{R} \sigma_r^2 \tag{2}$$

For this the runs must be registered within each subject. In the ARF analysis the GLM output $\mathbf{b}$ and σ^2, and the averaged data $\overline{\mathbf{b}}$ and $\overline{\mathbf{w}}$ are used. To perform connectivity analysis with ARF, in addition the raw time series data $\mathbf{y}$ are required (see Section 2.7).

2.2. Spatial model

The spatial model fitted to the averaged data consists of the sum of multiple Gaussian shapes each with its own set of parameter values. The Gaussian shape is defined by parameters for location, spatial extent (i.e., shape), and amplitude. Multiple Gaussian shapes are used to describe all active regions in an fMRI volume. The spatial model for voxel n and region $j = 1, ..., J$, with J indicating the number of regions in the model, is defined as:

$$f(\mathbf{x}_n, \theta) = \sum_{j=1}^{J} \frac{\theta_{10j}}{(2\pi)^{3/2}|\Sigma_j|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}_n - \mathbf{k}_j)^\top \Sigma_j^{-1}(\mathbf{x}_n - \mathbf{k}_j)\right] \tag{3}$$

In Equation 3 $\mathbf{x}_n$ is a vector containing the x, y, and z coordinates for voxel n, $\mathbf{k}_j$ is a vector containing the center locations of a region (θ_{1j}, θ_{2j}, and θ_{3j} respectively). The shape of region j is defined by matrix Σ_j, defining the main axes and rotation of the elliptical shape of the Gaussian:

$$\Sigma_j = \begin{bmatrix} \theta_{4j}^2 & \theta_{4j}\theta_{5j}\theta_{7j} & \theta_{4j}\theta_{6j}\theta_{8j} \\ \theta_{4j}\theta_{5j}\theta_{7j} & \theta_{5j}^2 & \theta_{5j}\theta_{6j}\theta_{9j} \\ \theta_{4j}\theta_{6j}\theta_{8j} & \theta_{5j}\theta_{6j}\theta_{9j} & \theta_{6j}^2 \end{bmatrix}$$

Where $|\Sigma_j|$ denotes the determinant of Σ_j. In Figure 2 an example of the Gaussian model for one region, overlaid to a structural brain image, is shown.

2.3. Parameter estimation

The parameters of the model are estimated by minimizing the weighted least squares (WLS) function:

$$S(\theta) = [\overline{\mathbf{b}} - f(X, \theta)]^\top W^{-1} [\overline{\mathbf{b}} - f(X, \theta)] \tag{4}$$

where $f(X, \theta)$ are the model estimates for each voxel in X, and W is a $(N \times N)$ diagonal matrix with the averaged variances $\overline{\mathbf{w}}$ on the diagonal. Since f is non-linear in θ the WLS function is minimized using a Newton-type algorithm using the R function `optim` (`"L-BFGS-B"` constrained optimization). To avoid invalid model predictions constraints are imposed on the rotation parameters (θ_7, θ_8, and θ_9) so they can only vary between -0.9 and 0.9. Furthermore constraints are imposed on the location and width parameters so that they cannot exceed the dimensions of the volume.

2.4. Model selection

The number of active regions (J) that best describe the fMRI data are determined by means of model selection, that is, to find a model that fits the data well, while not being overly complex. One metric that can incorporate model fit and complexity, and is especially useful

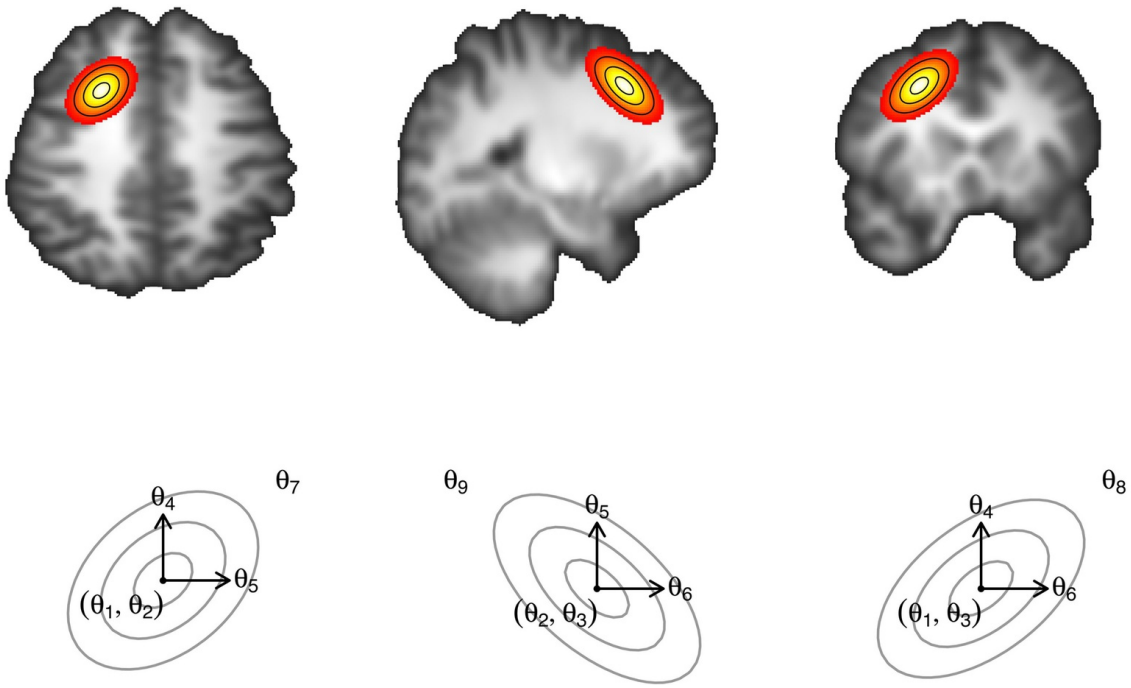


Figure 2: Overview of the Gaussian shape model for one region, overlaid to a structural brain image. The left panel shows the model in the $x-y$ plane, the middle panel in the $y-z$ plane, and the right panel in the $x-z$ plane. The region is centered at location $(x, y, z) = (\theta_1, \theta_2, \theta_3)$, with the width of the region defined by $(\theta_4, \theta_5, \theta_6)$. The rotation of the region is defined by $(\theta_7, \theta_8, \theta_9)$. Not shown is θ_{10} which defines the level of activation (i.e., amplitude) of the region. Note that the Gaussian is a continuous function, for purposes of clarity values outside the 95% isocontour of the region are not shown.

in neuroimaging applications, is the Bayesian Information Criterion (BIC, Schwarz 1978; Raftery 1999). The BIC penalizes for the number of parameters in the model, therefore keeping model complexity under control. Without constants the BIC equals:

$$BIC = \ln S(\hat{\theta}) + p \ln N \tag{5}$$

with p indicating the number of parameters in the spatial model. ARF fits models with increasing numbers of regions to the data and chooses the model with the lowest BIC value as the optimal model.

Currently the ARF method relies on the BIC for model selection to avoid complex models and therefore to increase power to detect activations. The BIC was chosen because when there is more than one model that is as close to the truth as possible, then the BIC will obtain the smallest one, whereas, for example, the AIC will not (Sin and White 1996). In Weeda *et al.* (2011) it was shown in simulations that BIC model selection yields adequate results. However, these simulations only included a limited number of regions and therefore future studies should address the question how well the BIC performs in more complex situations. Also, future work might incorporate other selection criteria (e.g., AIC, QIC) so that multiple criteria can be relied upon when choosing the optimal model.

2.5. Hypothesis testing

Once a model is chosen, hypothesis tests can be performed. For this the standard errors of the parameter estimates have to be calculated. Since the Gaussian shape is an approximation of the shape of the 'real' activation, the spatial model is misspecified. Also, this model may be misspecified in the number of regions that describe a dataset, as the number of regions in the model may not equal the actual number of regions. To take into account this misspecification (either in the number of regions or in the shape of the model), standard errors are derived using the sandwich estimator (Waldorp 2009; White 1980). For a full description of the sandwich procedure, see Weeda *et al.* (2009). Given this sandwich (co)variance matrix, hypothesis tests are performed using Wald statistics (see Weeda *et al.* 2009 and Weeda *et al.* 2011). The two most important hypotheses are (i) the amplitude of a region deviates from zero, and (ii) the spatial extent of a region deviates from zero. Optionally hypotheses on the location parameters can be tested to see whether they differ from a predefined location.

2.6. Procedure

The standard procedure for an ARF analysis is to fit models with different complexity (i.e., number of regions) to the data. Out of these models a BIC optimal model is chosen. Subsequently, hypotheses can be tested using the Wald statistics. For a model to be valid it must fulfill three conditions: (i) the minimization routine has converged and parameter estimates are not located on a bound, (ii) the model has the lowest BIC value (of a range of models), and (iii) all regions in the model have an amplitude and spatial extent greater than zero according to the Wald statistic. In general, the BIC optimal model has no or only a few regions that do not pass the Wald test. This makes sense as a region with very small amplitude or extent has little or no influence on the model fit, it will only 'cost' extra parameters.

2.7. Connectivity analysis

In addition to the localization of the active regions, a functional connectivity analysis, describing the (co)variation between the regions in the model, can be performed. In standard connectivity analyses ROIs have to be chosen, either a-priori (based on anatomical regions) or based on the outcomes of a GLM analysis. With ARF the activated regions in the model can be directly used as ROIs in a connectivity analysis (Weeda *et al.* 2011).

Functional connectivity analysis can be performed after model fitting. For this we need to estimate the single-trial data of the condition of interest. Note that we make a distinction between single trials and time points. The latter are the raw data at each time-point, while the former are estimates of brain activity of single trials and so need not be ordered temporally. To estimate single-trial activity the raw time series are regressed to a model with for every stimulus presentation (i.e., trial) a single regressor convolved with a hemodynamic response function (HRF, see Boynton *et al.* 1996), see Figure 3 for an example of this conversion. The outcome of this regression analysis thus leads to an estimate of brain activity of each trial. ARF uses these single-trial data to estimate connectivity[3] (see Weeda *et al.* 2011). For this, we first concatenate the raw time series data of all runs R to obtain one time series, from

[3]Double dipping (Kriegeskorte *et al.* 2009) is a fundamental issue in fMRI analysis. In the ARF method the locations of the regions are estimated from the mean activity map, whereas the connectivity is obtained from the (co)variances. The mean and (co)variance are statistically independent (see e.g., Muirhead 1982). Therefore, estimating functional connectivity using ARF is not affected by double dipping.

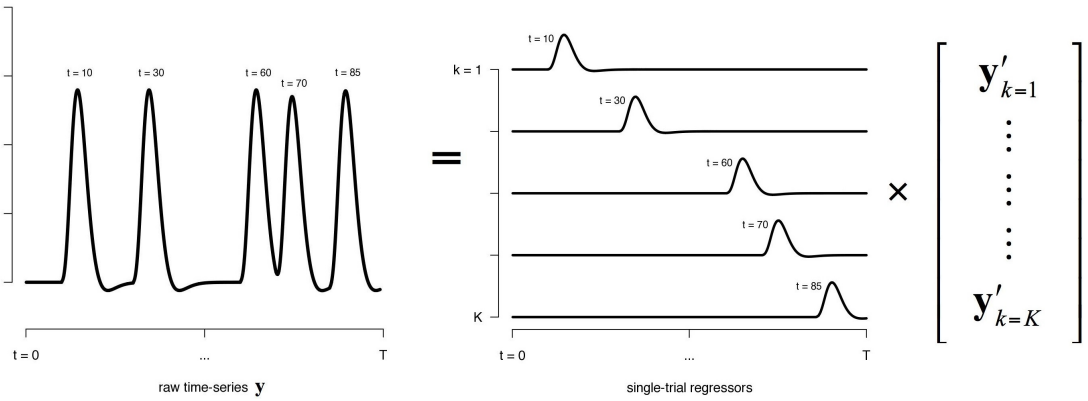


Figure 3: Raw time series to single-trial estimation. The raw time series ($\mathbf{y}$) are modeled with a single (convolved) regressor for each stimulus presentation (at $t = 10, 30, 60, 70, 85$ in this case). This results in estimates of activity ($\mathbf{y}'_k$) for each single trial $k = 1, ..., K$.

which we subsequently estimate the single-trial data. The outcomes of this analysis constitute the data connectivity is estimated on.

Let $\mathbf{y}'_k$ be the $(N \times 1)$ vector containing these estimated single-trial data for trial $k = 1, ..., K$. Let Z be a $(N \times J)$ matrix with in each column the model estimates ($f(X, \theta_j)$, with unit amplitude) for region $j = 1, ..., J$. For each trial k the single-trial data $\mathbf{y}'_k$ are regressed on the model:

$$\mathbf{y}'_k = Z\gamma_k + \epsilon_k \tag{6}$$

The vector of estimated trial-by-trial amplitudes $\hat{\gamma}_k$ is then estimated:

$$\hat{\gamma}_k = (Z^\top Z)^{-1} Z^\top \mathbf{y}'_k \tag{7}$$

We then construct a $(J \times K)$ matrix G with in each column the trial-by-trial amplitudes $\hat{\gamma}_k$. By correlating the rows of G we obtain a $(J \times J)$ matrix M containing the functional connectivity estimates between regions (Figure 1, lower-right panel).

3. The arf3DS4 package

The activated region fitting (ARF) method is implemented in the **arf3DS4**[4] (Weeda 2011) package for R (R Development Core Team 2011). This package was designed to work as seamless as possible with standard fMRI analysis packages (e.g., **FSL**, Smith *et al.* 2004), and therefore has a similar design: all data are stored in a predefined directory- and file-structure, and functions perform their operations on these files. This has the advantage that all data can be accessed easily, without putting too high demands on memory. It differs somewhat from standard R usage, where usually no predefined directory structure is required.

Most functions in the **arf3DS4** package save the objects they work on in predefined files in the directory structure while also returning them to the R environment. The returned objects

[4]The **arf3DS4** package can be obtained from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=arf3DS4`, load the package by typing `library("arf3DS4")`. The current version of **arf3DS4** requires R version 2.12.0 or higher.

thus act as a 'working copy' of the files in the experiment. This has the advantage that all information (fMRI data, ARF objects) is always in the same location within the directory structure, therefore facilitating analyses.

The **arf3DS4** package uses S4 class objects. S4 classes have the advantage that the slots in the classes are clearly defined and that classes can be used in a hierarchical way (e.g., inherited, extended, see Appendix B for an overview of working with S4 classes). In **arf3DS4** the S4 class objects are used in an 'object-oriented' way, meaning that ARF functions often take the same object as input and output. Most functions only modify (some of) the slots of the object and return it afterwards. This differs from standard R usage where often the object returned by a function is of a different type than the input object.

In the **arf3DS4** package three classes are most important: the `experiment` class, holding information on the directory structure, the `data` class, holding information on the data-files in an experiment, and the `model` class, holding information on the fitted models. In the next section these classes (together with some additional classes) will be explained in more detail. First, it is explained how to set up and load an 'experiment' (i.e., the directory- and file-structure). Thereafter, it is explained how to create and fit ARF models to data in the experiment and how to customize behavior of minimization procedure, (co)variance estimation, and Wald statistics calculation. Third, the procedure for finding an optimal model is described. Finally, it is explained how to perform a connectivity analysis.

3.1. Setting up an experiment

The first step in an ARF analysis is to set up the 'experiment' structure. This means defining the number and names of subjects in the analysis and the number and names of conditions. Once these are defined, the outcomes of the GLM analysis must be copied to the appropriate directories within the experiment structure. A last step is to perform the initial ARF data processing: checking the fMRI data and creating the averaged fMRI data files.

Creating experiment directories

An ARF experiment is defined by an object of class `experiment`. This holds the information on directory-locations, file-location, and number and names of subjects and conditions. To create the experiment directories and experiment object call:

```
makeExpDirs(path, name, subjectind, conditionind)
```

In this function `path` is the path where the experiment with name `name` will be created, `subjectind` is a character vector containing names of the subjects, and `conditionind` is a character vector with names of conditions. For example, to create an experiment named `"Word Decision"` on the desktop, with three subjects (`"PP001"`, `"PP002"`, and `"PP003"`) who each participated in two conditions (`"Different Strings"` and `"Equal Strings"`), type:

```
R> makeExpDirs("/Desktop/", "Word Decision", c("PP001", "PP002", "PP003"),
+    c("Different Strings", "Equal Strings"))
```

The directory structure of this experiment (located in `/Desktop/Word Decision/`) is shown in Figure 4. The actual experiment object is saved in the `experiment.Rda` file. This is also the file that will be read when loading the experiment. At this point the experiment still

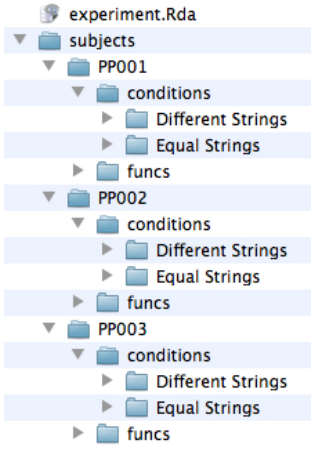


Figure 4: Directory structure for three subjects with two conditions.

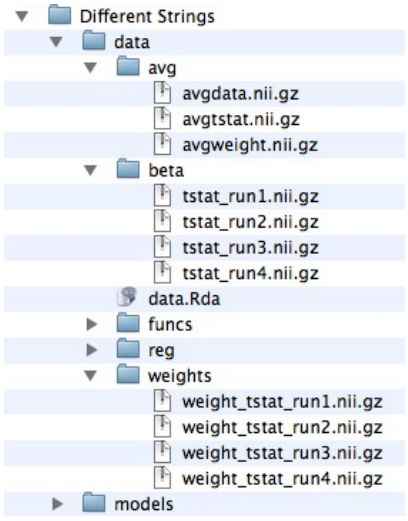


Figure 5: Directory structure of a single condition.

contains no fMRI data, only the directories are defined.

Copying fMRI data files

For each condition of each subject the data must be copied. FMRI data must be in NIfTI format (see Appendix A for an explanation of the NIfTI format and the `fmri.data` class). Within the directory of each condition there is a predefined directory structure to hold the fMRI data (see Figure 5, for the `"Different Strings"` condition of subject `"PP001"`). Within this `/data` directory the `/beta` and `/weights` directories hold the actual fMRI data ($\mathbf{b}_r$ and σ_r^2 respectively).

When the input data are t statistics only files containing the t statistics have to be copied to the `/beta` directory, the `/weights` can be left empty as the appropriate files will be created automatically. When the input data are β values, these have to be copied to the `/beta` directory while their standard errors have to be copied to the `/weights` directory. For example,

Slot	Description	type(length)
@n	Number of (brain)-voxels (all voxels that have a @mask value > 0).	numeric(1)
@mask	Vectorized mask for brain/non-brain voxels. Indicates for each voxel if it is used in the analysis (all values > 0 are used).	numeric(N)
@ss	Sums of squares of the data.	numeric(1)
@runs	Number of runs (R) in the condition.	numeric(1)
@betafiles	Vector with names of *beta*-files.	character(R)
@weightfiles	Vector with names of *weight*-files.	character(R)
@avgdatfile	Name of average *beta*-file.	character(1)
@avgWfile	Name of average *weight*-file.	character(1)
@avgtstatFile	Name of average t statistics file.	character(1)

Table 1: Overview of slots in the data class.

in Figure 5 there were 4 runs (of t statistics) in the "Different Strings" condition, so the only files copied were tstat_run1.nii.gz through tstat_run4.nii.gz, the other files were automatically created upon loading the experiment.

Loading the experiment

When all files are copied, the experiment must be updated with information on fMRI data location, filenames, runs, and if the data concern t statistics or β values. This information is stored within objects of the class data (see Table 1 for an overview of the data class). These objects are saved in the data.Rda files in each conditions' directory (see Figure 5). Note that data objects do not contain actual fMRI data. ARF can automatically gather information of fMRI data in an experiment and update all data objects. For this, the experiment must be loaded with the "set" argument. To do this for an experiment located in path experimentpath type:

```
R> loadExp(experimentpath, "set")
```

This will first gather information of fMRI data and create averages ($\overline{\mathbf{b}}$ in avgdata.nii.gz, and $\overline{\mathbf{w}}$ in avgweights.nii.gz, which will be saved in the /avg directory, see Figure 5). Subsequently, loadExp() will update the experiment (and the data objects) and load the former into memory. The next time the experiment is loaded, the "set" argument can be omitted. Only when adding/removing subjects or conditions, or when moving the entire experiment to a different location, the 'set' argument must be given the first time the experiment is loaded.

3.2. Creating and customizing a model

The core of the **arf3DS4** package is fitting models of increasing complexity and selecting the optimal model. The information on a model is stored in the model class. This class inherits an object of the data class and extends it with slots containing information on the model (see Table 2 for the slots in the model class).

Slot	Description	`type(length)`
`@convergence`	Convergence information.	`character(#)`
`@iterates`	Number of iterations of the minimization routine.	`numeric(1)`
`@minimum`	Value of objective function ($S(\hat{\theta})$) at the minimum.	`numeric(1)`
`@estimates`	Vector of parameter estimates, with $p = (10 \times J)$	`numeric(p)`
`@gradient`	Gradient at minimum.	`numeric(p)`
`@hessian`	Hessian matrix (2nd order partial derivatives).	`matrix(p,p)`
`@varcov`	(Co)variance matrix (using sandwich estimation).	`matrix(p,p)`
`@warnings`	Warnings returned by estimation procedures.	`character(#)`
`@fit`	Fit measures of the model (BIC (1) and RMSEA (2)).	`numeric(2)`
`@wald`	Wald statistics.	`wald class.`
`@regions`	Number of regions in spatial model.	`numeric(1)`
`@startval`	Starting values.	`numeric(p)`
`@proctime`	Processing time for minimization (1) and sandwich estimation (2), both in seconds.	`numeric(2)`
`@valid`	Is the model valid?	`logical(1)`

Table 2: Overview of slots in the `model` class.

Creating a model

A first step in performing the actual ARF analysis is creating a model. To create a model call:

```
R> mod <- newModel(modelname, regions, subject, condition)
```

This will create a new model object named `modelname` for subject `subject` and condition `condition` with `regions` region(s) in the spatial model, and passes it to `mod`. Simultaneously this will create a directory named `modelname` in the `/models` directory of the given condition (see Figure 6 for an example). In this directory are the `model.Rda` file, containing the ARF model object, the `options.Rda` file, containing the options of the ARF model, and the `start.Rda` file, containing starting values of the model. Also, there is a `/data` directory with the model estimates (`avgmodel.nii.gz`) and additional files used by the **arf3DS4** package.

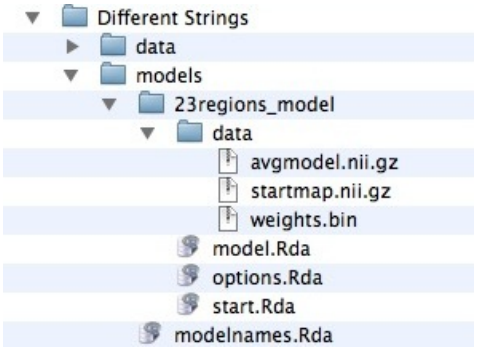


Figure 6: Directory structure of an ARF model.

Slot	Description	type(length)
@start.method	Method to obtain starting-values. Values can be 'use': use values in mod@startval, 'load': use values in start.Rda, or 'rect': determine values from data.	character(1)
@start.maxfac	Search window for starting-values of width parameters. Increase this value to get larger width starting values. Default is 1.	numeric(1)
@min.iterlim	Iteration limit for the minimization procedure.	numeric(1)
@min.boundlim	Boundary limit for the minimization procedure. If a parameter is located on a bound, the minimization procedure will exit if this parameter stays on a bound for @min.boundlim subsequent iterations.	numeric(1)
@min.routine	Which function to use for minimization. Currently only optim.	character(1)
@opt.method	Which optim method to use. Currently only L-BFGS-B.	character(1)
@opt.lower	Lower bounds for parameters $\theta_1, ..., \theta_{10}$.	numeric(10)
@opt.upper	Upper bounds for parameters $\theta_1, ..., \theta_{10}$.	numeric(10)
@sw.type	Which method to use for sandwich estimation. Values are 'diag': use diagonal residual elements, 'full': use full residual matrix.	character(1)
@output	Defines output during minimization: 'none' suppresses output, 'progress' displays a progress window.	character(1)

Table 3: Overview of slots in the options class.

Note that there are two instances of the same model object at this stage, namely in the mod object and in the object saved in the model.Rda file. All functions that use model objects save this object to the model.Rda while also returning it (in this case to mod). Model objects can also be saved manually by calling saveModel(mod). Now that the model is created, starting values have to be defined, and optionally the settings of the minimization routine can be modified.

Customizing the model: Options

A model can be customized by modifying (i) settings of the minimization procedure, (ii) how starting-values of the model are calculated, and (iii) selecting a method for parameter (co)variance estimation. For this, an object of class options (saved in options.Rda) is available in each model directory (for an overview of the slots in the options object, see Table 3). To modify the options load them first by typing:

```
R> opt <- loadOptions(mod)
```

After changing slots of opt a call to saveOptions(opt, mod) saves the options object. It will be loaded automatically by all ARF functions.

Customizing the model: Starting values

The last thing to do before a model can be fitted to the data, is specify starting-values. There are several methods to obtain starting-values, and all can be modified by adjusting the `@start.method` slot of the options object. To let ARF determine the starting-values from the data, set `@start.method` to `'rect'`. The method will then determine starting-values for the location, width, and amplitude parameters of all J regions, using the method described in Weeda *et al.* (2009). Performance of this method can be modified by the value in `@start.maxfac`; increasing this value will lead to larger starting-values for region width.

Alternatively starting-values can be supplied manually. When `@start.method` is set to `'use'`, ARF uses starting-values in the `mod@startval` slot. When set to `'load'`, ARF loads starting-values from the `start.Rda` object in the model directory. The starting-values must be supplied as a vector of length p beginning with the values of θ_1 through θ_{10} for region 1, for region 2, etc.

3.3. Fitting a model

When the starting values are set we can fit the model to the data. The simplest way to fit a model is by calling:

```
R> mod <- fitModel(mod)
```

This will call the minimization routine, given the options in the `options.Rda` file. Note that the above function-call is a clear example of the object-oriented use of the S4 classes: the entire model object (`mod`) is passed to `fitModel`. This function updates (some) slots of the object, and returns the updated object to `mod` again (and it also saves the updated object to the `model.Rda` file).

If `@output.mode` is set to `'progress'`, during minimization an extra window will open, showing progress of the fit procedure. At each iteration the objective function value ($S(\hat{\theta})$), the norm of the gradient vector, and the decrease in the objective function are given. In addition the spatial extent parameters ($\theta_{7j}, \theta_{8j}, \theta_{9j}$) of the regions are displayed with indicators whether they are at a bound. Display of the progress window can be suppressed by setting the `@output.mode` slot of the options object to `'none'`.

After model convergence the Hessian matrix is calculated, as well as the residuals of the model and the BIC. Subsequently the model object is saved and returned. If the model does not converge, the model returns (and saves) a model object with encountered errors in the `@warnings` slot. In this case the `@valid` slot will be set to `FALSE`, otherwise the `@valid` slot will be `TRUE`.

Sandwich (co)variance estimation

The next step is to calculate the sandwich (co)variance matrix of the parameter estimates, as these are used in the Wald statistics procedure. The sandwich estimator requires the (co)variance matrix of the residuals ($\overline{\mathbf{b}} - f(X, \hat{\theta})$) to penalize the parameter (co)variance estimates for model misspecification. ARF has two options to calculate the residual (co)variance matrix, (i) using only the diagonal elements in this matrix (`@sw.type = "diag"`), or using all elements (`@sw.type = "full"`). The `"diag"` method is very fast, but can slightly under-

Slot	Description	type(length)
@consts	Matrix where constants used for the hypotheses are defined (defaults to all zero's)	matrix(J,5)
@stats	Values of Wald statistic	matrix(J,5)
@df1	Vector of (model) degrees of freedom for the F test.	numeric(5)
@df2	Vector of (error) degrees of freedom for the F test.	numeric(5)
@pvalues	P values for the F test on the Wald statistics.	matrix(J,5)

Table 4: Overview of slots in the wald class.

estimate the variance of parameter estimates when data were smoothed with a large kernel width (note that smoothing is not advised, see Section 2.1).

To calculate the (co)variances call:

```
R> mod <- varcov(mod)
```

This will calculate the sandwich (co)variance estimates, and also update the @varcov slot, save model to the model.Rda file, and return it to mod. If any errors occur during estimation, varcov() will add these to the @warnings slot. Errors may occur for example when the (co)variance matrix is singular.

Wald statistics

Wald statistics are calculated for each region in the model separately. There are three hypotheses that can be tested: (i) the spatial extent ($|\Sigma_j|$) of region j is greater than zero, (ii) the amplitude (θ_{10j}) of region j is greater than zero, and (iii) the center ($\theta_{1j}, \theta_{2j}, \theta_{3j}$) of region j is located at a predefined location. To estimate Wald statistics call:

```
R> mod <- wald(mod)
```

This will fill in the @wald slot of the model object with an object of class wald (see Table 4 for the slots of this class), save the model and return in it to mod.

Each column in the matrices of the wald class defines a hypothesis, while rows indicate regions. The fourth column tests the spatial extent hypothesis, the fifth column the amplitude hypothesis, and first three columns test the location hypotheses. To customize hypotheses the matrix in @consts can be modified. For example, to test the hypothesis that the center location of region j differs from a certain value (i.e., $(\theta_{1j}, \theta_{2j}, \theta_{3j}) = (25, 12, 16)$), the first column of the jth row of @consts is set to 25, the second to 12, and the 3rd to 16. For extent and amplitude hypotheses the columns of this matrix are left at zero (testing whether these parameters deviate from 0).

3.4. Finding an optimal model

The range in which to search for an optimal model depends on the data. For example, are the data a contrast between two active conditions or between an active and baseline condition? The former will usually elicit far less activation than the latter.

Determining a range of models

Ideally, the entire range of possible models is estimated, starting with a model with one region and ending the search when the BIC has increased for successive models. In practice this approach can be be very time consuming. Alternatively, the following approach might be adopted.

First, plot the averaged t statistics and threshold this map liberally. This (hopefully) shows some 'clustered' areas of activation. Count the number of clusters, and use this as the initial number of regions in the model. Note that ARF always uses *unthresholded* data (optional thresholding is only used for visualization purposes). Second, fit this model, and check the Wald statistics. If all (or almost all) statistics for extent and amplitude are significant, the optimal number of regions is probably higher than the initial estimate. If several statistics for extent and amplitude do not pass the Wald test, the optimal number of regions is probably lower than the initial estimate. Adjusting the number of regions in large steps (> 10 regions) will give an approximate range in which the optimal model lies. All models within that range can then be fitted and the one with the smallest BIC is chosen.

3.5. Connectivity analysis

Besides the localization of active regions, the **arf3DS4** package can also estimate functional connectivity between active regions. Connectivity estimates are derived after an optimal model is found and requires the dataset used in the GLM analysis: the raw time series data ($\mathbf{y}_{tr}$).

Locating trials in raw time series

The raw time series must be available in each subjects' `/funcs` directory (see Figure 7 for subject `"PP001"`). To calculate the single-trial data ARF needs to know at which time-points in the raw-time series a stimulus was presented. For every run ($r = 1, ..., R$) in a condition, the appropriate times in the raw times-series at which stimuli were presented must be set. To set the timings call:

```
setFuncTimings(subject, condition, run, timings, func_data)
```

This changes the timings[5] for the run `run` (of condition `condition` of subject `subject`) with the values (in seconds) in vector `timings` and links them to the raw time series file specified in `func_data` (which should be in the subjects' `/funcs` directory). For example when the stimuli of the 2nd run of the `"Different Strings"` condition (of subject `"PP001"`) were presented at $10, 30, 60, 70$ and 85 seconds in the experiment, type: `setFuncTimings("PP001", "Different Strings", 2, c(10, 30, 60, 70, 85), "raw_time_series.nii.gz")`. This links the data of the second run to the data in the raw time series. Note that the raw time series do not have to be in a single file. Just change `func_data` to the appropriate filename in the call to `setFuncTimings`.

[5]When stimulus presentation lengths are available these can be added to the values of `timings` by using a `stimlen` attribute: `attr(timings, "stimlen") <- stim_len`. In this case `stim_len` is a vector of the same length of `timings` containing the presentation lengths.

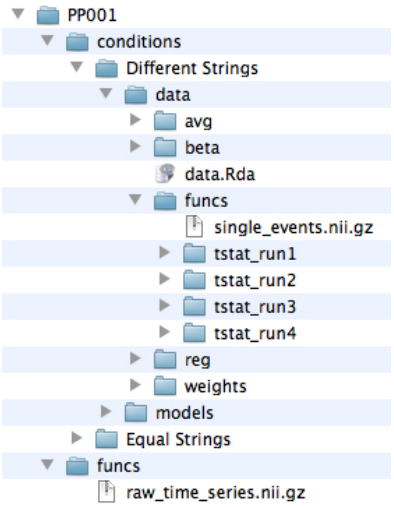


Figure 7: Location of files used for functional connectivity.

Calculating single-trial data

Once the timings are set, the actual single-trial data can be calculated. This is done by calling:

```
R> makeSingleTrialEvents(subject, condition, sefilename)
```

This will concatenate the raw time series data (for runs $r = 1, ..., R$) and regress these to a model with a single HRF[6] regressor for each trial $k = 1, ..., K$ (using the timings we've just linked). When completed the function will create a file named `sefilename` (in this example `sefilename = "single_trials.nii.gz"`) in the `/data/funcs` directory (see Figure 7 for subject `"PP001"`) of condition `condition` of subject `subject`, with the single-trial data.

Estimating connectivity

Now that we have the single-trial data for all voxels in a volume, we only need to estimate the single-trial amplitudes of the individual brain regions (i.e., the trial-by-trial activity of each brain region). To perform this estimation call:

```
R> con <- fitConnectivity(mod, funcfilename)
```

This will use the model estimates in `mod` to obtain, for each single trial k in file `funcfilename`, estimates for the amplitude of each region in the model ($\hat{\gamma}_k$). The vectors of trial-by-trial amplitudes are then used to calculate correlations between regions. Output is an object of class `arfcorrelation` containing slots for the trial-by-trial estimates (`@timebyreg`, $(K \times J)$), correlation matrix M (`@corr`, $(J \times J)$), and their respective p values (`@corr.pval`). The arfcorrelation object is also saved in the `/data/funcs` directory as a file named `funcfilename` with a `.Rda` extension.

[6]By default a standard double-gamma HRF is used, the parameters of the double-gamma function can be adjusted by modifying the values in the `hrf.control`-list passed to `makeSingleTrialEvents`.

4. The ARF example data

In this section a step-by-step ARF analysis is performed based on the example data included in the **arf3DS4** package. The example data are a simplification of data found in real experiments to allow the user to freely experiment with the data without the computational burden of a real dataset. To see the ARF method applied to real data, the dataset used in Weeda *et al.* (2011) is available to users (see Section 5).

The example data consist of three anatomically shaped regions (HarvardOxford Probabilistic Atlas from **FSL**, Smith *et al.* 2004; Inferior Frontal Gyrus (IFG), as used in Weeda *et al.* 2011) placed at spatially distinct locations in a $32 \times 32 \times 16$ map, with white noise added. The time series of each region consist of 10 stimuli convolved with a single-gamma HRF (Glover 1999), spaced 10 seconds apart. The total length of each time series is 110 seconds with a 1 second sampling interval. The amplitude of each stimulus was varied within the time series. Between the three regions the amplitudes of the time series were set to correlate moderately (0.35, 0.5 and 0.7). In total two runs were simulated for the example data.

To analyze the example data, first load the **arf3DS4** package:

```
R> library("arf3DS4")
```

Then, to load the dataset type:

```
R> data("arf-example-data")
```

This will create a function `makeExample()` in the R environment. Running `makeExample()` will create the experiment directories, copy the data-files to the experiment, and load it. The 'setting up' part will thus be performed automatically. The directory structure of the example experiment can be seen in Figure 8. By default, the experiment is created in the directory where the **arf3DS4** package was installed. Optionally, it can be saved in a user specified path. Here we will make the example experiment directories on the desktop (you can modify this to your own directory). To perform this action type:

```
R> makeExample("/users/wouter/Desktop/")
```

R will yield the following output:

```
Experiment correctly set. Experiment saved to /Users/wouter/Desktop/
example-experiment/experiment.Rda
Loaded experiment example-experiment (version 2.5-2)
```

indicating that the experiment passed al sanity checks and is loaded into the environment.

4.1. The experiment structure

When loading the experiment ARF stores several internal values in a separate environment (`.arfInternal`). This environment is crucial to the ARF functions and must therefore always be available. It also contains the experiment object. To see this experiment object type:

```
R> getExp()
```

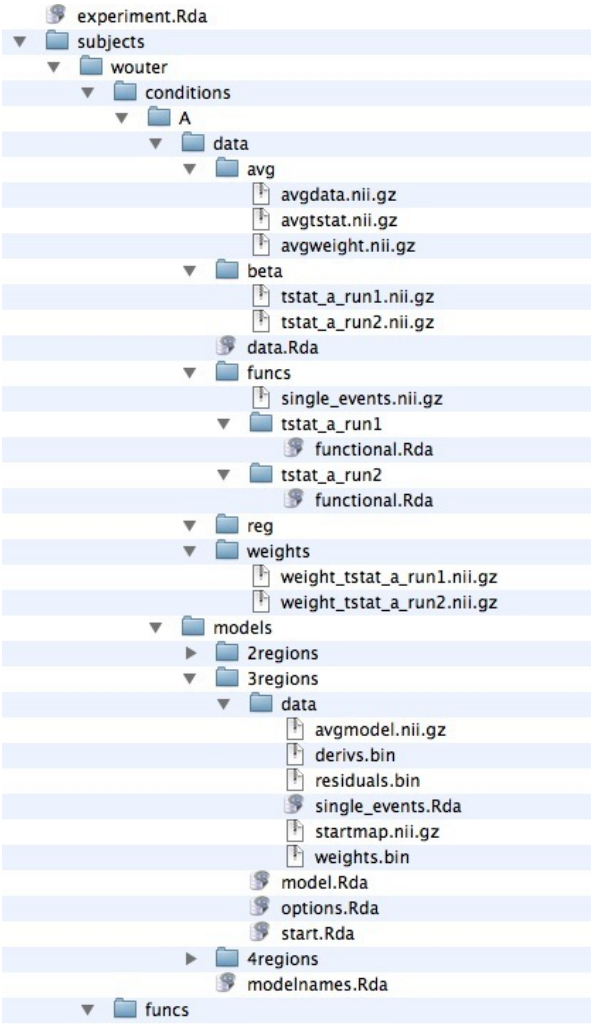


Figure 8: Expanded directory structure of the example experiment, showing all data and object files.

```
[ ARF experiment ]
name:       EXAMPLE-EXPERIMENT
path:       /Users/wouter/Desktop/example-experiment/

subjects[1]
  > wouter

conditions[1]
  > A
```

This shows that the directories are in `/Users/wouter/Desktop/example-experiment/`, and that there is one subject (`"wouter"`) who participated in one condition (`"A"`). The function `makeExample()` has copied two files, `tstat_a_run1.nii.gz` and `tstat_a_run2.nii.gz`. These contain t statistics and therefore are the only files that were copied. Subsequently

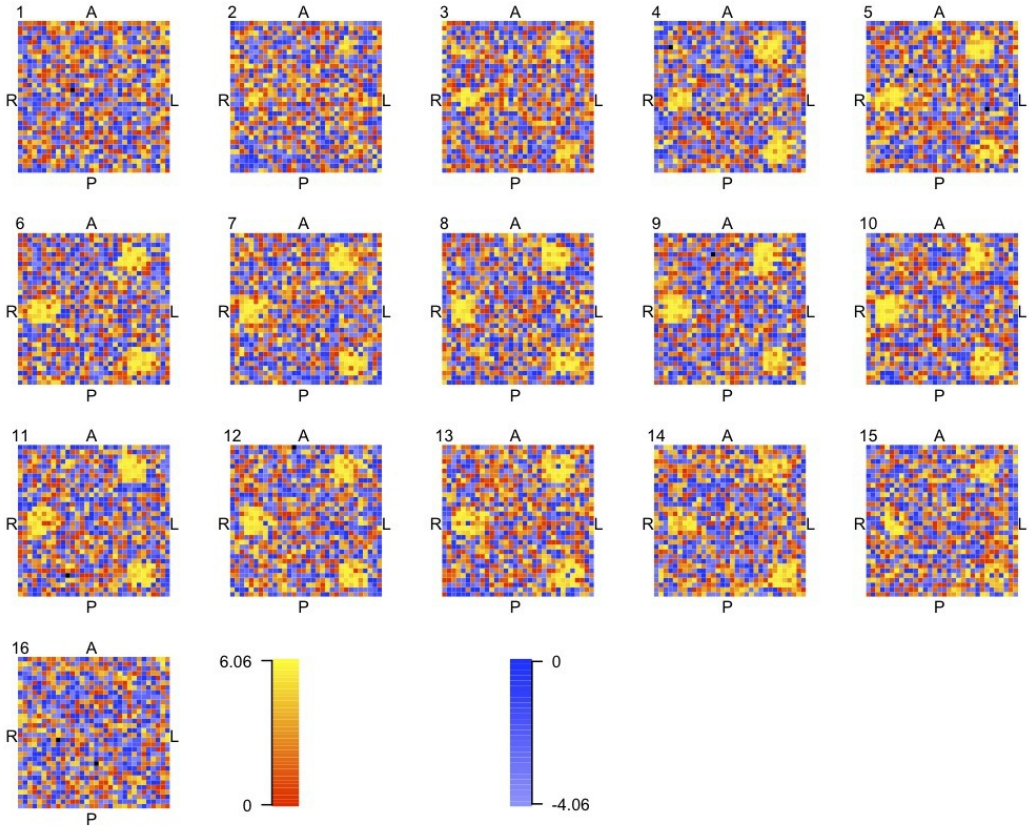


Figure 9: Slice-by-slice plot of the average t statistics of the example experiment.

`loadExample()` was called with the `"set"` argument, creating the weight files (`weight_tstat_a _run1.nii.gz` and `weight_tstat_a_run2.nii.gz`) and the average data files (`avgdata.nii.gz`, `avgweight.nii.gz`, and `avgtstat.nii.gz`).

4.2. Getting a feel for the data

Now that the experiment is loaded, we can set up a model. First, it might be an idea to get a feel for the data. This may help in finding an initial number of regions for a model. We'll first load the data object (not the actual fMRI data) of condition `"A"` of subject `"wouter"`, by typing:

```
R> dat <- loadData(subject = "wouter", condition = "A")
```

This loads the information on where the data files are located, and how they are named. We can use the slots of this object to read in the fMRI data. More specifically, the average t statistics file, which we will plot, is in the `@avgtstatFile` slot (containing the full path of this file). To read in this data (now it is the actual fMRI data) type:

```
R> avgtstat <- readData(dat@avgtstatFile)
```

The fmri data is now in the `avgtstat` object. To plot the data the standard `plot` command can be used:

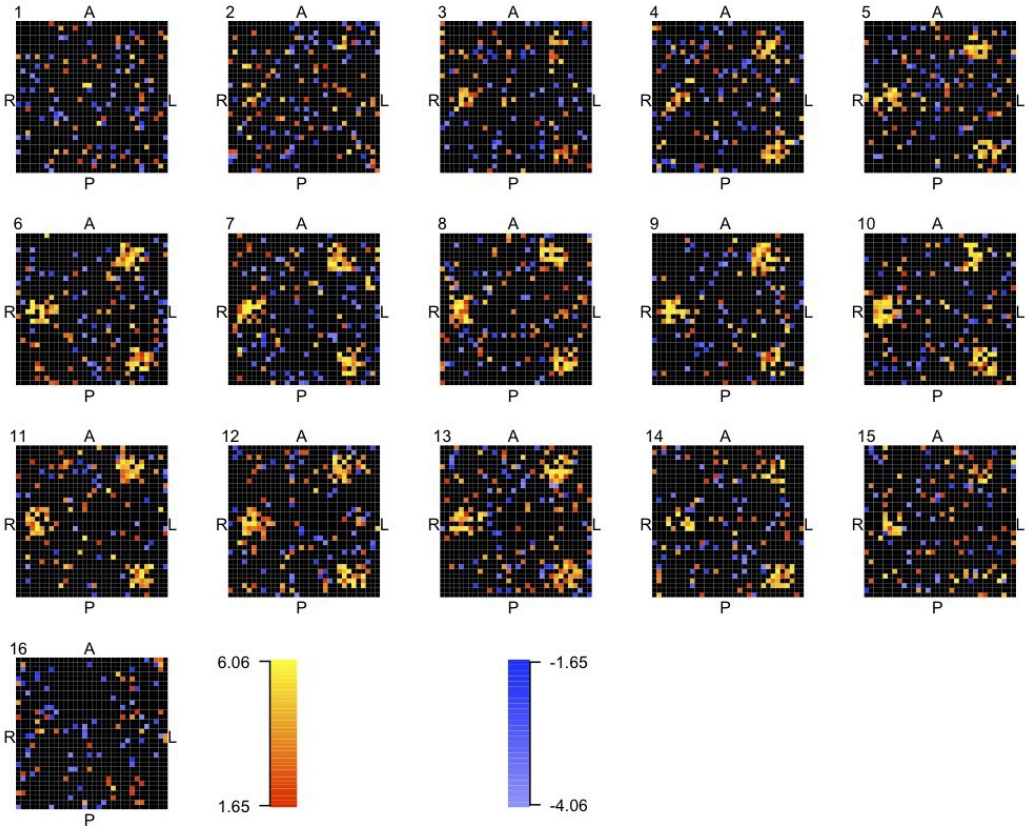


Figure 10: Plot of t values exceeding 1.65.

```
R> plot(avgtstat)
```

This will show a slice-by-slice overview of the data volume (see Figure 9). As Figure 9 shows the entire range of data-values, even the ones close to zero, this might be misleading. We can therefore better show a plot in which small values are omitted. We can for example use a t value of 1.65 (roughly corresponding to applying an uncorrected threshold of $p < 0.05$) to threshold the image. We can plot this image by supplying a `zerotol` argument to `plot`:

```
R> plot(avgtstat, zerotol = 1.65)
```

As can be seen in Figure 10 there are three quite distinct activation blobs, and a lot of scattered single voxel activity. We therefore decide to start with three activated regions.

4.3. Fitting the model

To create a model with three regions we'll call the `newModel()` function:

```
R> mod <- newModel(modelname = "3regions", regions = 3, subject = "wouter",
+     condition = "A")
```

This will create a model named `"3regions"` (with 3 regions in the spatial model) and save it within the `/models/3regions` directory, while also passing it to `mod`. We want the ARF

procedure to determine the starting values from the data, so we'll have to change the default options for this model. To change the `@start.method` slot to `"rect"`, and save the options again, call:

```
R> opt <- loadOptions(mod)
R> opt@start.method <- "rect"
R> saveOptions(opt, mod)
```

The model is now ready to be fitted. Instead of calling the fitting, (co)variance, and Wald functions separately, we'll this time use a wrapper function. By calling `processModel()`, ARF will determine the starting values, run the minimization procedure (`fitModel()`), calculate the (co)variance matrix (`varcov()`), and perform the Wald tests (`wald()`) on the amplitude and spatial extent of the regions in the model:

```
R> mod <- processModel(mod)
```

The process may take one or two minutes (depending on the speed of the computer). While processing the R console will look like this:

```
[ 3regions ]
 arf process for data wouter - A started 2010-10-23 18:31:16
 fitting 3 region(s)
```

After convergence this information will be appended with:

```
 [optim] Optim converged in 232 iterations.
 <modelfit>
  minimum: 17737
    BIC  : 47397
    RMSEA: 1.3

 calculating variance/covariance matrix...ok
 calculating wald statistics...ok
```

If the model is valid, the information of the fitted model is displayed:

```
[ ARF 3regions ]
 regions:  3
 valid:    TRUE
 warnings:

 modelinfo:
  [optim] Optim converged in 232 iterations.
  fit (BIC,RMSEA): 47397 1.3
  minimum:    17737.41
  estimates:
   [  1]  ( 28, 16,  9)  [  1.9   1.9   4.5 ~  -0.1  -0.0   0.0]*   [ 666]*
   [  2]  (  7,  5,  9)  [  1.9   1.8   5.0 ~  -0.0  -0.1   0.0]*   [ 642]*
   [  3]  (  9, 27,  9)  [  1.9   1.9   4.8 ~   0.1   0.1   0.1]*   [ 670]*

  * Wald tests significant at .05 (uncorrected for number of regions)
```

```
modelfile saved in '/Users/wouter/Desktop/example-experiment/subjects/wouter/
conditions/A/models/3regions/model.Rda'
 arf process stopped at 2010-10-23 18:31:43
```

In this case we have a valid model without warnings. The minimization procedure converged in 232 iterations[7], with the minimum of the objective function $S(\hat{\theta}) = 17737.41$. The BIC-value for this model was 47397. The parameter estimates for the three regions are grouped by type, first the three location parameters, then the extent parameters (widths ˜ rotation), and finally the amplitude parameter. This model indicates that there are three (approximately equally sized) regions at locations $(28, 16, 9)$, $(7, 5, 9)$ and $(9, 27, 9)$, all approximately equally active. The * behind the extent and amplitude parameters indicates that the Wald test was significant, which for this model is true for all regions.

4.4. Selecting an optimal model

We now have a valid model, but we don't know if this model has the minimum BIC value. For this we have to fit other models with different numbers of regions and compare the BIC values. Judging from the Wald tests (indicating that all regions have an amplitude and extent greater than zero), we probably need at least these three regions in the model.

We will set up a simple sequence of models by making a loop around the model fit procedure. We'll try models with 2 (just to be sure) and 4 regions.

```
R> for(region in c(2, 4)) {
+    mod <- newModel(modelname = paste(region, "regions", sep = ""),
+    regions = region, subject = "wouter", condition = "A", options = opt)
+    mod <- fitModel(mod)
+  }
```

This will create a new model, one with 2 and one with 4 regions, and fit these models. Notice that the `opt` object (containing the options from the 3 regions model) is passed to the `newModel()` function. This creates a new options object for the new model using `opt` as a template (this saves having to load and save the options at every step). After running we'll now have to decide which model has the minimum BIC. We can do this, by checking the output on the console and seeing which model has the minimum, or more formally, by loading each model and getting the BIC values from there. The fastest way to do this is by calling the `minBIC()` function:

```
R> minBIC("wouter", "A")
```

This will check the models in the `/A/models` directory and return an object of class `sequence` with the names of the models, the number of regions in each model, their BIC values, and minimum ($S(\hat{\theta})$).

```
<ARF sequence>  modelname   regions   minimum       BIC   valid   optimal
```

[7]The actual number of iterations may differ between platforms. The actual parameter estimates, (co)variance estimates, and Wald statistics are equal across platforms within acceptable precision.

```
2regions      2     18917     48479    TRUE     FALSE
3regions      3     17737     47397    TRUE      TRUE
4regions      4     17690     47446    TRUE     FALSE
```

Note that only valid models are shown. The optimal model (with the lowest BIC value) is also indicated. In this case our original model with 3 regions has the minimal value, and is thus the optimal model for this dataset. We can plot the model estimates easily by calling `plot` on the model object (that we'll have to load again, since we used the same `mod` object for the other models too). A model can be loaded by `loadModel()`. This function takes as input the model name, subject name, and condition name. In our example this is:

```
R> mod <- loadModel("3regions", "wouter", "A")
```

It can also take as input a list of model names and a number indicating which model of the list to load. To get a list of all models of a condition of a subject type:

```
R> mnames <- showModels("wouter", "A")
R> mnames

experiment: example-experiment
   subject: wouter
 condition: A
modelnames: [1] 2regions
            [2] 3regions
            [3] 4regions
```

The `"3regions"` model can then be loaded by typing:

```
R> mod <- loadModel(mnames, 2)
```

Now that our model is loaded, we can plot it by calling `plot(mod)` on the model object. This has the same effect as reading in and plotting the model estimates file (`/A/models/3regions/data/avgmodel.nii.gz`), which was saved there after minimization. Note that this file can also be opened in fMRI data viewers like **FSLView** or **MRICron**. Figure 11 shows the estimated model.

4.5. Connectivity analysis

The activated regions of our optimal model can be directly used as ROIs for connectivity analysis. In the example dataset, the single-trial data (`single_events.nii.gz`) are already calculated from the raw time series, and are located in the `/data/funcs/` directory (see Figure 8). The only thing to be done is to call the connectivity analysis:

```
R> con <- fitConnectivity(mod, "single_events.nii.gz")
```

This will calculate the trial-by-trial amplitudes and estimate the connectivity between the regions. The function returns an object of class `arfcorrelation` to `con` and saves a file named `single_events.Rda` in the `/data/models/3regions/data` directory. To see the correlations between the regions we can look at the `@corr` and `@corr.pval` slots of `con`:

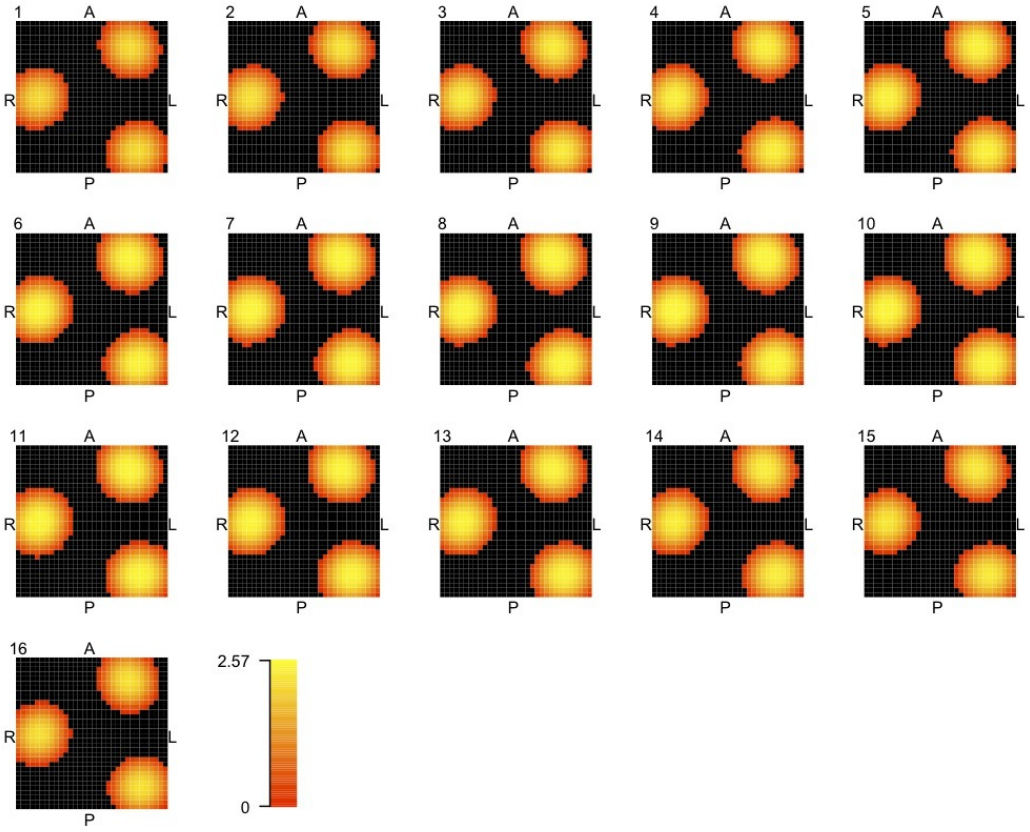


Figure 11: Slice-by-slice plot of the estimated model.

```
R> con@corr
```

```
          [,1]      [,2]      [,3]
[1,] 1.0000000 0.2172220 0.7340559
[2,] 0.2172220 1.0000000 0.1321265
[3,] 0.7340559 0.1321265 1.0000000
```

```
R> con@corr.pval
```

```
             [,1]      [,2]         [,3]
[1,] 0.0000000000 0.3576002 0.0002291734
[2,] 0.3576002365 0.0000000 0.5786998380
[3,] 0.0002291734 0.5786998 0.0000000000
```

This is the 3×3 correlation matrix of the 3 regions with the associated p values. As can be seen only the amplitudes of region 1 and 3 (co)vary significantly ($\rho_{1,3} = 0.73$, $p < 0.0017$, Bonferroni corrected).

5. Empirical data

The outline in the previous section gives an overview on how to perform an analysis with ARF. In practice datasets are far more complex, but the same steps can be followed nonetheless. To give an overview of the analysis process for a real dataset, the data used in Weeda *et al.* (2011) are available online (`http://home.medewerker.uva.nl/w.d.weeda1/`). The data are from a single subject from a single condition of a go/no-go experiment where the go/no-go cues were given subliminally (for details see van Gaal *et al.* 2010; Weeda *et al.* 2011). The archive contains the experiment directory structure and the optimal model. To use the data, extract the directory structure to a certain location and open the script in R. This will load the experiment and open the optimal model with 23 regions. In addition, a slice-by-slice plot of the estimated model is shown. This plot shows the 23 estimated active regions for the *unconscious no-go* > *go* contrast. Values in red-yellow indicate regions where *no-go* activity was greater than *go* activity, values in blue-lightblue indicate regions where *go* activity was greater than *no-go* activity.

6. Conclusion

Activated region fitting (ARF) uses a spatial model to parameterize active brain areas. This allows researchers to test hypotheses of location, spatial extent, and amplitude of these brain areas with greater power. An additional advantage is that the parameters of the activated regions can be directly used to calculate functional connectivity between brain areas.

The use of Gaussian shaped functions to estimate regions of activity has one fundamental assumption, namely that the underlying activity is spatially smooth (Hartvig 2002). This assumption reflects the trade-off between power to detect activation and spatial resolution. In the current implementation ARF uses a relatively simple spatial model (a low number of active regions) that increases the power to detect activation and reduces spatial resolution. Using a more complex spatial model (that is, using multiple Gaussian shapes to model a single active region), will increase the spatial resolution, but does so at the cost of decreased detection power. Currently, ARF also assumes that there is tissue homogeneity within activated regions. In other words, information about different tissue types is not taken into account in the procedure. Future work might include this type of information. For example, shapes can be used that are constrained to remain within the same tissue type.

Smoothing in combination with random field theory (RFT) is another way to increase power in fMRI. It has, however, been shown that RFT can be conservative when the smoothing is insufficient (Nichols and Hayasaka 2003). A possible solution to this problem is to adapt the smoothing procedure, taking into account spatial correlations (Tabelow *et al.* 2006). In ARF spatial information is used in the modeling step, as the parameters (i.e., the Gaussian shapes) of the spatial model are estimated from the data, leading to increased power without smoothing.

The **arf3DS4** package (Weeda 2011), implementing the ARF method, is designed to be compatible with other fMRI analysis packages. This makes the usage of the package somewhat different from standard R use. Especially the emphasis on the directory structure of an experiment, requires different user input. The main difference in this sense is that all data in an experiment (fMRI data, and R objects) are directly saved in, and accessed from, the experiments' directory structure.

The use of S4 classes, makes extensions of the **arf3DS4** package possible. As the slots of the classes are defined (and fixed), developing, for example, new methods for these classes is straightforward. Possible extensions of the **arf3DS4** package include functions for automatically finding an optimal model, or functions to facilitate copying data to the directory structure. The possibilities of using R for the analysis of fMRI data are endless (see for example Tabelow *et al.* (2011)), and the **arf3DS4** package can be a useful addition to the free tools available for fMRI analysis.

Acknowledgments

The authors wish to thank Ingmar Visser for helpful comments on the **arf3DS4** package and Simon van Gaal for providing the empirical data.

References

Bassett DS, Bullmore E (2006). "Small-World Brain Networks." *Neuroscientist*, **12**(6), 512–523.

Biswal B, Yetkin FZ, Haughton VM, Hyde JS (1995). "Functional Connectivity in the Motor Cortex of Resting Human Brain Using Echo-Planar MRI." *Magnetic Resonance in Medicine*, **34**, 537–541.

Bordier C, Dojat M, Lafaye de Micheaux P (2011). "Temporal and Spatial Independent Component Analysis for fMRI Data Sets Embedded in the **AnalyzeFMRI** R Package." **44**(9), 1–24. URL http://www.jstatsoft.org/v44/i09/.

Bowman FD (2005). "Spatio-Temporal Modeling of Localized Brain Activity." *Biostatistics*, **6**(4), 558–575.

Boynton GM, Engel SA, Glover GH, Heeger DJ (1996). "Linear Systems Analysis of Functional Magnetic Resonance Imaging in Human V1." *The Journal of Neuroscience*, **16**, 4207–4221.

Christoffels IK, Formisano E, Schiller NO (2007). "Neural Correlates of Verbal Feedback Processing: An FMRI Study Employing Overt Speech." *Human Brain Mapping*, **28**(9), 868–879.

Eichler M (2005). "A Graphical Approach for Evaluating Effective Connectivity in Neural Systems." *Philosophical Transacations of the Royal Society of London B*, **360**(1457), 953–967.

Forman SD, Cohen JD, Fitzgerald M, Eddy WF, Mintun MA, Noll DC (1995). "Improved Assessment of Significant Activation in Functional Magnetic Resonance Imaging (fMRI): Use of a Cluster-Size Threshold." *Magnetic Resonance in Medicine*, **33**(5), 636–647.

Friston KJ, Ashburner JT, Kiebel SJ, Nichols TE, Penny WD (2007). *Statistical Parametric Mapping: The Analysis of Functional Brain Images*. Academic Press, London.

Friston KJ, Buechel C, Fink GR, Morris J, Rolls E, Dolan RJ (1997). "Psychophysiological and Modulatory Interactions in Neuroimaging." *NeuroImage*, **6**(3), 218–229.

Friston KJ, Holmes AP, Worsley KJ, Poline JP, Frith CD, Frackowiak RSJ (1995). "Statistical Parametric Maps in Functional Imaging: A General Linear Approach." *Human Brain Mapping*, **2**, 189–210.

Genovese CR, Lazar NA, Nichols T (2002). "Thresholding of Statistical Maps in Functional Neuroimaging Using the False Discovery Rate." *NeuroImage*, **15**(4), 870–878.

Glover GH (1999). "Deconvolution of Impulse Response in Event-Related BOLD FMRI." *NeuroImage*, **9**(4), 416–429.

Hartvig NV (2002). "A Stochastic Geometry Model for Functional Magnetic Resonance Images." *Scandinavian Journal of Statistics*, **29**, 333–353.

Honey CJ, Sporns O, Cammoun L, Gigandet X, Thiran JP, Meuli R, Hagmann P (2009). "Predicting Human Resting-State Functional Connectivity from Structural Connectivity." *Proceedings of the National Academy of Sciences of the United States of America*, **106**(6), 2035–2040.

Huettel SA, Song AW, McCarthy G (2008). *Functional Magnetic Resonance Imaging*. 2nd edition. Sinauer, Sunderland, MA.

Jezzard P, Matthews PM, Smith SM (2001). *Functional MRI: An Introduction to Methods*. Oxford University Press, Oxford.

Kriegeskorte N, Simmons WK, Bellgowan PSF, Baker CI (2009). "Circular Analysis in Systems Neuroscience: The Dangers of Double Dipping." *Nature Neuroscience*, **12**(5), 535–540.

Lazar N (2008). *The Statistical Analysis of Functional MRI Data*. Springer-Verlag, New York, NY.

Lukic AS, Wernick MN, Tzikas DG, Chen X, Likas A, Galatsanos NP, Yang Y, Zhao F, Strother SC (2007). "Bayesian Kernel Methods for Analysis of Functional Neuroimages." *IEEE Transactions On Medical Imaging*, **26**(12), 1613–1624.

Marchini JL, Lafaye de Micheaux P (2011). ***AnalyzeFMRI**: Functions for Analysis of FMRI Datasets Stored in the ANALYZE or NIfTI Format*. R package version 1.1-13, URL http://CRAN.R-project.org/package=AnalyzeFMRI.

Muirhead RJ (1982). *Aspects of Multivariate Statistical Theory*. John Wiley & Sons, New York, NY.

Nichols TE, Hayasaka S (2003). "Controlling the Familywise Error Rate in Functional Neuroimaging: A Comparative Review." *Statistical Methods in Medical Research*, **12**, 419–446.

NIfTI Data Format Working Group (2005). *Neuroimaging Informatics Technology Initiative*. URL http://nifti.nimh.nih.gov/.

Penny W, Friston K (2003). "Mixtures of General Linear Models for Functional Neuroimaging." *IEEE Transactions On Medical Imaging*, **22**(4), 504–514.

Raftery A (1999). "Bayes Factors and BIC – Comment on "A Critique of the Bayesian Information Criterion for Model Selection"." *Sociological Methods & Research*, **27**(3), 411–427.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Samanez-Larkin GR, D'Esposito M (2008). "Group Comparisons: Imaging the Aging Brain." *Scan*, **3**, 290–297.

Schwarz GE (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**, 461–464.

Sin C, White H (1996). "Information Criteria for Selecting Possibly Misspecified Parametric Models." *Journal of Econometrics*, **71**(1-2), 207–225.

Smith SM, Jenkinson M, Woolrich MW, Beckmann CF, Behrens TEJ, Johansen-Berg H, Bannister PR, De Luca M, Drobnjak I, Flitney DE, Niazy RK, Saunders J, Vickers J, Zhang Y, De Stefano N, Brady JM, Matthews PM (2004). "Advances in Functional and Structural MR Image Analysis and Implementation as FSL." *NeuroImage*, **23**, S208–S219.

Smith SM, Miller KL, Salimi-Khorshidi G, Webster M, Nichols TE, Ramsey JD, Woolrich MW (2011). "Network Modelling Methods for FMRI." *NeuroImage*, **54**(2), 875–891.

Sporns O (2010). *Networks of the Brain*. MIT Press, Cambridge MA.

Sporns O (2011). "The Human Connectome: A Complex Network." *Annals of the New York Academy of Sciences*, **1224**(1), 109–125.

Tabelow K, Clayden JD, Lafaye de Micheaux P, Polzehl J, Schmid VJ, Whitcher B (2011). "Image Analysis and Statistical Inference in Neuroimaging with R." *NeuroImage*, **55**(4), 1686–1693.

Tabelow K, Polzehl J (2011a). ***fmri**: Analysis of fMRI Experiments*. R package version 1.4-4, URL http://CRAN.R-project.org/package=fmri.

Tabelow K, Polzehl J (2011b). "Statistical Parametric Maps for Functional MRI Experiments in R: The Package **fmri**." *Journal of Statistical Software*, **44**(11), 1–21. URL http://www.jstatsoft.org/v44/i11/.

Tabelow K, Polzehl J, Voss HU, Spokoiny V (2006). "Analyzing fMRI Experiments with Structural Adaptive Smoothing Procedures." *NeuroImage*, **33**(1), 55–62.

van Gaal S, Ridderinkhof KR, Scholte HS, Lamme VAF (2010). "Unconscious Activation of the Prefrontal No-Go Network." *The Journal of Neuroscience*, **30**, 4143–4150.

Waldorp LJ (2009). "Robust and Unbiased Variance of GLM Coefficients for Misspecified Autocorrelation and Hemodynamic Response Models in FMRI." *International Journal of Biomedical Imaging*, **2009**, 723912.

Waldorp LJ, Christoffels I, Van de Ven V (2011). "Effective Connectivity of FMRI Data Using Ancestral Graph Theory: Dealing with Missing Regions." *NeuroImage*, **54**, 2695–2705.

Weeda WD (2011). ***arf3DS4**: Activated Region Fitting, fMRI Data Analysis (3D)*. R package version 2.5-3, URL http://CRAN.R-project.org/package=arf3DS4.

Weeda WD, Waldorp LJ, Christoffels I, Huizenga HM (2009). "Activated Region Fitting: A Robust High Power Method for FMRI Analysis Using Parameterized Regions of Activation." *Human Brain Mapping*, **30**, 2595–2605.

Weeda WD, Waldorp LJ, Grasman RPPP, van Gaal S, Huizenga HM (2011). "Functional Connectivity Analysis of FMRI Data Using Parameterized Regions-of-Interest." *NeuroImage*, **54**, 410–416.

White H (1980). "Using Least Squares to Approximate Unknown Regression Functions." *International Economic Review*, **21**, 149–170.

Worsley KJ, Marrett S, Neelin P, Vandal AC, Friston KJ, Evans AC (1996). "A Unified Statistical Approach for Determining Significant Signals in Images of Cerebral Activation." *Human Brain Mapping*, **4**, 58–73.

Xu L, Johnson TD, Nichols TE, Nee DE (2009). "Modeling Inter-Subject Variability in FMRI Activation Location: A Bayesian Hierarchical Spatial Model." *Biometrics*, **65**(4), 1041–1051.

A. NIfTI files and the `fmri.data` class

The NIfTI format (NIfTI Data Format Working Group 2005) is implemented in the **arf3DS4** package using the `fmri.data` class. Objects of this class contain the fMRI data, NIfTI header-information, and additional information on the file (e.g., `.hdr/.img` pair or single `.nii` file). Table 5 gives the most important slots of the `fmri.data` class that can be accessed. For all NIfTI header information variables please refer to http://nifti.nimh.nih.gov/. An object of class `fmri.data` holds all data in the `@datavec` vector. This vector is indexed with x increasing fastest, then y, and then z. The filename and location of the fMRI data-file are in the `@name` and `@fullpath` slots, the `@extension` slot holds the extension of the filename.

A file can be read using `readData(filename)`, with `filename` indicating the name of the file (including the path; if the path is not included the current working directory is searched). `readData` returns an object of class `fmri.data`. To write to a NIfTI file type `writeData( object, datavector)`, with `object` an object of class `fmri.data` and `datavector` an optional vector of datapoints. If `datavector` is not given the existing values in the `@datavec` slot are used. The `@fullpath`, `@name`, and `@extension` slots are used to determine the filename (and location) to which is written.

Access to the fMRI data of the `fmri.data` object can be direct via the `@datavec` slot (in vectorized form) or using R array indexing (in a 3D/4D array). For example, to show the data of the 16th slice out of a 3-dimensional volume type (with `fmri_object` an object of class `fmri.data`):

```
R> fmri_object[ , , 16]
```

Also elements can be replaced using array indexing:

```
R> fmri_object[12, 32, 16] <- 0
```

this sets the element at $x = 12$, $y = 32$, and $z = 16$ to 0. Note that after this replacement, the file has to be written to disk (using `writeData(fmri_object)`) to save the changes.

A.1. Visualizing fMRI data

There are `summary` and `plot` functions for the `fmri.data` class. The `summary` function gives

Slot	Description	`type(length)`
`@fullpath`	Full path of the location of the file.	`character(1)`
`@name`	Name of the file (excluding extension).	`character(1)`
`@extension`	Extension of the file (`.nii`, `.img/.hdr`).	`character(1)`
`@gzipped`	Is the file gzipped (compressed)?	`logical(1)`
`@dims`	dims[1] indicates number of dimensions (3=3D volumes, 4=4D time series), dims[2..4] indicate x,y, and z dimensions, dims[5] indicate number of volumes.	`numeric(8)`
`@pixdim`	Voxel dimensions (in `@xyzt_units` units).	`numeric(8)`
`@xyzt_units`	Spatial and time units (default is *mm* and *seconds*).	`character(1)`
`@datavec`	Vector of datapoints for all x, y, z, t	`numeric(#)`

Table 5: Overview of slots in the `fmri.data` class.

Variable	Default value	Description
`zerotol`	`1e-3`	Tolerance for zero-values, all values below the value of `zerotol` are set to 0.
`what`	`c("all", "pos", "neg")`	Plot all `"all"` data, or only positive `"pos"` or negative `"neg"` data.
`col`	`c("rgb", "gray")`	Modify the plot colors to be RGB, or grayscale.
`volume`	`1`	Which volume to plot (when data are time series).
`slices`	`1:x@dims[4]`	Vector of slices to plot.
`max.asp`	`NULL`	Maximum aspect ratio between x and y axes.
`device`	`NULL`	Which device to plot to (can be for example `pdf()`, `x11()`, or `jpeg()`). By default (when `device = NULL`) the default plot device of R is used.

Table 6: Plot options for the `fmri.data` class.

information on the distribution of the data (quantiles, interquartile range, minimum, maximum, mean, and median). Plotting an `fmri.data` object by default gives a slice-by-slice overview of the data. There are several options that can be passed to the plot function to modify how the data is plotted. The defaults of a call to `plot(fmri_object)` are shown in Table 6. When plotting multiple slices to a graphics device that is to small, an error (most likely: `figure margins are too large`) is thrown. Increasing the size of the graphics device will usually solve this problem.

B. S4 classes

All `arf3DS4` objects belong to an S4 class. These classes have predefined slots, holding different types of information. To access a slot of a class use the '`@`' operator. So, for example to access the `@estimates` slot of object `mod` of class `model` type:

```
R> mod@estimates
```

To replace this slot with the vector `c(1, 2, 3)` type:

```
R> mod@estimates <- c(1, 2, 3)
```

Alternatively **arf3DS4** has specific accessor and replacement functions for all classes. These functions are defined as follows: `.classname.slotname(object)`, where `classname` is the name of the class, `slotname` is the name of the slot and `object` is an object of class `classname`. For example to access the `@estimates` slot of object `mod` of class `model` type:

```
R> .model.estimates(mod)
```

This method can also be used to replace the slot of the object: `.classname.slotname(object) <- value`, will replace the appropriate slot of `object` with the value in `value`. For example, to replace the `@estimates` slot with `value` type:

```
R> .model.estimates(mod) <- value
```

The slots in the S4 classes have a predefined type (e.g., numeric, logical, character, matrix) and trying to replace a slot with an object of a different type will result in an error. Most S4 class objects in **arf3DS4** also have default values of the slots. Objects of S4 classes can be created (with slots at their default values) by typing: `new("classname")`, with classname being the name of the class. To view the available slots of a class type `slotNames(object)`. To also view summaries of the values of these slots type `str(object)`.

Affiliation:

Wouter D. Weeda
University of Amsterdam
Department of Psychology
Roetersstraat 15
1018 WB Amsterdam, The Netherlands
E-mail: w.d.weeda1@uva.nl
URL: http://home.medewerker.uva.nl/w.d.weeda1/
Telephone: +31/20/5256908

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 44, Issue 14 *Submitted:* 2010-11-14
October 2011 *Accepted:* 2011-06-28