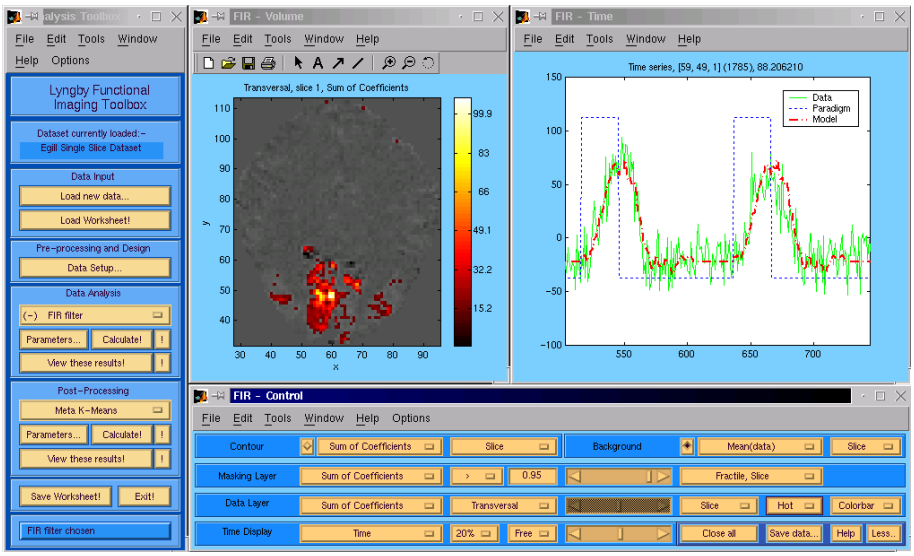
LYNGBY
Matlab toolbox for functional neuroimaging analysis



Peter Toft, Finn Årup Nielsen,
Matthew G. Liptrot, and Lars Kai Hansen

IMM — Informatics and Mathematical Modelling
DTU — Technical University of Denmark

Version 1.05 — August 14, 2006

Contents

List of Figures

List of Tables

Chapter 1

Introduction

$Revision: 1.26 $
$Date: 2002/08/14 16:52:07 $

1.1 The Purpose of the Lyngby Toolbox

Lyngby is a Matlab toolbox for the analysis of functional magnetic resonance imaging (fMRI) time series. The main purpose of the toolbox is to model four-dimensional fMRI data (i.e. 3D spatial volume over time) and to derive parameter sets from them that will allow easy interpretation and identification. The toolbox was primarily written for the analysis of experimental data obtained from controlled multicentre trials - The Human Brain Project "Spatial and Temporal Patterns in Functional Neuroimaging", carried out by the International Consortium for Neuroimaging. All of the methods have low-level modelling functions and a graphical user interface (GUI) interface for easy access to the data and modelling results. It is important to realize that no single model can grasp all the features of the data. Each of the models have their own contributions, and the assumptions underlying the models are very different in nature.

The Lyngby toolbox can import data in various different formats, and it is also able to cope with non-supported formats with very little user-intervention. In addition to the large set of data modelling strategies, there is also a choice of pre-processing steps and routines, and the ability to perform post-processing on the modelling results.

The toolbox was developed on Linux and SGI platforms and should work on all platforms with Matlab version 5.2. Lyngby was previously supported with Matlab 4.2 and some of the functions will still work with this version. There are still some teething problems with running the toolbox under MS Windows however, due to some differences in the way Matlab operates between the Unix and Windows flavours. We are working to overcome these problems, but as they are due to subtle, undocumented Matlab 'inconsistencies', it is something of a trail-and-error process. For the time-being, we encourage people to use the toolbox in Matlab under Linux, where the majority of our development is now carried out. Any advice regarding the development of the toolbox in Matlab for MS Windows is, of course, welcomed.

In addition to this manual, there are also a few other pieces of documentation to aid use of the toolbox. A *"Getting Started"* guide has been written to show the first-time user how to download, install and run the toolbox as quickly and simply as possible. In addition, an *"Example Analysis"* guide demonstrates how Lyngby can be used to analyse one of the sample datasets, extract some meaningful results, and then interpret them. Both these documents are included in this manual as appendices.

1.1.1 Models Available in the Toolbox

Currently, the toolbox includes the following modelling methods (The exact details of the algorithms are covered later on in the manual):

Cross-Correlation Cross-correlation with the paradigm (The activation function, which usually is a square shaped design function). A well-established method suited for estimation of delay and activation strength from a given paradigm.

FIR filter A general linear regression model of finite length. The time series are modelled as a convolution between the paradigm and a linear filter of a finite length, on a per voxel basis.

Exhaustive FIR filter The same as the FIR filter, but using an exhaustive search of all FIR models up to a specified filter length. The optimal filter is found from generalization theory.

K-means Clustering A non-linear statistical method for clustering (labelling) the data using either the raw time series or the cross-correlation with the paradigm. This method is suited for identification of areas with similar activation and delay. The method is described in (Bishop, 1995, pp. 187-189) and has been used for fMRI in (Toft et al., 1997; Goutte et al., 1999).

Grid Search Lange-Zeger model A parameterized convolution model with a grid search of optimal parameters with zoom built in. The same as above but a non-iterative grid search scheme for estimating the parameter is used. The model estimates three parameters that have easy physical interpretation.

Iterative Lange-Zeger model A parameterized convolution model with an iterative scheme for estimating the model parameters. The model estimates three parameters that have easy physical interpretation.

The Ardekani t-test A linear transform of the time-series mapped to a subspace controlled by the paradigm. In this method the activation estimator can be approximated to be Student t-distributed, hence a statistical measure of the correlation to the paradigm is found. The method has been described by Babak Ardekani and Iwao Kanno (Ardekani and Kanno, 1998).

Ardekani F-test A linear transform of the time-series mapped to a subspace controlled by a finite size Fourier based subspace. In this method the activation estimator can be approximated to be F-distributed, hence a statistical measure of the energy in the subspace relative to the energy lying outside of this subspace is found. The method has been described by Babak Ardekani and Iwao Kanno (Ardekani and Kanno, 1998).

Ardekani F-test with nuisance subspace A variation of the Ardekani F-test, where a nuisance subspace is identified and extracted from the signal. The method is described in (Ardekani et al., 1999).

Ordinary t-test From a square wave activation function each of the time series is split into an activation part and a baseline part. In this model the difference in means relative to a deviation measure is given a statistical interpretation.

The Kolmogorov Smirnov test In the Kolmogorov Smirnov-test each of the time series is split into an activated part and a baseline part. The maximal distance between the histograms of the two parts is taken as a measure of match. A probabillity measure is also derived.

Neural Network regression A feed-forward artificial neural network is trained to predict the time series for each voxel. If the neural network is better at predicting the time series than a "null"-model for a specific voxel, that voxel can be said to be activated. The neural network regression is a non-linear generalization of the FIR filter model.

Neural Network Saliency A feed-forward artificial neural network is trained to classify the scans according to the paradigm. After the training, the neural network is analyzed to reveal which voxels were the most important in the prediction.

Poisson Filter An fMRI time-series model.

Singular value decomposition (SVD) The same as principal component analysis (PCA).

Independent component analysis (ICA) A multivariate methods to find independent (not necessarily orthogonal) components with an implementation of the simple Molgedey-Schuster algorithm.

Strother CVA Canonical variate analysis.

Strother Orthonormalized PLS The PLS (partial least square) method finds the images and sequences that explain the majority of the variation between two matrices. The two matrices in the Strother orthonormalized PLS are the datamatrix and a designmatrix that is constructed by putting the scans in the same period in each run into the same class.

1.2 Organisation of this Book

This book fulfills several requirements. The first - explaining what the `Lyngby` toolbox is for, what it can do, who developed it and other related information - is here in the first chapter. The second - the "user manual" element - is contained in Chapter 2. It explains how to go about using the toolbox, including getting your own data into it, and shows you how to do the different analyses. Consider it as a "How to..." guide. The following three chapters explain the different stages of using the toolbox in more detail and provide more background information. The first of these covers the different data formats that `Lyngby` supports, how data is loaded and stored in the toolbox, and the pre-processing that is usually done to the data before any analysis is performed. The second (Chapter 4) looks at the actual data analysis stage and considers each of the modelling algorithms in detail. The final chapter of this group (Chapter 5) covers post-processing of the data. This involves the analysis of the results from the (previous) modelling stage by way of meta-analysis algorithms. Finally, the appendices cover any derivations of modelling algorithms and other similar details refererred to in the main part of the book.

In summary:

Chapter 1 Introduction to `Lyngby` – what is it, who wrote it, and what is it for?

Chapter 2 The main user manual – a step-by-step guide to the user interface.

Chapter 3 Details of the preliminary steps – the data formats and the pre-processing stages.

Chapter 4 Details of the analysis algorithms in the toolbox.

Chapter 5 Details of the post-processing stages (meta-analysis).

Appendix A Glossary of fMRI and `Lyngby` terms.

Appendix B Detailed list of all the functions and script files within the toolbox.

Appendix C Derivations of the algorithms used in the toolbox.

Appendix D The Getting Started Guide – useful for first-timers.

Appendix E Example Analysis – An step-by-step example of how to to use the toolbox to analyse fMRI datasets.

1.3 Explanation of the Typographic Conventions

Within this manual, we have tried to improve the readability of the text by using certain typographic conventions. For instance:

```
Any reference to Matlab code,
either on the commandline (at the >> prompt)
or within a Matlab script (*.m) file,
will be written in monospace with all
the inherent formatting included.
```

In addition, commands written at the Matlab commandline will have the Matlab prompt included on the left-hand side, e.g.:

```
>> help lyngby
```

Any `Lyngby` variables or Matlab files mentioned in running text, such as `PARADIGM` or `data_paradigm.m` will also be in bold monospace.

Any reference to a particular button in the `Lyngby` GUI will have a box around it, whilst *Window Titles* and *Window Menus* will be shown in italics.

1.4 fMRI Analysis, Matlab and the `Lyngby` Toolbox

The `Lyngby` toolbox was created for the analysis of fMRI images using advanced mathematical and statistical methods in a way that was easily accessible to researchers from all fields. fMRI is still a relatively new field, and this is reflected in the continually growing number of analysis routines available. Originally, methods developed for the analysis of PET images were adapted to work on fMRI data, and some of these are still present in the toolbox. However, we have also added a considerable number of innovative analysis methods, and are continually developing more. One of the purposes of the toolbox is to enable researchers to add their own algorithms with relative ease, without having to worry about handling the data input, image display, statistics or saving of results. By writing the toolbox in the universally adopted Matlab package, and allowing the code to be open, we hope that other researchers will be able to add their own algorithms with ease. We also hope that they will consider submitting to us any they think would be useful to other people in the field so that we may include them in a subsequent release of the `Lyngby` toolbox. Of course, any other suggestions for developments of the toolbox will also be greatly welcomed.

1.4.1 Definitions and Glossary

It would probably be useful at this point to define some of the terms used throughout this book. Although the fMRI field is developing rapidly, some of the terms used are still ambiguous and those coming into the field for the first time may not appreciate the difference between, for example, "volume" when it refers to a 2D-slice-with-time vs. a spatial-3D-volume. For consistency and clarity we will define what we mean by the standard terms, along with some `Lyngby`–specific terms. These are all in the Glossary at the back of the book — see Appendix A.

1.5 Support on the Web and Downloading the Toolbox

The `Lyngby` toolbox has a mailing list where the latest development is announced, and its own homepage:

```
http://hendrix.imm.dtu.dk/software/lyngby/
```

The parts of the `Lyngby` homepage relating to downloading of software and documentation is updated automatically every day. The `Lyngby` toolbox can either be downloaded file by file (more than 300 files), or you can get the compressed tar file that contains all the files. You can download the latest version from:

```
http://hendrix.imm.dtu.dk/software/lyngby/code/lyngby.tar.gz
```

Note that currently the software and documentation is password protected. Please fill in the form on the `Lyngby` *homepage or contact the authors regarding access.*

We use CVS to control the code revisions, which enables very easy updating and automatic generation of the web documentation. On the `Lyngby` homepage it is also possible to see which files have changed (and view the change logs) over the last 7 days and last 30 days.

The (closed) mailing list `lyngby@isp.imm.dtu.dk` can be joined by sending an email to

```
majordomo@isp.imm.dtu.dk
```

with the only line in the body being:

```
subscribe lyngby
```

The mailing list is used for discussions, problems, and announcements.

1.6 Other Software

Other software packages for the analysis of functional neuroimages do also exist, most notably SPM from Wellcome Department of Cognitive Neurology (Friston et al., 1995b) and Stimulate developed by John P. Strupp from University of Minnesota Medical School (Strupp, 1996). (Gold et al., 1998) gives a review of these two together with several other software packages.

1.7 The Lyngby Development Team

Lyngby has been designed and implemented by a modeling group at Informatics and Mathematical Modelling (IMM, formerly called Department of Mathematical Modelling), Technical University of Denmark (DTU). The head of this group is Lars Kai Hansen. The programming has been carried out mainly by Finn Årup Nielsen, Peter Toft and Matthew Liptrot as well as Carsten Helbo. Peter Toft and Carsten Helbo are now involved in other work.

1.7.1 Acknowledgments

Furthermore, Cyril Goutte, Lars Kai Hansen, Ulrik Kjems, and the student Pedro Højen-Sørensen have supplied methods and code for the package.

Jan Larsen should also be mentioned for his long-term support and discussions.

Much valuable input has come from the HBP partners, especially Stephen Strother, Nick Lange, and Babak Ardekani.

This work has been supported by the The Danish Research Councils through the programs "Functional Activation in the Human Brain" (9502228), "CONNECT" Computational Neural Network Center (9500762), the US Human Brain Project "Spatial and Temporal Patterns in Functional Neuroimaging" (P20 MH57180), and the European Union project MAPAWAMO.

1.7.2 Contact us

Besides the mailing list lyngby@isp.imm.dtu.dk, we can also be contacted by email directly – see the list in the table below.

Lars Kai Hansen	Associate Professor	lkhansen@isp.imm.dtu.dk
Finn Årup Nielsen	Post Doc	fn@imm.dtu.dk
Matthew G. Liptrot	Research Assistent	mgl@imm.dtu.dk

Peter Toft can be contacted via pto@sslug.dk, however he is no longer employed at IMM or officially involved with Lyngby.

Chapter 2

User Manual

$Revision: 1.30 $
$Date: 2002/08/14 15:54:22 $

2.1 Introduction

This chapter deals with everyday usage of the toolbox. It is a *"How do I...?"* guide showing, in a step-by-step manner, how data is loaded in, how a typical analysis would be carried out, what results can be obtained, and how these results can be viewed and saved. More information on each of the stages, such as a description of the available data formats and the different modelling algorithms, can be found in the later chapters.

2.2 Installation and Getting Started

2.2.1 First-time Users

It is recommended that, if you have not yet installed `Lyngby`, you first read a seperate document entitled *"Getting Started"* which takes the user step-by-step through downloading, installing and running the toolbox. The aim is to have a first-time user able to start experimenting with real data within a couple of minutes of downloading the `Lyngby` toolbox. As an additional help, the `Lyngby` toolbox also comes with a sample dataset that allows the first-time user to quickly get started with using `Lyngby` without first spending time working out how to get their own data loaded in.

It is highly recommended that first-time users then read the follow-up document entitled *"Example Analysis"*. This takes the user through a typical fMRI analysis session, from loading in the data and ensuring all the pre-processing steps are done, to performing some modelling of the data, and finally through to analysing the results obtained. This will give the user a good overview of how `Lyngby` can help them in the analysis of their own data. Both of these documents are available on the `Lyngby` website for seperate download, and are also included in this book as Appendices D and E, respectively.

2.2.2 Using `Lyngby` On Your Own Data

If you want to use `Lyngby` to analyse your own data, there are two main approaches. If the data is in one of the supported formats (Analyze, Vapet, Stimulate or Raw) then you can simply load the data directly into the toolbox. However, if you data is in any other format, you will need

a few simple *conversion files* to act as format translators. These are very simple to write, and once stored alongside the data, will allow seamless loading of your data with little or no user intervention required. These files, the supported data formats and detailed instructions on how to load data into the toolbox are given later on in this chapter and in the next.

2.3 Using Lyngby

From this point we will assume that the user has followed the *"Getting Started"* guide and installed Lyngby correctly. It may also be advantageous for the user to have worked through the *"Example Analysis"* guide to see how the Lyngby toolbox can be used to analyse fMRI data.

2.3.1 Starting Lyngby

Lyngby can be run either through the graphical interface or through the Matlab commandline. The toolbox consists of over 300 Matlab script (*.m) files, which can all be executed from the Matlab prompt. This allows the advanced user great flexibility and the ability to add and alter the code as desired, as well as the option of writing batch scripts to run Lyngby automatically, or on large jobs. However, it can be difficult to remember all the different script names and their required parameters, so the novice user will find that the graphical interface is far easier to use. Here the user is prompted for all the required information, removing the need to call anything from the commandline (once the toolbox is started).

To start the graphical interface, change directory within Matlab to the directory that contains the data files that are to be processed, and then type:

```
>> lyngby
```

If you get an error saying "`??? Undefined function or variable lyngby.`" it is probably because you haven't set up the path to the `lyngby` functions. To fix this (assuming that you have placed the `lyngby` code in the directory `/usr/local/lyngby`), simply type:

```
>> path(path, '/usr/local/lyngby');
```

You can put this command in your `~/matlab/startup.m` file.
Details on how to use Lyngby via each method are given in Sections 2.5 and 2.6 respectively.

2.3.2 Getting Help

An overview of all the functions in the toolbox can be obtained through the standard matlab help function:

```
>> help lyngby
```

You can also get help for each individual function, for example:

```
>> help lyngby_fir_main
```

To distinguish the functions in this toolbox from other Matlab toolboxes all the functions have the prefix `lyngby_*`. All the functions associated with the graphical interface to lyngby are called `lyngby_ui_*`. Furthermore, functions to each algorithm also have a special prefix. For example, the FIR filter algorithm uses the files: `lyngby_fir_*` and `lyngby_ui_fir_*`. A

very useful index of all the toolbox function files and their inter-dependencies can be found on the `Lyngby` website.

Within the GUI, there are Help buttons at various stages. Clicking on a Help button will bring up a single dialog box explaining the purpose of the different windows and the options within them. In addition, most of the windows have a *"Tooltips"* option, invoked from the *Options* window menu or with the `<Ctrl-T>` key combination. These Tooltips will pop-up handy explanations for any button over which the mouse pointer rests temporarily. They are meant for the new user, although they are set to be off by default as they can become distracting after a short while.

2.4 Data Setup

This section will deal with getting data into `Lyngby`. It will cover the file formats that `Lyngby` can currently read and will also explain how your own data format, even highly custom or non-standard formats, can be read in.

2.4.1 How `Lyngby` Reads in Data

There are two ways to get new data into `Lyngby` — either by the file formats already supported, or via a conversion process that will allow any non-supported file format to be read. For the supported formats, nothing needs to be done outside the GUI in order to read in the data — all the required information is extracted from the header files and from details entered into the GUI's *Load New Data* window. Non-supported file formats can be read-in by the use of *"conversion files"* (`data_readdata.m` and `data_readinfo.m`). These are small matlab files which act as header and data-reading files for the `Lyngby` program so that it can understand the data. These are very simple to write, and detailed examples of how to do this are given later.

For all file formats, there may be external influences on the data that you will also need for your analysis. For instance: the *paradigm* and *run* signals, any previous *voxel-masking*, or a *time-mask* to remove unwanted scans. These can be specified through the GUI, but it may be easier to save this information as small matlab files that can then be loaded in at the same time as the data. Examples include `data_paradigm.m` and `data_run.m`.

In addition, it may be easier to save the parameters that specify the data, such as image size and data location, in a seperate file as well. Then the user does not have to enter them every time data is to be loaded. This file will work for any data format. This file is called `data_init.m`

Details on how to create all these files are also given later.

2.4.2 Setting-up the Conversion Files for Non-supported Data Formats

It is easiest to explain how to set-up the files that perform the conversion by use of an example. To do this, we will use the *Jezzard Turner Friston* (Friston et al., 1994) dataset: Visual stimulation and a single coronal slice. This is one of the sample datasets available from the `Lyngby` website.

The data is stored in a filetype that is not recognized by `Lyngby`, so we have to make our own functions that read the data: `data_readinfo.m` and `data_readdata.m`, the latter of the two being the most important.

These files are called from the *Load new data...* window of the GUI. Several `Lyngby` internal functions are called once the data is required to be loaded. The first of these is

`lyngby_getdata.m`. This then calls `lyngby_getvolume.m`, which, for non-supported data, then calls the user-created file `data_readdata.m`. This acts as the actual data-reader, returning the data to the program in the form of a vector. Details about the actual shape and order of the volume are not specified in this file, though of course they will be required in order to write the data-reading function within it that will actually return the data.

However, `Lyngby` itself will not know the shape of the data, the number of runs/volumes/time-points etc. This information has to be entered into the GUI, or can be stored in a matlab file (`data_init.m`) to save the user typing in the relevant details each time the data is loaded.

In summary, then, there is only one file that is absolutely necessary when reading in non-supported data (`data_readdata.m`), although up to four others may be used. These others are optional but do make life easier. Any file that is to be supplied by the user will have a filename stem of `data_*.m`. Their individual purposes are summarized below:-

`data_readdata.m` The only necessary file required for reading in non-supported file formats, it contains a few lines of matlab code that actually return the non-supported data as a vector. It is also where the user will put in any re-organisation of the data in order to get the byte-read-order correct. The user must use the standard Matlab functions to re-organise their data to match this ordering. An example of how to do this is given later.

`data_readinfo.m` This file is only required if the user wants to use the facility within `Lyngby` to check the headers of their data files. This is accessed by pressing the Try to setup! button in the *Load new data* window. It is therefore not an essential requirement for loading in non-supported data formats, and indeed may become redundant in future versions of the toolbox.

`data_paradigm.m` This is used to specify the paradigm (the external reference function) which is used in the actual analysis stage. It can be specified within the GUI, but it is far easier to keep it within a file that is loaded in automatically. It is simply a 1D vector indicating "on" and "off" time points, and as such usually only requires a single line of code. The GUI can be used to make changes to the paradigm function if required.

`data_run.m` This specifies which time-points belong to which *run* within a given *experiment* or *session*. As for the paradigm file, this too can be specified through the GUI, although only if the lengths of all the runs are equal. But once again, it is far easier to keep the information in a file that is loaded automatically. Changes to the function can also be made from within the GUI if required.

`data_init.m` This file contains all the parameters that the user would normally enter in the "Load new data" window. As such, it also is not essential, but it makes life easier because it then saves the user specifying the parameters every time data is loaded. It is simply a file of variable assignments, specifying such parameters as the size and shape of the volume, the number of scans and runs, and a TIME_MASK variable which can be used to remove unwanted scans.

Note that the latter three files can be used for *all* data formats, and it is only the first two which are specific to custom data.

2.4.3 Examples of the Data Conversion Files

This section looks at how to write the actual code within the data conversion files. It uses the Jezzard dataset as a basis for the code examples. This is a single trial, single slice fMRI experiment, with 64 *volumes* (i.e. *time points* or *frames*), each of size 64-by-64-by-1.

2.4.3.1 The data_readdata.m and data_readinfo.m Files

The contents of the `data_readinfo.m` file are read when the user wants to pass the header information contained within the file into the *Load Setup* window. This is achieved by pressing the Try to setup! button. An example of the file is shown below:

```
function  [siz, vdim, name, datatype] = data_readinfo(index);

  siz = [64 1 64];
  vdim = [0.003 1 0.003]
  name = 'jezzard';
  datatype = 'short'
```

Pressing the Try to setup! button then attempts to locate the file in the current directory and read the contents. These are then placed in the correct fields of the *Load Setup* window, and the relevant status boxes are set to *Header*. For all the other file formats, pressing the Try to setup! button ignores any `data_readinfo.m` file, and instead the relevant header files for the chosen format are attempted to be read and the extracted parameters again passed into the relevant fields in the *Load Setup* window, with the associated status boxes also being changed to *Header*.

The other function, `data_readdata`, is the function that should return the actual data. The data has to be flipped because, in this case, the original ordering is not the same as that used by `Lyngby`.

The `Lyngby` toolbox will always assume that the vector data returned from `data_readdata` is of the following form:-

$$V = [P_1(x, y, z), P_2(x, y, z), P_3(x, y, z), \ldots]$$

where:

P refers to an individual data point

(x,y,z) refers to the location in 3D space of the data point

x refers to the sagittal slice (i.e. ear-to-ear/left-right)

y refers to the coronal slice (i.e. back-to-front)

z refers to the transaxial slice (i.e. bottom-to-top)

. . . and so this is the order into which the user must reshape their own data.

The X index changes quickest, followed by the Y index, with the Z index changing the slowest. Thus, the data cycles through the X's first, then the Y's, and finally through the Z's. So, for a 10-by-10-by-10 cube, the data should be in the order:-

$P_1 = (x1, y1, z1)$, $P_2 = (x2, y1, z1)$, $P_3 = (x3, y1, z1)$, ... , $P_{10} = (x10, y1, z1)$, ...
$P_{11} = (x1, y2, z1)$, $P_{12} = (x2, y2, z1)$, $P_{13} = (x3, y2, z1)$, ... , $P_{20} = (x10, y2, z1)$, ...
...
$P_{101} = (x1, y10, z1)$, $P_{102} = (x2, y10, z1)$, $P_{103} = (x3, y10, z1)$, ... , $P_{110} = (x10, y10, z1)$, ...
$P_{111} = (x1, y1, z2)$, $P_{112} = (x2, y1, z2)$, $P_{113} = (x3, y1, z2)$, ... , $P_{120} = (x10, y1, z2)$, ...
...
$P_{991} = (x1, y10, z10)$, $P_{992} = (x2, y10, z10)$, $P_{993} = (x3, y10, z10)$, ... , $P_{1000} = (x10, y10, z10)$.

Note that the direction along each of the three orthogonal axes is also important. The `Lyngby` toolbox uses the Talairach co-ordinate space as its reference. The origin of this space is located at the centre of the brain, with the directions of the axes as given in Table 2.1

-	+
Left	Right
Back	Front
Bottom	Top

Table 2.1: Direction of the axes in Talairach coordinate space.

The `Lyngby` toolbox will assume that the data is written with the most negative values first, increasing through the origin, and up to the most positive values last.

For the Jezzard data, the file looks like:

```
function V = data_readdata(index);

  fid = fopen(sprintf('jezzard0716phot1_unwarp.%03d', index), 'r', 'ieee-be');
  if fid==-1
    error('readdata: Could not open file');
  end
    fread(fid, 8, 'char');
    V = fread(fid, 'int16')';
    V = reshape(fliplr(flipud(reshape(V,64,64))),1,64*64);
  fclose(fid);
```

Let's look at this in more detail.

The first thing to do is open the file with:

```
fid = fopen(sprintf('jezzard0716phot1_unwarp.%03d', index), 'r', 'ieee-be');
```

The `fid` is the file pointer. Note that the filename is actually the file-stem of the relevant data files, and that the individual datafile is specified by the *index* parameter. This *index* is passed to the `data_readdata` function when it is called, and so this function will be called for each "volume" that is to be read in. The number of times that `data_readdata` is called is given by the difference between the *Start* and *Stop* scan indices as specified in the *Load Data* window or in the `data_init.m` file. The former takes preference. (Note that some data formats don't use a different file for each time-point, or frame, and instead the whole time-series is contained

within a single file. When this is the case, the *File Pattern* field, and the *Start* and *Stop* indices are not used.)

The `r` means that the files are to be opened "read-only", while the `ieee-be` specifies that they are in "big-endian" format. As a rule, files stored on Intel (Pentium) and DEC Alpha machines are "little-endian" (`ieee-le`), whilst those stored on Sun and SGI machines are "big-endian" (`ieee-be`). Make sure that the correct format is chosen for your architecture.

The next few lines check that the file was actually there and that it could be opened, returning an error if not:

```
if fid==-1
  error('readdata: Could not open file');
end
```

The next line skips the header in each data file. The Jezzard data files have an 8-byte header:

```
fread(fid, 8, 'char');
```

Next, the data in the file is read into a single vector, V. Remember that this is only one volume at a time. The size of the individual data points must be specified — a 2-byte signed integer in this case. It is probably wise to specify the data size in architecture-independent form, as shown in Table 2.2 (e.g. use "int16" instead of "short"):

```
V = fread(fid, 'int16')';
```

MATLAB	C or Fortran	Description
'char'	'char*1'	character, 8 bits
'uchar'	'unsigned char'	unsigned character, 8 bits
'schar'	'signed char'	signed character, 8 bits
'int8'	'integer*1'	integer, 8 bits.
'int16'	'integer*2'	integer, 16 bits.
'int32'	'integer*4'	integer, 32 bits.
'int64'	'integer*8'	integer, 64 bits
'uint8'	'integer*1'	unsigned integer, 8 bits.
'uint16'	'integer*2'	unsigned integer, 16 bits.
'uint32'	'integer*4'	unsigned integer, 32 bits.
'uint64'	'integer*8'	unsigned integer, 64 bits
'float32'	'real*4'	floating point, 32 bits.
'float64'	'real*8'	floating point, 64 bits.

Table 2.2: Architecture Independent Variables

We now have a single volume of the full data (i.e. a spatial volume for a given time point) in the form of a 1D vector. The next bit re-orders this data so that it will appear in the correct order for `Lyngby` as specified earlier. This is the most complex bit!

```
V = reshape(fliplr(flipud(reshape(V,64,64))),1,64*64);
```

Looking at this function in more detail:

First of all, the vector V is reshaped into a 2D matrix, 64-by-64. Note that this case is relatively easy because the Jezzard data is only single-slice, so the volume is 64-by-64-by-1. (For a multiple-slice dataset, you would have to use the `reshape` function with an extra parameter. Type `help reshape` at the Matlab prompt for more information.)

```
reshape(V,64,64)
```

Next, the matrix is flipped about the horizontal axis:

```
flipud(matrix)
```

and then about the vertical axis:

```
fliplr(matrix)
```

Finally, this re-ordered matrix is put back into the vector V with another `reshape` command:

```
V = reshape(matrix, 1, 64*64)
```

This results in a vector V that has been re-ordered so that the elements appear in the same order that `Lyngby` is expecting them, i.e. X, Y, Z, X, Y, Z, X, etc with the Z indices changing the quickest.

Finally, the file is closed with:

```
fclose(fid);
```

2.4.3.2 The data_paradigm.m and data_run.m files

The `data_paradigm` function for the Jezzard data is:

```
function P = data_paradigm();

  P = [kron(ones(3,1), [zeros(10,1) ; ones(10,1))] ; zeros(4,1)];
```

Another example of a `data_paradigm` function is:

```
function P = data_paradigm

P = lyngby_kronadd(zeros(8,1), [ zeros(24,1) ; ones(24,1) ; zeros(24,1) ]);
```

This function was created for an 8 run study with 72 scans in each run: first are 24 base line scans, then 24 activation scans, and finally 24 base line scans. Note that as well as using this compact way of specifying the paradigm it is also valid to type the paradigm as a long vector, i.e. specify the paradigm in long-hand:

```
function P = data_paradigm

P = [ 0 0 0 1 1 1 0 0 0 ... 0]';
```

The total length of the paradigm vector must be the same as the whole time series, i.e. *before* the time mask is applied. This is also true for the function defining the run structure — `data_run`:

```
function R = data_run();

    % The Jezzard dataset contains three runs: The first with 24 scans,
    % then one with 20, and the last also with 20.

    R = [ones(4,1); ones(20,1) ; 2*ones(20,1) ; 3*ones(20,1)];
```

Another example of a data_run.m file is:

```
function R = data_run

R = lyngby_kronadd((0:7)', ones(72,1));
```

Both examples define a staircase function, with one level per run.

Two of the examples use the convenient `lyngby_kronadd` function, which is actually a "kronecker addition". It works like the kronecker tensor product (the matlab `kron` function), just with addition instead of multiplication.

Alternatively, instead of defining functions you can define `*.mat` files (`data_paradigm.mat` and `data_run.mat`) with a `PARADIGM` and `RUN` variable in them, or you can define an ascii file (`data_paradigm.txt` and `data_run.txt`).

When you start `Lyngby` two functions are automatically called: `lyngby_paradigm` and `lyngby_run`. These will call your `data_paradigm.*` and `data_run.*` function/files and setup the global variables `PARADIGM` and `RUN`.

2.4.3.3 The data_init file

A `data_init` function is created to setup some of the global variables. These variables could also have been setup through the command line. However it is convenient to have them in a file so you will not need to set them up every time.

For the Jezzard data:

```
function data_init();

  lyngby_global;

  NUM_VOXELS      = [ 64 1 64 ]';
  ROI_VOXELS      = [29 46 ; 1 1 ; 14 43];
```

A list of the general global variables is given in Table 2.3 on page 37, and a list of the user-interface related global variables is given in Table 2.4 on page 38.

If another program has masked the volumes so that non-brain voxels are already zero, we will take advantage of this by setting a `VOXEL_MASK`, — reading in the first volume and extracting only the voxels that are different from zero.

2.5 Using the Lyngby Windows Interface

The `Lyngby` toolbox is easy to use due to a graphical user interface (GUI) front-end embedding the numerical routines. The main window that pops-up when starting `Lyngby` is shown in figure 2.1.

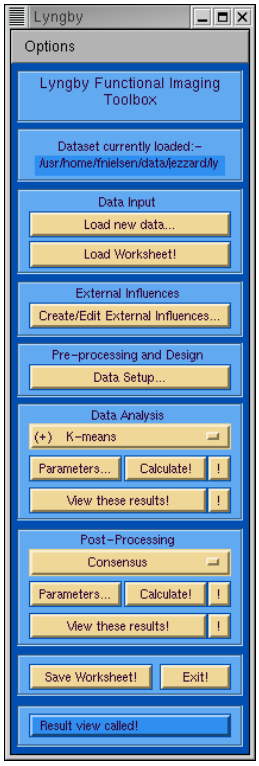


Figure 2.1: The main control window in Lyngby.

2.5.1 The Workflow in the Lyngby Toolbox

As seen from figure 2.1, a top to bottom flow is intended The main window is split into sections (or *frames*) according to their purpose as follows:

Title frame The name of the Toolbox.

Data This shows the name or location of the current dataset being used.

Data Input Data is loaded into the toolbox here

External Influences This is where the *Run* and *Paradigm* variables are loaded in and/or edited.

Pre-processing and Design Any set-up of the data, including removal of unwanted scans or the subtraction of the mean across each time-series, can be performed here.

Data Analysis The heart of the toolbox, this allows the data to be analysed by any of the built-in algorithms and viewing of the subsequent results.

Post-Processing Once a result dataset has been created, further analysis, such as clustering of the results, can then be performed.

Close The worksheet is saved, and the toolbox closed, from here.

Status Feedback on what process is currently running is shown here, and the toolbox copyright infomation can also be obtained by pressing the status line.

Note that we use the following convention for the menus:

- Ok Accept the values currently shown in the display and close the window.
- Apply Accept the values currently shown in the display.
- Cancel Don't accept the values currently shown in the display (note that pressing Cancel after Apply will first accept the current values and then close the window).

2.5.2 A Typical Analysis Session

A typical analysis procedure would progress as follows:

2.5.2.1 Data Loading

The first stage of the process is to get some data into the toolbox. This is done by either loading an existing worksheet from a previous session, or loading new data from scratch. A previously saved worksheet (usually `*.mat`) is recalled by pressing the Load Worksheet! button. This brings up a standard Matlab file chooser, starting from the directory where `lyngby` was started.

New data is loaded by pressing the Load new data... button. This then displays a window called *Load setup*, shown in Fig. 2.2, allowing specification of the number of scans, the file format, voxel masking etc.... To aid this process, there is a Try to setup! button. This attempts to probe the header files for the chosen file format (or the `data_readinfo.m` file in the case of the `Lyngby` format) and place any extracted information about the datafiles into the correct fields in the *Load Setup* window. Any fields successfully read are indicated by the change of the respective status box to *Header*.

There are several options within the *Load Setup* window. The majority of the fields are used to set up the file structure and image size etc..., whilst the bottom right frame is used to specify some external factors, such as specifying a region-of-interest (ROI) and the removal of any unwanted scans.

2.5.2.2 The Load Window – Data Fields

There are many fields in this window, and the various file formats support a different number of these fields. Those that are not supported, or are yet to be fully implemented, are greyed-out and hence cannot be used (note how the accessibility of the fields changes as you change the file format).

File Format: `lyngby` can load in several different file formats. Apart from its own native format, `lyngby` can also read Vapet, Analyze Stimulate and Raw formats. The `lyngby` format requires some additional small conversion files to be written that describe how the data should be read (examples were given earlier in Subsection 2.4.3 on how to create your own). Once this has been done for your dataset, the `lyngby` format then becomes the easiest to load, as the image parameters have already been specified.

The other formats will require the user to specify the location of the data, the image size, the size of each data unit, the ordering of the data etc.... It is usual for the data to be split across many individual files, and this is catered for by the *File Pattern*, *Start Index* and *No. of Scans* choices, which allows the user to specify the common file stem and the scope of the file index.

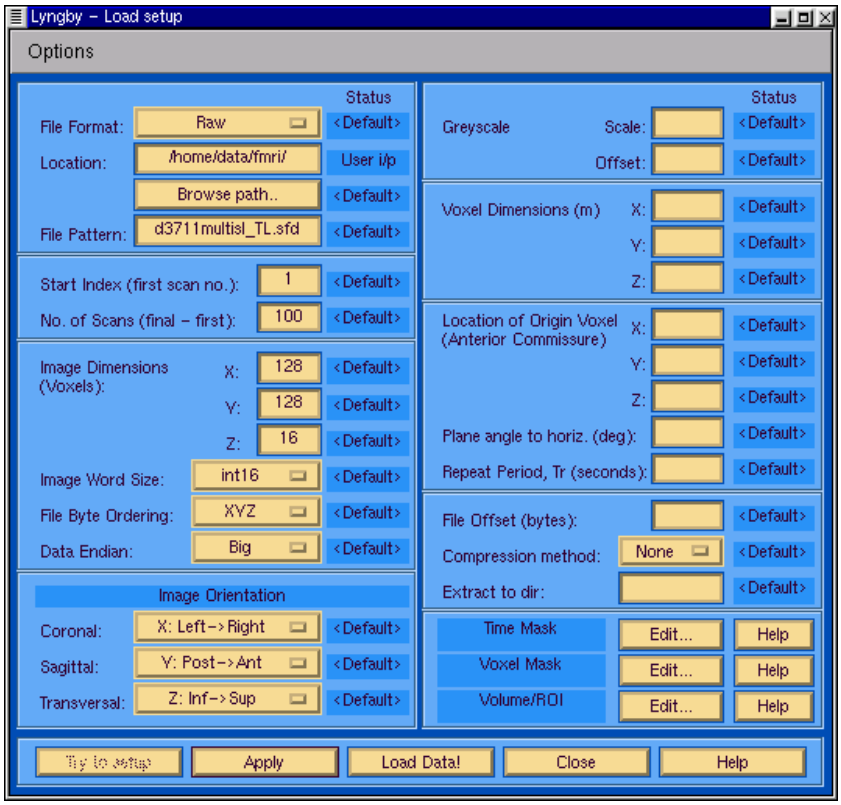


Figure 2.2: The file load window in Lyngby.

Image Dimensions: Specifies the size of the full data volume, *in spatial terms* i.e. for a single time-point, before any region-of- interest is selected.

Image Word Size: The size of the individual words in the datafile.

File Byte Ordering: The order of the bytes within the file.

Data Endian: Intel (Pentium) and DEC Alpha usually use big endian, while Sun and SGI usually use little endian when storing data on disk.

Data Orientation: allows the user to change the orientation of the displayed image. Note that this refers to the mirroring around each of the three orthogonal axes, and not to the way the data is stored within the file — that is catered for by the *Ordering* option.

Greyscale: How the greyscales are mapped to the colourmap.

Voxel Dimensions: Specifies the real-world size of the image voxels, to allow for calibration etc. . .

Location of Origin Voxel: Allows spatial calibration of the brain.

Plane angle to horizontal: Compensates for non-orthogonal orientation of the slice-planes.

Repeat period: The frame-rate – allows the calculations to be calibrated in seconds instead of in frames.

File Offset: The number of bytes to skip in the datafile in order to get past the header info and into the data. Very useful for loading in raw data without creating any seperate translation files.

Compression method: The file compression used on the datafiles. The toolbox is capable of decompressing the files 'on-the-fly', allowing the data to be kept compressed and thus saving space. It also allows the data to be read compressed straight from a CDROM.

Extract to dir: The temporary directory used for on-the-fly decompressing of the data as it is loaded-in.

2.5.2.3 The Load Window – Bottom-Right Frame

This frame controls some of the external factors that affect the loading-in of the data. Other external influences, such as the *Run* and *Paradigm* variables, don't affect the way the data is loaded-in and as such are seperated off into a later stage.

Time Mask: Before actually loading the data, a *time mask* can be specified by pressing the Edit... button. This is convenient if you want to discard transient scans or scans at the start that behave badly (such as highly saturated T1 scans). The associated global variable is `TIME_MASK`, and the window is shown in figure 2.3.

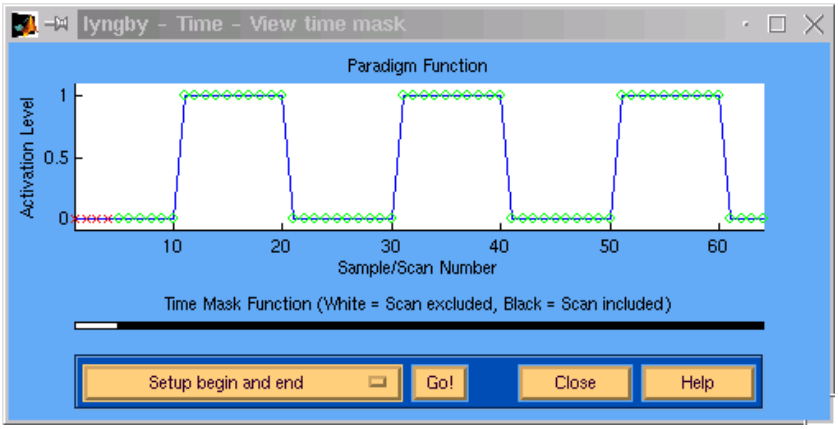


Figure 2.3: The time window in `Lyngby`.

Voxel Mask: This allows the exclusion of specific voxels within the selected region of interest (ROI) to be loaded. It is usually in the form of a sparse matrix. At the moment, the front-end interface for this feature is not yet finished, but the supporting code is in place and as such you may load in a voxel mask from the command prompt. Details of how to do this are given later.

Volume/ROI: The ROI can be specified by pressing the Volume/ROI Edit... button. In order to edit the volume, it must first, of course, be loaded in. This requires the data parameters to be specified first, so once this is done, press the Apply button at the bottom and this will then unshade the Volume/ROI Edit... button. Note that the whole data matrix is not loaded at this point — only the relevant volume. In this window, a rectangular ROI can be specified by the user, who can also browse through all the data in

both time and space. As can be seen in Fig. 2.4, the user has the choice of five different viewing layouts to examine the data. The first one simply shows a single slice of the data, with the choice of the three orthogonal directions and the ability to move along each of these axes. The second choice allows two slices to be shown alongside one another, either from the same or different viewpoints. There is also the ability for linking the two views if they are of the same direction, allowing for easy-stepping through the volume. The third choice is the standard triple orthogonal view. This is also repeated in the fourth option, alongside a 3-dimensional view of the data. The final option is for view multiple slices from the same viewpoint, displaying 8 adjacent slices.

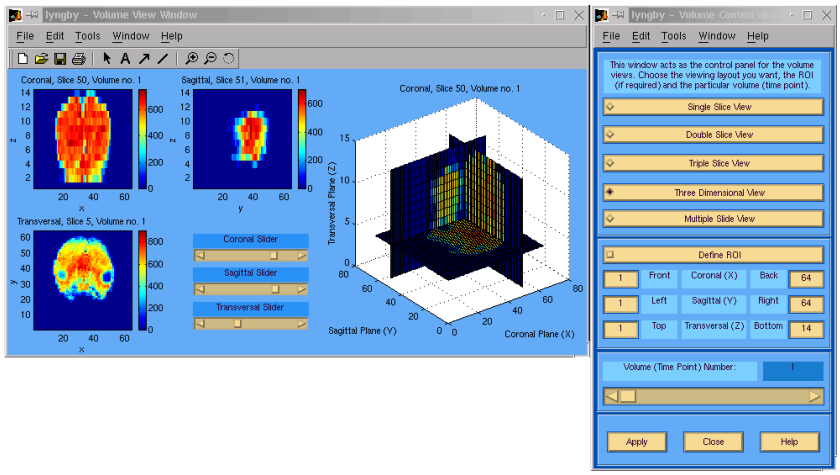


Figure 2.4: The volume window in Lyngby, showing three orthogonal views and a 3-dimensional one.

Once all the loading parameters have been set, press the Apply button to ensure that they are all stored, and then press the Load Data! button to start the loading-in of all the data. The *Load Setup* window is closed and the status bar at the bottom of the main window indicates the progress of the loading.

2.5.2.4 Data – External Influences

In the main window, there were links to create, edit and load external influences on the data that affected the way or the amount of data loaded into the toolbox. This section covers the external influences on the data that are applied *once the data is loaded*. To access this stage, press the Create/Edit External Influences... button in the main window. This will bring up the *External Data Influences* window, as seen in figure 2.5. Most of these external influences are usually specified in external files first, and then altered here if necessary. Guidelines for writing these files are given later:

- The run structure can be verified by pressing the Create of edit the Run function... button. Each of the runs is given a unique number so the graph should be a staircase as shown in figure 2.6. You will only need this structure if you plan to make a pre-processing step or analysis type that requires run information, e.g. run centering. The global variable associated with the run structure is called `RUN`. Like the paradigm, this function can be altered here, but is usually defined in an external file.

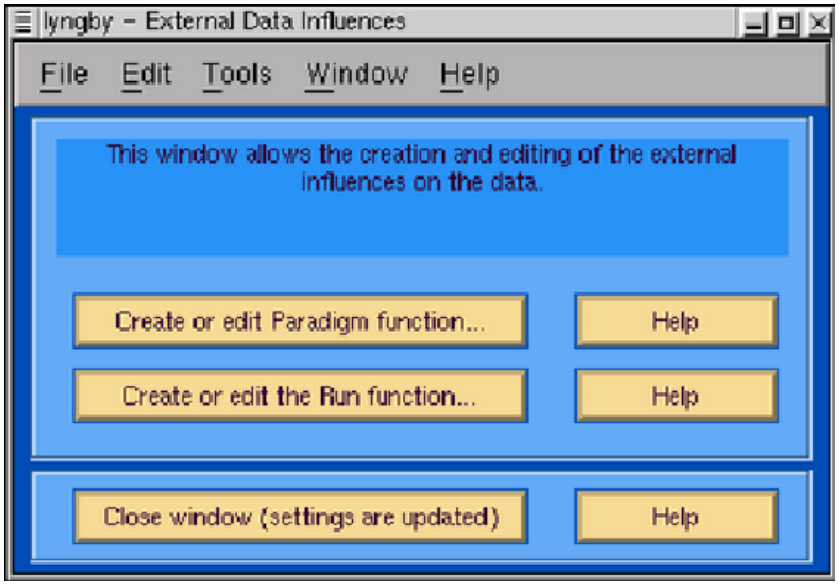


Figure 2.5: The External influences window in Lyngby.

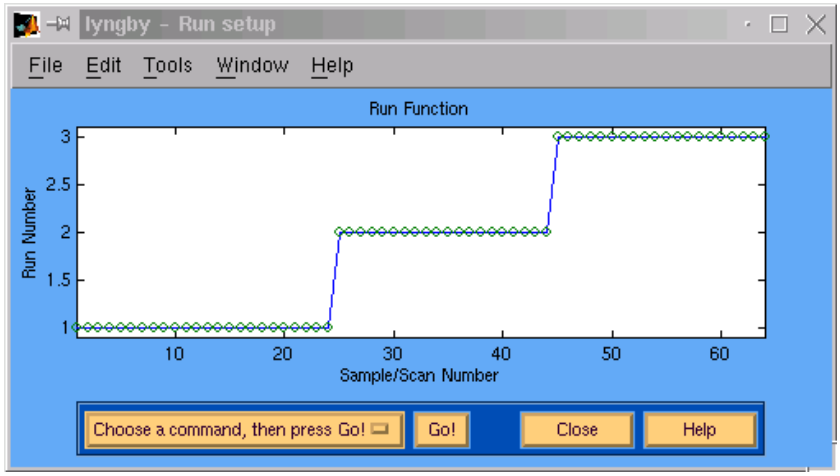


Figure 2.6: The run window in Lyngby.

- The paradigm can be verified by pressing the Create or edit the Paradigm function... button. The global Matlab variable associated with this window is called `PARADIGM`. The GUI allows alteration of the paradigm function that is usually defined in an external file (`data_paradigm` located at the same place as the data. The *Paradigm* window is shown in Fig. 2.7.

2.5.2.5 Data Pre-Processing

Before the actual modelling, the data should be pre-processed. By pressing the Data Setup... button, the window shown in Fig. 2.8 pops up. This window has a left and right pane which serve different purposes. The user should do any work on the left pane first, before selecting the

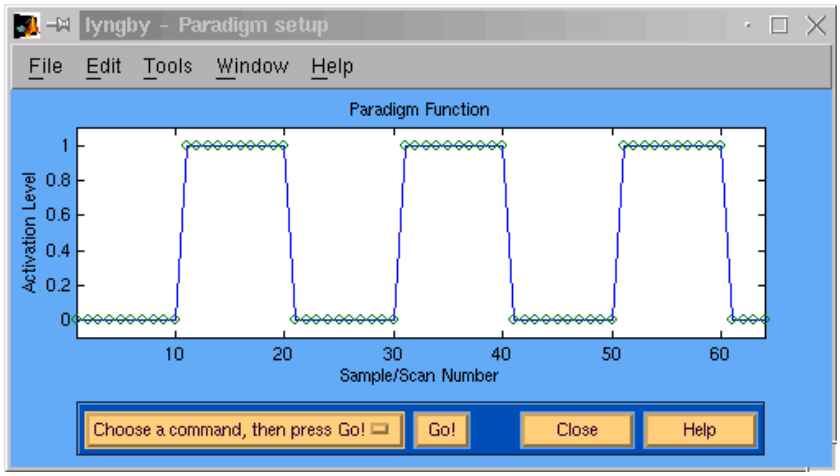


Figure 2.7: The paradigm window in Lyngby.

relevant options on the right pane and then pressing the Apply.. button.

On the left, the top button is used to remove unwanted scans by applying the `TIME MASK` variable, which was previously specified in the *Load Setup* window, to the `PARADIGM` and `RUN` variables. The lower button removes the vertical offset ("DC component") from the `PARADIGM` variable. This is necessary for those analysis algorithms which require a zero-mean paradigm.

The right pane focuses only on the data. Three of the choices involve the removal of the mean from the data: over the whole time series, the whole volume and over each run respectively. The remaining choice normalizes the variance. The relevant choices should be selected by pressing the toggle buttons, and then these are applied by pressing the Apply... button at the bottom of the window. This also closes the pre-processing window.

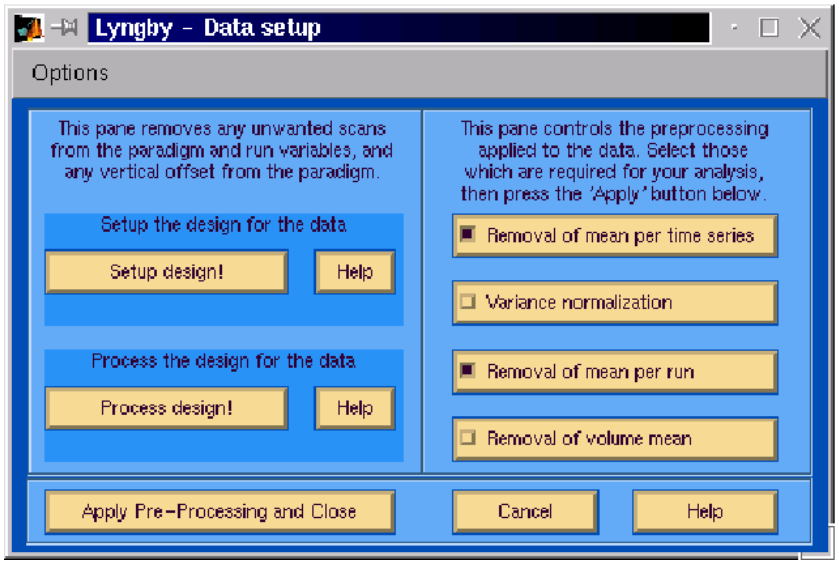


Figure 2.8: The preprocessing window in Lyngby.

2.5.2.6 Data Analysis and Modelling

The next step is to select the actual data modelling algorithm from the *Data analysis* frame. The top button in this frame determines the analysis algorithm that will be used. Pressing this button causes a list to pop-up of all the available algorithms. Simply select the desired one, and the list will disappear and the button will display the name of the chosen algorithm. The *Original* option simply displays the un-analysed data (which is also the input data used by all the other analysis algorithms). The actions of the associated buttons are then dependent on the choice of algorithm, but always follow a standard pattern:

- The Parameters button is used to choose the parameters for the selected modelling algorithm, and as such the window that pops up after pressing it will be different for each one. For the Neural Network Saliency algorithm, the parameter window is shown in Fig. 2.9. Once the required parameters have been chosen, then click Apply, followed by Close.

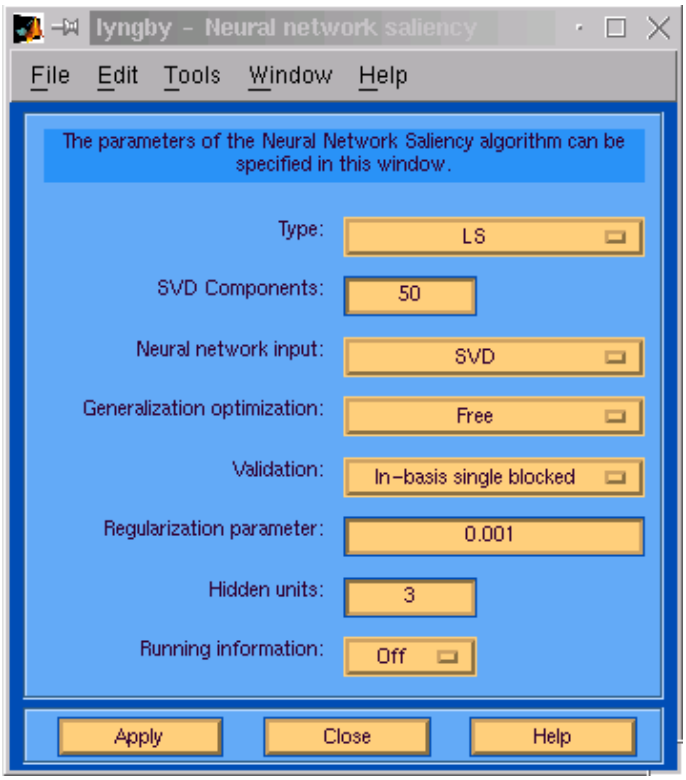


Figure 2.9: The Neural Network Saliency parameter window in Lyngby.

- The Calculate! button will then run the algorithm and the progress of the processing is shown on the status line at the bottom of the main window.
- Finally, pressing the View button shows the result of the analysis.
- The two ! buttons adjacent to the View and Calculate! buttons will display, in the command window, the actual code/script that will be run when their neighbouring buttons are pressed. This allows the user to easily keep track of what variables will be

modified and displayed at each stage.

Note that the popup menu showing the algorithms will have a (+), (-) or a (!) alongside each one, indicating whether or not the results of that algorithm are available for visualization :– (-) indicates no result, (+) indicates a new result, and (!) indicates that the parameters have been changed and the algorithm has not yet been run with these new variables.

The next step is the actual viewing of the modelling results. Whichever modelling or analysis method is chosen, the same three windows are used to explore the results. For the Neural Network Saliency algorithm, these result windows are shown in Fig. 2.10.

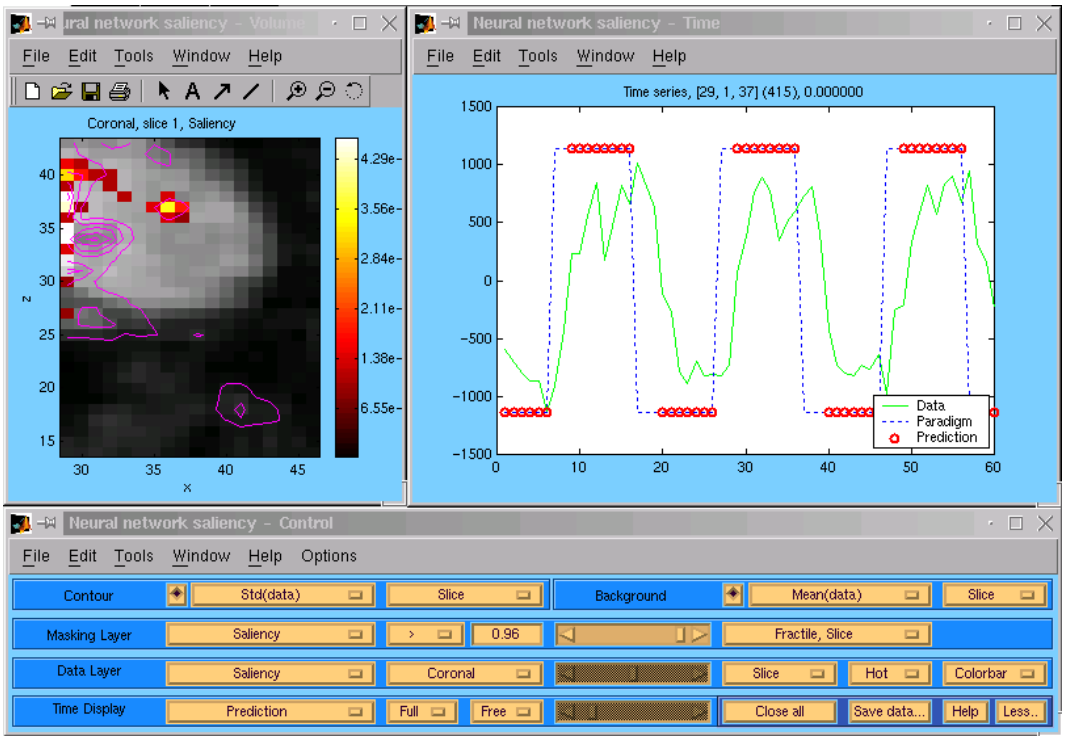


Figure 2.10: The Neural Network Saliency results window in Lyngby.

Let's go through the features of these windows in more detail:

- There is always one control window underneath two data-viewing windows, which usually show spatial/voxel information and time series information respectively. The control window is arranged in *layers* to reflect the concept of overlapping data layers in the spatial window. This is perhaps illustrated more clearly with the aid of a diagram — see figure 2.11.
- Clicking on a voxel in the volume window will automatically update the other window to show the time-series for the chosen voxel, plus the result of any modelling algorithm for that particular voxel.

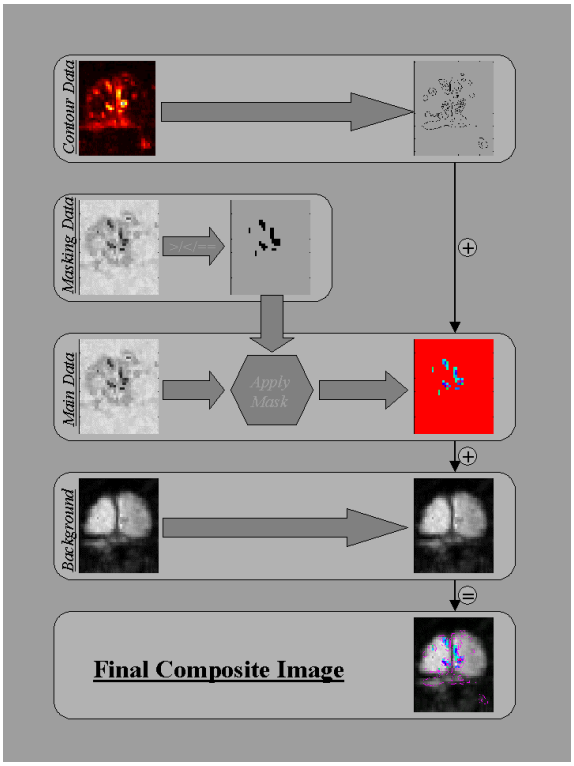


Figure 2.11: The Concept of Layers in the Result Windows.

- The control window can be expanded to show extra layers with the [More...] button. These allow the addition of a contour overlay, a background underlay, and data-masking features.

  Time Display Layer: The lower layer controls the right window, which usually displays the time-series of the currently selected voxel. However, some algorithms also produce histograms, error curves, periodograms etc..., either voxel-based or full-volume-based, which can be selected from the left-most button in this layer and displayed in the right window.

  The [Full] button controls the scaling of the bottom axis, and so allows "temporal zooming".

  The next button switches the vertical axis scaling between fixed and automatic.

  Adjacent to this, the slider controls the temporal location of the time-series display, and is used in conjunction with the horizontal zoom button to move through the time-series. It is disabled when the full time-series is displayed.

  The next four buttons apply to all three windows. The [Close all] button closes all three windows. The results are still kept, though, and can be accessed again by pressing the [View these results] button in the main window. The [Save data...] button displays the *Save Layer*, allowing specific volumes or results to be selected for saving. This is covered again later. The [Help] button brings up help for all three windows, and the [More...] or [Less...] button adds/removes the extra layers for masking, background display etc...

Data Layer: This controls the main results of the modelling algorithm. As in the other three voxel-related layers (*Masking*, *Contour* and *Background*), the result set being used is chosen from the leftmost button's popup list. Some algorithms only have a few result sets, whilst others may have dozens. The beauty of this layering approach is that it allows you to display one result set, filtered by another, over a background of a third, whilst overlaying a contour plot of a fourth.

The next button specifies the slice direction for observing the data volume. This is only really interesting if the data is a 3D spatial volume. The adjacent slider controls the position of the slice plane within the volume. Note that these also alter the viewing direction of the other layers as well (it is no use plotting the transversal activation over a sagittal background!)

Next to this is a button controlling what each voxel represents. There is a choice of the current slice, as selected by the previous two buttons, the mean of the volume, or the maximum voxel value, both measured along the current direction.

Next comes a button specifying the colourmap to be used for this layer. The *Default* setting uses the default colourmap as set in the file `lyngby_ui_option.m`.

Finally, the last button controls whether the colourbar on the spatial plot should be displayed or not.

Masking Layer: This layer creates a binary mask from the selected dataset, and then applies it to the Data Layer below. Thus the only parts of the result set that are visible are those that occur where the mask allows. This is useful for viewing a thresholded activation result over a background image. This masking process is only activated when the second button, initially labeled *None*, is switched to some other value.

Next to the first button, which chooses which of the result sets to use to create the mask, is the button which selects the threshold type. The choice is between '¿', '¿¡' (absolute) or '==' (integer equals). The first is a simple threshold. The second is an absolute threshold to remove both positive and negative values greater than a certain value, and the third is used only with a few result sets to pick out a particular level (for instance, in the K-Means clustering, you may want to extract the results of just the 3rd cluster). The next two buttons are used to select the threshold level, either via typing it into the edit box, or by moving the slider.

The final button controls whether the threshold level is an absolute one, or is done on a percentage bssis within the current slice.

Contour Layer This is the left side of the top layer, and controls the addition of a contour overlay on top of the current volume view. It is turned on and off via the radio button to the left. Next to this is the popup button allowing the selection of the result set to use for drawing the contours. The adjacent button specifies whether the contour is drawn using the current slice of the chosen result set, or using its mean or maximum values along the current direction.

Background Layer This is the right side of the top layer, and it controls the 'underlay' of a background image. At the moment, this background image must be chosen from the current result set. We are hoping to incorporate the option of loading in a seperate anatomical background image soon (although this raises several questions regarding registration and resolution). For the moment though, a very good approximation to an anatomical image

can be obtained by use of the mean image, which is automatically included in all result sets.

Once again, the radio button to the left controls the activation of the layer, and the next button allows the choice of which result set to use as a background. The final button specifies whether the background is taken as the values in the current slice, or as a mean or maximum of the volume along the current direction.

Save Layer: This appears on top of the other four layers and has a red background to highlight it. In addition, the layers upon which the save layer is acting are also coloured red. The first button controls what is to be saved e.g. the whole volume, the current slice or the time-series of the current voxel.

The next button specifies the format in which to save the data. The options depend on what is to be saved. For instance, the volume has a choice of ASCII, Matlab, Analyze, STD or VAPET. More choices will be added in future, as will more flexible saving options. These will be located in the next button which is currently unavailable.

The next button allows choice of a compression algorithm. This currently only works on Unix-type systems, as it employs the unix-compress and gzip routines.

The filename to save the file under can be typed into the next box, and saved in the current working directory by pressing the adjacent Save here! button. If you want to choose a different directory, simply use the Save in dir... button instead.

Mosaic View: Originally, most fMRI studies were single slice, but as the amount of volume studies has grown, we have added in a mosaic slice viewer. This allows the display of mutliple adjacent slices of a volume so the whole result set can be viewed at once. To turn this on, select the *Toggle Mosaic View* from the *View Options* menu in the volume window. Alternatively, use the *Control+M* keyboard shortcut when the volume window is the selected window.

An example of the mosaic view can be seen in figure 2.12.

The mosaic view requires a minimum of 3 slices of the current view plane. In addition, it won't work if you are looking at the mean or maximum result values (as this would lead to multiple copies of the same image). The current slice is used as the first slice in the mosaic list, and so there must be at least two more slices left after it for the mosaic view to work. So if you are currently looking at the last slice of a 16 slice volume, move the slider to the left to select an earlier slice before choosing the mosaic view.

Once selected, the mosaic view will display up to 20 slices from the current volume. You can move the slice slider as before to move through the volume. The result sets all use the same colourmap (i.e. the colourmap has been scaled to the maximum and minimum of the current result set) so individual colourbars are not necessary, and the slices update far quicker wiith them turned off. A single colourbar for this view will be added very shortly.

You can turn the mosaic view off again with the same menu or keyboard shortcut that turned it on.

2.5.2.7 Data Post-Processing

After the data analysis, the user can perform some post-processing on the results data. This post-processing allows the analysis of the previous result sets in a formal way. At the moment,

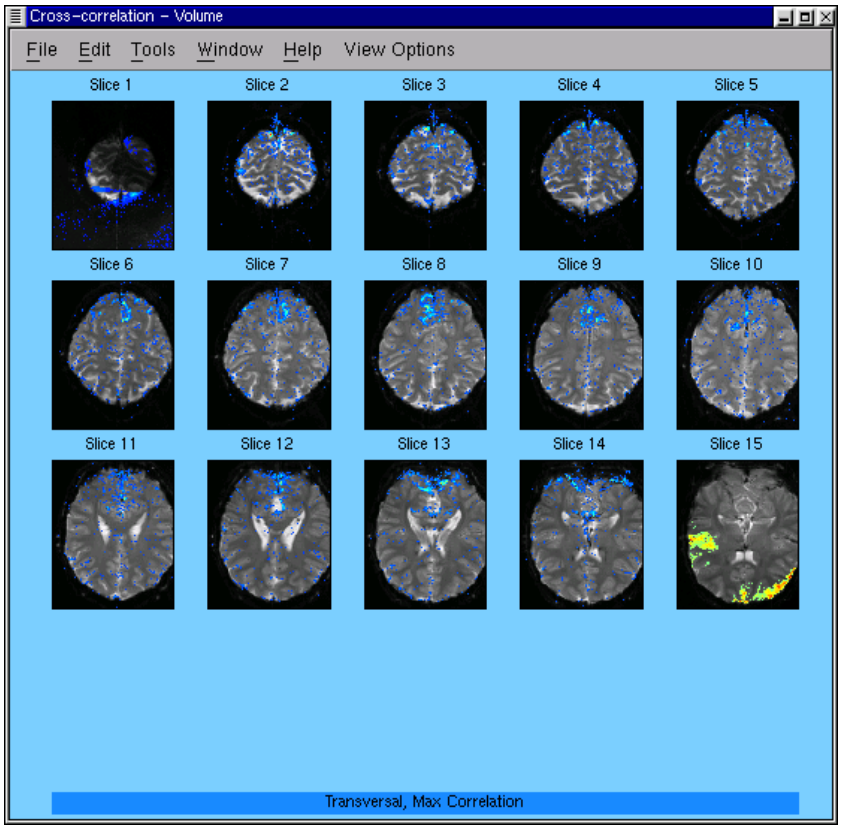


Figure 2.12: The volume result window in mosaic mode.

only one post-processing algorithm, Meta K-means clustering, is included in the toolbox, but more are due to be added later. To use this feature, one first has to have a collection of result parameters that require analysis. This *result dataset* is presently acquired by using either the FIR Filter or the Iterative Lange Zeger routines. Once this has been created, then the post-processing can be done. The procedure is exactly the same as for the main analysis routines, with the user setting the initial parameters of the algorithm via the Parameters... button, shown in Fig. 2.13 , and then starting the calculation by pressing the Calculate! button. The results are also viewed using the same interface as before, accessed via the View these results! button.

The Meta K-Means algorithm attempts to cluster, for example, the *parameters* of the filters used to model each of the voxels, in effect looking for commonality of *filter types* instead of common voxel time-series. In this way, it is a 'higher level' of analysis, and should provide another viewpoint on the data.

2.5.3 Exporting Variables, Printing and Saving Results

Generally, the result variables are not available from the commandline as they are global variables. However, it is easy to access them by issuing the command:

```
>> lyngby_ui_global
```

You can then list all the variables by doing:

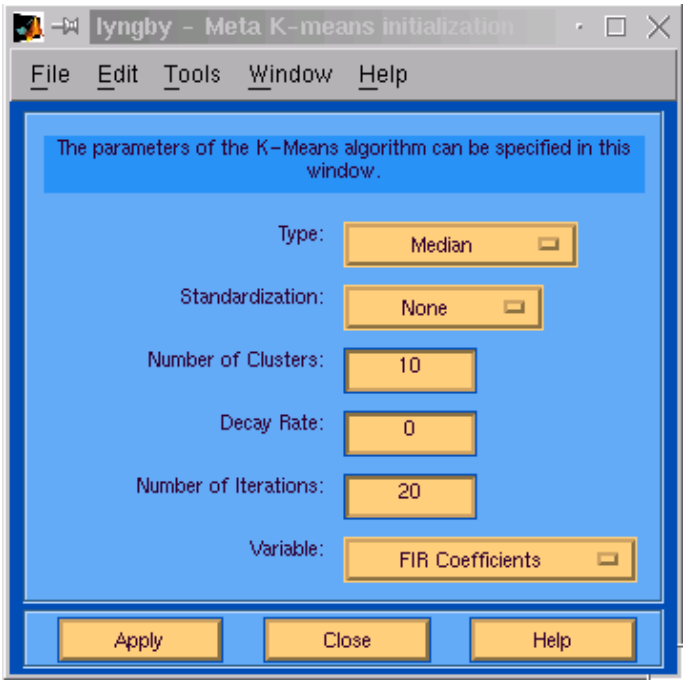


Figure 2.13: The parameter options for the Meta K-Means post-processing algorithm.

```
>> whos
```

This allows you to both see the size of the result variables and save them directly from the command line. For example:

```
>> save MyFilename.txt RESULT_LZ -ascii
```

This will export the Lange-Zeger result to an ASCII-file named MyFilename.txt. Note that you might want to change to another directory first – you are in the data directory while working with the `Lyngby` package.

Alternatively, you can save the entire worksheet by using the Save Worksheet button near the bottom of the main window. This will bring up a file chooser, allowing you to store all the data, the variables and all the results obtained into a single file, usually of the form `*.mat`.

To print a window of the lyngby toolbox in Matlab 5.0, you can use the print function in the window frame or you can use the ordinary matlab function `print`:

```
>> print -deps MyFilename.eps
```

An alternative is to save the data from the *Result View - Control* window. To specify which data to save, click on the Save data... button towards the bottom right of the window. This will expand the window to five layers - the normal four layers and an extra red-framed "Save" layer. Changing the choices in the first button on the Save layer specifies which layers, and hence which data, is to be saved. These layers are also highlighted in red. The other buttons in the Save layer allow the choice of the Save options - the file format, file compression, filename and file location.

2.6 Using Lyngby from the commandline

You may prefer to use Lyngby from the command-line instead of using the window-interface. As the GUI simply calls individual Matlab *.m files, then these can also be called directly from the command-line. However, if you are working without the GUI you must setup the parameters in the data_init.m file. In addition, it may help to have the run and paradigm files setup as for the GUI. If your data is a non-supported format, then it may also be easier to put the data-loading functions into the data_readdata.m file rather than having to enter them on the commandline each time.

2.6.1 Data Loading

- Change to the relevant data directory
- Setup the global variables. This is usually done within data_init.m, data_paradigm.m, and data_readdata.m
- X = lyngby_getdata;

2.6.2 Pre-Processing

Pre-processing has been used with different meanings in neuroimaging. In terms of the Lyngby toolbox, by pre-processing we mean the step after loading the data, but before the actual data-analysis.

In the GUI, the setup of the pre-processing is managed by lyngby_ui_preproc. This is called from lyngby_ui_main upon pressing the Data Setup ... button. The function and its associated window are used to setup the global variables in lyngby_prep_global. These variables are used by lyngby_ui_main in its call to lyngby_normalize — the function that does the actual pre-processing. This is normally done by pressing the Apply Pre-Processing and Close button in the *Data Setup* window.

The loaded data, held in the global variable X, is passed to the pre-processing step, which then outputs the global variables XN, X_MEAN and X_STD. You do not have to use the lyngby_normalize function, and you can set these variable yourself directly from the command-line:

```
XN = X;
X_MEAN = mean(X);
X_STD = std(X);
```

2.6.3 Global Variables

The Lyngby toolbox has to use a set of global variables that under normal conditions are hidden from the user. The user can make the these variable available to the workspace by calling lyngby_global. A list of the Lyngby global variables is given in table 2.3.

Furthermore, the result variables and the data matrix are also global, but are seperate from the ones shown above. You can make the these variables available to the workspace by calling lyngby_ui_global. An example list of these global variables is given in table 2.4.

X and P will be defined once the data have been loaded (normally done upon pressing the Load Data button). XN and PN will be defined once the pre-processing has been done, and the RESULT_* will first be defined once an analysis has been performed (via the Calculate! button).

The next section gives more details on how to run the analysis algorithms from the commandline.

Name	Description
DATATYPE	Type of data, e.g. 'float', 'double', 'short', 'byte', 'long'
DEFAULT_SAVE_PATH	The default location in the file chooser when saving data
DISCRIM_TMASK	
FILENAME_PATH	Path to the data file
FILENAME_PATTERN	String with the filename pattern. Example: 'volume%02d'
FILENAME_STARTINDEX	Start index for use with the FILENAME_PATTERN
FILENAME_WORKSPACE	The name of the workspace loaded in (lyngby_workspace.mat is the default)
FILE_READING_TYPE	1=Analyze/Vapet (obsolete), 2=Lyngby (i.e. custom), 3=Raw, 4=Analyze, 5=Analyze4D, 6=SDT (Stimulate), 7=VAPET4D
LOGFILENAME	File name to write log information into - not currently used
NUM_RUNS	Number of runs in the data matrix
NUM_SCANS	Number of scans in the data matrix
NUM_SUBJECTS	Number of subjects in data - not used at the moment
NUM_VOXELS	Number of voxels in the datamatrix. 3x1 vector
ORDERING	The file order of the voxels:'xyz', 'yxz', etc...
ORIENTATION	How the data is mirrored to display it the correct way around ['lr', 'pa', 'is']
ORIGIN	Centre voxel
ROI_VOXELS	Number of voxels in the (rectangular) ROI. 3x1 vector
TIME_MASK	Mask in time to discard unwanted scans
UI_ON	If UI_ON=1 then the program is run from the GUI interface, if 0 then text based (not up-to-date)
VOXELSIZE	Voxelsize
VOXEL_MASK	Spatial mask, e.g. to remove non-brain voxels

Table 2.3: Global variables

2.7 Writing Scripts to Run Lyngby Automatically

The functions in Lyngby do not need to be run from the GUI and can easily be run from the commandline, although the structure and ordering in which everything is done is not as obvious. This is the advantage of using the GUI. But the commandline approach does have an advantage also, and that is the ability to write and run "scripts", equivalent to batch files. This means you can load your datafiles, perform the pre-processing, analyse and model the data and do any post-processing, all without any user intervention. Obviously this allows you to set-up a whole selection of jobs running overnight on different data, using different modelling parameters, and

Name	Description
`X`	The original datamatrix
`XN`	The preprocessed datamatrix
`RUN`	The original run series
`PARADIGM`	The original paradigm series
`P`	The paradigm variable after the time mask has been applied
`R`	The run variable after the time mask has been applied
`PN`	The preprocessed paradigm (zero-mean)
`X_MEAN`	Mean of each scan
`X_STD`	Standard deviation of each scan
`DELAY_*`	The results from algorithms that include a measurement of temporal shift between the paradigm and the voxel's response
`RESULT_*`	The main results from the different algorithms. See below for examples.
`STRENGTH_*`	The results from algorithms that include a measurement of the strength of the voxel's response
`RESULT_FIR`	The result from the FIR analysis
`RESULT_BAT`	The result from the Ardekani t-test
`RESULT_BAF`	The result from the Ardekani f-test
`RESULT_LM_ASSIGN`	The assignment labels from the K-means analysis
`RESULT_KM_CENTER`	The cluster center in K-means analysis
`RESULT_CC`	The result from the Cross-correlation analysis
`RESULT_TS`	The result from the ordinary t-test (t-measure)
`TS_PROBMAP`	The result from the ordinary t-test (probability map)
`RESULT_LZ`	The result from the Lange-Zeger analysis
`KS_PROBMAP`	The result from the Kolmogorov Smirnov test (probability map)
`RESULT_KS`	The result from the Kolmogorov Smirnov test analysis

Table 2.4: Some Examples of the Global GUI variables

then look at the results the next morning.

It is very straightforward to write scripts for `Lyngby`, although it will be easier once you are familiar with using the GUI. You will then know the order in which the data is processed and the range of possible choices (e.g. the different pre-processing options) and algorithms available. A full list of all the functions within the `Lyngby` toolbox and their purpose is given in Table B.1 of Appendix B. In addition, you may want to have a look at the `Lyngby`website, where there is ahyperlinked index of all the toolbox files, giving details of its purpose and its interdependencies with the other files.

When using the commandline, you must have a `data_init.m` file to specify the initial data parameters, as well as the `data_run.m`, `data_paradigm.m` and `data_readdata.m` files. These are set up in exactly the same way as would be required when using the GUI (see Section 2.4.2 for details and examples).

For example, for a particular dataset, the initialisation and conversion files would be as shown below:

The data_init.m file:

```
% data_init.m
function data_init

lyngby_global

NUM_VOXELS = [24 12 1];
NUM_SCANS = 384;
FILE_READING_TYPE = 2;

DISCRIM_TMASK = lyngby_index2tmask(...
   lyngby_dropedge(TIME_MASK*lyngby_paradigm, 4, 4), ...
       length(TIME_MASK*lyngby_paradigm));
```

The data_paradigm.m file:

```
% data_paradigm.m
function P = data_paradigm

P = kron(ones(8,1), [zeros(12,1) ; ones(24,1) ; zeros(12,1)]);
```

The data_run.m file:

```
% data_run.m
function R = data_run

R = kron( (1:8)', ones(48,1));
```

The data_readdata.m file:

```
% data_readdata.m
function V = data_readdata(index)

fid = fopen('../data/simfmri.1', 'r');
X = fscanf(fid, '%f');
fclose(fid);
offset = (index-1)*288;
V = X( (1:288)+offset );
```

Then the main script file may look something like this:

```
% session_load          Load data
    lyngby_init
    lyngby_ui_global

    % Set up X
    data_readx

    % Set up design
```

```
    P = lyngby_paradigm;
    R = lyngby_run;

% session_prep          Compute data pre-processing
    lyngby_ui_global

    % Preprocessing
    lyngby_prep_global
    PREP_CENTERING = 1;
    PREP_RUNCENTERING = 0;
    PREP_IMAGECENTERING = 0;
    PREP_NORMALIZATION = 0;

    [XN, X_MEAN, X_STD, X_SEQMEAN, X_SEQSTD] = ...
        lyngby_normalize(X, ...
        'Centering', PREP_CENTERING, ...
        'RunCentering', PREP_RUNCENTERING, ...
        'ImageCentering', PREP_IMAGECENTERING, ...
        'Normalization', PREP_NORMALIZATION);

    PN = P - mean(P);

% session_lz           Compute Lange-Zeger

    lyngby_lzit_global
    LZ2_THETA1INIT = 10;
    LZ2_THETA2INIT = 2;
    LZ2_STEPSIZE = 1;
    LZ2_MINCHANGE = 1e-4;
    LZ2_ITERATIONS = 90;

    RESULT_LZIT = lyngby_lzit_main(PN, XN, ...
        'Iterations',LZ2_ITERATIONS,...
        'MinChange',LZ2_MINCHANGE,...
        'StepSize',LZ2_STEPSIZE,...
        'Theta1Init',LZ2_THETA1INIT,...
        'Theta2Init',LZ2_THETA2INIT...
        );
    RESULT = RESULT_LZIT;
    STRENGTH_LZIT = RESULT_LZIT(1,:);
    DELAY_LZIT = RESULT_LZIT(2,:) ./ RESULT_LZIT(3,:);

% session_save_lz      Save LZ results

    [y, m, d] = datevec(now);
    s = sprintf('%d-%02d-%02d', y, m, d');

    t = pwd;
```

```
t = t(length(t));

eval(sprintf('save RESULT_LZ_BETA_%s_%s.txt STRENGTH_LZIT -ASCII', s, ...
    t));

eval(sprintf('save RESULT_LZ_DELAY_%s_%s.txt DELAY_LZIT -ASCII', s, ...
    t));
```

2.8 Adding New Algorithms to the Toolbox

The `Lyngby` toolbox is meant as a development platform as well as an analysis one, and as such we encourage you to add your own functions and models. If you have any you think would benefit other researchers we would be happy consider it for the next release of the toolbox.

The process of adding your own functions is straightforward and you need only follow the steps outlined below.

1. In `lyngby_ui_main.m` : Add the new main algorithm '.m' file to the list in the header under

   `(See also:)`

2. Add the variables returned from the new method to the ones in `lyngby_ui_global.m` - often `RESULT_{NAME}` where `{NAME}` is the identifier for the new method. Both in the help text and the global settings.

3. In `lyngby_ui_main.m` : Add a line under the line

   `% Globals`

   with the name `lyngby_{NAME}_global`

4. Add the file `lyngby_{NAME}_global.m` where the control variables of the new algorithm are made global. (See `lyngby_fir_global.m` as an example). The control variables should be named `{NAME}_{SOMETHING}`, where `{SOMETHING}` describes the variable contents.

5. In `lyngby_ui_main.m` under the line

   `% Method Identifiers`

   add an identifier for the new method `am_{NAME} = {NEXT#}` where `{NEXT#}` is the next available number. (It is ok to insert and shift the remaining.)

6. In `lyngby_ui_main.m` under the line

   `% Initialize`

   add the line

   ```
   lyngby_{NAME}_init;
   ```

7. Add the file `lyngby_{NAME}_init.m` where the variables used in `lyngby_{NAME}_global.m` are initialized to suitable values.

8. In `lyngby_ui_main.m` add a new line under the line

   ```
   % Analysis Buttons
   ```

   which should look like

   ```
   '(-)    {EXPLAINING TEXT}|',...
   ```

   Note that the order should match the selection from 5:

9. In `lyngby_ui_main.m` locate the line

   ```
   elseif command == 500
   ```

   and add two lines

   ```
   elseif AnalysisMethod == am{NAME}
     lyngby_log('{EXPLAINING TEXT} chosen');
   ```

10. Locate the line

    ```
    elseif command == 501
    ```

    add two lines

    ```
    elseif AnalysisMethod == am{NAME}
      lyngby_ui_{NAME}_init;
    ```

11. Add the file `lyngby_ui_{NAME}_init.m`. Here a GUI interface to setting the parameters from 7: is made. Look at the other `lyngby_ui_{OTHERNAME}_init.m` for examples.

12. In `lyngby_ui_main.m` locate the line

    ```
    % The actual analysis
    ```

    Add a new block for the new algorithm, which maybe look like

    ```
    elseif AnalysisMethod == am{NAME}
      RESULT_{NAME} = lyngby_{NAME}_test(XN, PN, R, ...
                          'Something', {NAME}_{SOMETHING});
      RESULT = RESULT_{NAME};
    ```

    `XN` is the normalized data matrix at this point, `PN` is the activation function (with mean subtracted), and `R` is the run structure.

13. In `lyngby_ui_main.m` locate the line

    ```
    elseif command == 503
    ```

    Add a new block for visualization of the new method

14. Add the file `lyngby_{NAME}_test.m` which returns the result of the method. The file `lyngby_ui_main.m` will turn the `RESULT` into activation strength, delay etc.

Chapter 3

Data Formats, Masking, and Pre-Processing

$Revision: 1.11 $
$Date: 2002/08/14 14:21:24 $

3.1 Introduction

These next three chapters are reference-texts, covering the different stages of data-input, analysis and post-processing in more detail than in the previous "user manual".

This chapter deals with the issues of data formats and preparation before the analysis stage. The next chapter describes the different analysis algorithms available within the toolbox, whilst the one after it covers the post-processing, or meta-analysis, algorithms.

Each of the modelling algorithms needs to do several processing steps. Some of them are common and have been pulled into a common pre-processing step. The most common step is the substraction of the mean for each of the time series. Furthermore it is becoming more and more clear that the pre-processing steps are very important; examples could be motion correction and correction for field inhomogeneity. Currently we have chosen to concentrate our efforts into the modelling of the data, and to only supply basic pre-processing steps and leave other preprocessing to other packages. However, we are open to contributions.

3.2 Volume file formats

Given that many have designed their own "best" file format, we cannot support all formats, and currently the toolbox can read fMRI data in the formats show in table 3.1

The *Custom* option allows `Lyngby` to read any data format with a little help from the user. The interface/conversion files that are required to help `Lyngby` understand the user's own data are very simple and straightforward to write and should only take a few minutes to construct. Full details and examples on how to write these files are given in the previous chapter — see section 2.4.2.

If your format cannot be read automatically (and you don't want to use the custom option) tell us about the format and maybe it can be incorporated in the toolbox soon. Currently we would like to support *AnalyzeSPM* (slightly changed *Analyze* format), the *VoxelView* format, and the *XPrime* format.

Name	Files	Extension	Usage
Analyze	2	hdr, img	Widely used: SPM, AIR and ANALYZE
Analyze 4D	2	hdr, img	All the time frames are in a single file
SDT	2	sdt, spr	Stimulate format
Vapet	1	—	Used at the Minneapolis Veterans Medical Center, PET Center
RAW	1	—	Support for raw binary data with several options of orientation and type
Custom	1	—	Support for other data where the user writes a small interface file that returns the volume at a given time step

Table 3.1: Data formats read by `Lyngby`.

3.3 Masking

The core of this toolbox lies in the composition of a datamatrix (named X in our Matlab code), where a certain row is the volume at that particular time, and a certain column corresponds to the time series for that particular voxel. This is shown in figure 3.1.

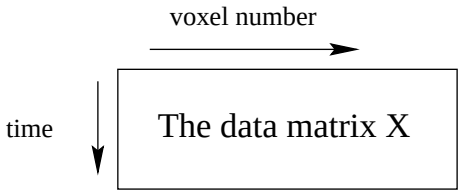


Figure 3.1: The data matrix X.

The toolbox has the potential to mask in both time and space, although currently the user only has a GUI interface with which to specify a box region.

A mask in time, $T_{\texttt{TIME\_MASK}}$ (the variable TIME_MASK in the code) is multiplied from the left of the full data matrix. And likewise a mask in space, $S_{\texttt{VOXEL\_MASK}}$ (the variable VOXEL_MASK in the code), is multiplied from the right. Before the `VOXEL_MASK` is applied, a further masking step can be made: A box region can be specified with the limits defined in the variable ROI_VOXELS.

$$X = T_{\texttt{TIME\_MASK}}\, X_{full}\, S_{\texttt{ROI\_VOXELS}}\, S_{\texttt{VOXEL\_MASK}} \tag{3.1}$$

We use sparse matrices for the `TIME_MASK` and the `VOXEL_MASK`, so these extra matrices only require a modest amount of memory. Note that if all the data is to be used then the two masks can be set to 1.

3.4 Preprocessing steps

3.4.1 Centering

Currently we have support for the following steps for removal of mean in various ways:

- Subtraction of mean from each voxel, i.e., the mean of each time series is forced to zero.

- Subtraction of mean from each scan, i.e., the mean of each volume at a certain time step is forced to zero.
- Subtraction of mean from each run, i.e., the mean of each run in each of the time series is forced to zero. Note that this makes the first substraction meaningless.

3.4.2 Variance Normalization

Algorithms such as neural networks in general use data normalization to unit variance in order to improve the stability of the algorithms. Note that this option might lead to unreliable results in other algorithms.

Chapter 4

Analysis — The Modeling Algorithms

$Revision: 1.46 $
$Date: 2004/01/21 10:54:40 $

4.1 Cross-correlation

The "Cross correlation" measures the temporal cross correlation between the paradigm and the measured data.

4.1.1 Our implementation

In the toolbox the paradigm can be cross-correlated with the time series. This produce a cross-correlation function, i.e. a function that is approximately twice as long. In general the cross-correlation function is only interesting around lag zero, and the user can decide how many lags that should be computed and stored. Note that people often only compute the cross-correlation for lag zero, which can lead to unreliable results in regions with long delays, so we maintain the temporal information as well.

In the display of the cross-correlation function we have enabled the display of the maximal value of the cross-correlation function as well as the delay defined as the position of the maximal value and the energy of the cross-correlation function.

4.2 FIR filter

The finite impulse response (FIR) filter is the same as a regression model or an ARX(0,n) model (that is an autoregressive model with exogenous input). The voxel denoted by u is regarded as a linear system with the input x (the paradigm), the filter (transformation function) to be estimated h, and the output (fMRI data) y which is disturbed by additive noise.

$$y(u,t) = \sum_{\tau=0}^{L-1} h(u, t-\tau)x(u,\tau) + \varepsilon_u(t) \tag{4.1}$$

The algorithm fits the parameters $h(\tau)$ so that the error is the least square solution. This

assumes that the noise ε_u is gaussian distributed. In that case the model $\hat{y}$ is

$$\hat{y}(u,t) = \sum_{\tau=0}^{L-1} h(u, t-\tau)x(u,\tau) \tag{4.2}$$

or in matrix vector notation for each value of u:

$$\hat{\mathbf{Y}} = \mathbf{X}\,\mathbf{h} \tag{4.3}$$

where $\hat{\mathbf{Y}} = [\hat{y}(u,0), \hat{y}(u,1), ..., \hat{y}(u,N-1)]$ and $\mathbf{X}$ contains the time-shifted values of the paradigm where each row of the matrix has L samples.

From the assumption that the error is Gaussian, the minimization of the error function

$$E_u = \sum_u (\hat{y} - y)^2 \tag{4.4}$$

can be identified directly through the so-called "normal equation":

$$\mathbf{X}^T\,\mathbf{Y} = \mathbf{X}^T\,\mathbf{X}\,\mathbf{h} \Rightarrow \tag{4.5}$$

$$\mathbf{h} = (\mathbf{X}^T\,\mathbf{X})^{-1}\mathbf{X}^T\,\mathbf{Y} \tag{4.6}$$

If the model is too complex — if the model is allowed to use too many filter coefficients — the model will not only fit to the signal within the data but also to the noise.

In the toolbox two ways of estimating the inverse matrix have been implemented for handling the often ill-posed inversion problem.

The first method for reducing the model complexity without reducing the lag size is to introduce regularization. One regularization technique is the ridge regression controlled by a single parameter κ_{ridge}. Note that the neural network community calls this weight decay and that this technique biases the filter coefficients towards zero.

$$\mathbf{h} = (\mathbf{X}^T\,\mathbf{X} + \kappa_{ridge}\mathbf{I})^{-1}\mathbf{X}^T\,\mathbf{Y} \tag{4.7}$$

where $\mathbf{I}$ is the unity matrix.

The other estimation technique computes the singular value decomposition (SVD) of the matrix to invert (In this case the $\mathbf{U}$ and $\mathbf{V}$ will be equal, so that cheaper and equivalent numerical methods can be used):

$$\mathbf{X}^T\,\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{4.8}$$

hence

$$(\mathbf{X}^T\,\mathbf{X})^{-1} \approx \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^T \tag{4.9}$$

where a threshold κ_{SVD} controls how many singular values

enter the pseudo inverse.

$$\sigma^{+}_{i,j} = \begin{cases} \sigma_{i,i}^{-1} & \text{if} \quad \sigma_{i,i} \geq \kappa_{SVD} \quad \text{and} \quad i = j \\ 0 & \text{if} \quad \sigma_{i,i} < \kappa_{SVD} \quad \text{or} \quad i \neq j \end{cases} \tag{4.10}$$

Note that this inversion does not depend on the voxel position u, hence a transformation matrix $\mathbf{T} = (\mathbf{X}^T\,\mathbf{X})^{-1}\mathbf{X}^T$ can be generated once and then the filter for each value of u can be generated from the linear transform

$$\mathbf{h}_u = \mathbf{T}\,\mathbf{Y}_u \tag{4.11}$$

This type of regularization is the same as principal component regression (Jackson, 1991).

Note that for the FIR filter with a symmetric paradigm (symmetric with respect to rise and fall flange) the response will also be symmetric. This means that if you have, for example, an on/off square wave paradigm, then a positive spike at the rise flank will force a negative spike at the fall flank.

4.2.1 Estimation of the Activation strength in the FIR filter

Using the estimated FIR filter, an activation strength can be defined as the standard deviation of the data estimate, normalized by the standard deviation of the paradigm.

$$A(u) = \sqrt{\frac{V(\hat{y}(u,t))}{V(\mathbf{h})}} \tag{4.12}$$

where V is a variance estimator.

4.2.2 Estimation of the delay from the FIR Filter

Dealing with a FIR filter makes it easy to derive several descriptors. One is the group delay, e.g., (Oppenheim and Schafer, 1989, pp. 205) defined as,

$$\tau_{FIR}(\omega) = -\frac{\partial}{\partial \omega} \arg H(\omega) = \frac{\frac{\partial H_r(\omega)}{\partial \omega} H_i(\omega) \; - \; \frac{\partial H_i(\omega)}{\partial \omega} H_r(\omega)}{|H(\omega)|^2} \tag{4.13}$$

where index r and i denote the real and the imaginary part respectively.

$$H(\omega) = \sum_{n=0}^{N} h(n) e^{-j\omega n} = \sum_{n=0}^{N} h(n)(\cos(\omega n) - j \sin(\omega n)) \tag{4.14}$$

$$\frac{\partial H(\omega)}{\partial \omega} = -\sum_{n=0}^{N} n h(n) e^{-j\omega n} = \sum_{n=0}^{N} n h(n)(-\sin(\omega n) - j \cos(\omega n)) \tag{4.15}$$

which means that the delay has a frequency dependence, and given that the paradigm functions often are dominated by the low frequencies, a simple approximation is to neglect the frequency dependence and only consider the delay for $\omega = 0$;

$$\tau_{FIR} = - \left. \frac{\frac{\partial H}{\partial \omega}}{H(\omega)} \right|_{\omega=0} = \frac{\sum t \, h(u,t)}{\sum h(u,t)} \tag{4.16}$$

For some combinations of filter coefficients the denominator of equation 4.16 can be very small which will influence the delay estimate dramatically. This problem can be explained from the fact that equation 4.16 is based on a low frequency dominance and in case of a very high noise level the assumption is in general not valid, and the estimate should rather be disregarded.

A simple way to check whether the assumptions is fulfilled is to evaluate

$$\left| \sum h(u,t) \right| > \gamma \sum |h(u,t)| \tag{4.17}$$

where γ should at least be larger than 0.5. More details can be found in (Nielsen et al., 1997).

4.2.3 Our Implementation

Currently the user has to choose to parameters depending on whether the inversion type is done using equation 4.9 or equation 4.7.

- The order of the filter can be varied.
- The regularization parameter

- For SVD: The lowest level of singular values accepted κ_{SVD}.
- For ridge regression: the weight decay parameter κ_{Ridge}

The order of the filter should depend on the number of scans in the time series and the magnitude of the regularization. With a too high order the model will mainly fit the noise in the data, — with a too small order it will not be able to fit the data.

Currently, although a suitable order must be chosen manually, this does not seem to degrade the results significantly. If an automatic determination of the order is required, then the more time consuming algorithm described in section 4.3 should be used.

4.2.4 References

The standard linear regression we employ here should be described in most books about time series analysis. The ridge technique in regression was first applied by (Hoerl and Kennard, 1970b) (Hoerl and Kennard, 1970a). For an annotated bibliography see (Alldredge and Gilb, 1976). Ridge-regresion is a form of Tikhonov regularization for linear models (Tikhonov, 1963) (Tikhonov and Arsenin, 1977) (Hansen, 1996). Principal component regression is described in (Jackson, 1991).

The application of the FIR filter on fMRI is described in (Goutte et al., 2000) and shortly in (Nielsen et al., 1997). Another type of linear time series analysis on fMRI is described in (Locascio et al., 1997).

4.3 Exhaustive FIR filter

In the FIR filter method (section 4.2) the user has to make the choice about the length of the filter. In principle the filter length is a parameter that can be varied as a function of the voxel index. In voxels where the signal is complex and there is a high signal to noise ratio many filter coefficients may be needed. On the other hand, in voxels where only noise is observed none of the estimated filter coefficients are stable, and the best stable signal estimator is a constant of value zero, i.e. zero taps in the filter.

In the Exhaustive FIR filter, the idea is to find the optimal filter length in each voxel. The optimal model is chosen by using the mean generalization error as a measure based on a training and test scheme. (this scheme is especially used in artificial neural networks, see section 4.9).

The basic idea is that each run in a time series is considered as an independent time series. Using resampling without replacement with that prior it is possible create a new dataset that gives good statistics in the generalization error.

The mean generalization error for a given model is defined as

$$E_{gen}(u) = \frac{1}{NT} \sum_{t=1}^{T} \sum_{i=1}^{N} (Y_i(u,t) - Y_{estimat,i}(u,t))^2 \tag{4.18}$$

where i denotes the resample index and t is the time.

The new dataset created with the resampling is split into a training and a test set. The ratio between the size of these two datasets is used in the algorithm as a variable.

The Exhaustive FIR filter uses a varying filter length combined with the SVD regularization. This means that for each filter length there is a "filter length" of singular values that can enter the pseudo inverse. So for each filter length there is a model using the largest singular value, and another model using the two largest singular values etc. The number of models is now given

by $1+2+...+\text{Maxorder} = \frac{\text{Maxorder}(\text{Maxorder}+1)}{2}$, which can lead to several hundred different models being tested.

The job is now to calculate the generalization error (test error) which depends of the different models and the split ratio (for splitting the dataset). The model with the smallest mean generalization error is chosen as the optimal model for the voxel.

When the optimal model for each voxel has been estimated, the activation strength can be found from equation 4.12. Note also that the number of filter coefficients will vary across the volume, and this might also be of interest to investigators.

4.3.1 Our Implementation

All exhaustive FIR functions have an infix of `_efir_`. The two parameters that can be set are the "filter length" and the "reshuffle" amount. The reshuffle amount determines how many times the data set resampling is done.

4.3.2 References

We can not make any direct references in connection with the exhaustive FIR model. Consult section 4.2.4 for reference in relation to the architecture of the model. Generalization and train/test set are terminology from the artificial neural network society, and reference to that can be found in section 4.9.2.

4.4 The Ardekani t-test

In (Ardekani and Kanno, 1998) another type of t-test is described. The method is rather different than the one described in section 4.14 and it is actually closer to the cross-correlation method.

The basic idea is to project each of the time-series onto a subspace driven by the paradigm function. The t-statistics are obtained by dividing by the estimate of the standard deviation found from the part of the energy in the time-series that is not explained by the paradigm.

4.4.1 Our Implementation

Our implementation follows the paper rather closely and as suggested in the paper we have implemented a delay option, so that the paradigm can be delayed a number of samples compared to the time-series in order to adapt to a possible delay in the data.

4.4.2 References

Babak Ardekani's t-test is described in (Ardekani and Kanno, 1998).

4.5 The Ardekani F-test

In (Ardekani and Kanno, 1998) an F-test has been presented to identify activated regions. The strategy has some common ideas with the Ardekani t-test (described in the same article).

The basic idea is to project each of the time-series onto a subspace driven by a truncated Fourier expansion of the data. The power of the data in the subspace driven by the expansion is divided by the power of the data that lies outside of the Fourier subspace. Thus an F-statistic is obtained without using requiring any knowledge of the paradigm.

In (Ardekani et al., 1999) the F-test is extended with the modeling of a nuisance subspace: A subspace orthogonal to the subspace spanned by the Fourier expansion is found and this subspace is then extracted from the original signal. The dimension of the subspace is found by using Akaike's information criterion.

4.5.1 Our Implementation

Functions belonging to the (Barbak) Ardekani F-test have the infix `_baf_`, and function for the Ardekani F-test with nuisance have the infix `_baf2_`. Our implementation follows the paper rather closely. It is also intended to implement alternatives to the Fourier expansion.

4.5.2 References

Babak Ardekani's F-test is described in (Ardekani and Kanno, 1998). The F-test with nuisance parameters extension is described in (Ardekani et al., 1999).

4.6 K-means Clustering

The K-means algorithm is a simple non-parametric clustering method.

The idea behind K-means clustering here is to classify the individual voxel in the volume with respect to their time series. This is indicated in figure 4.1. After the clustering the individual clusters may be analyzed further. The method is currently sensitive to the initial cluster positions and the algorithm converges fast.

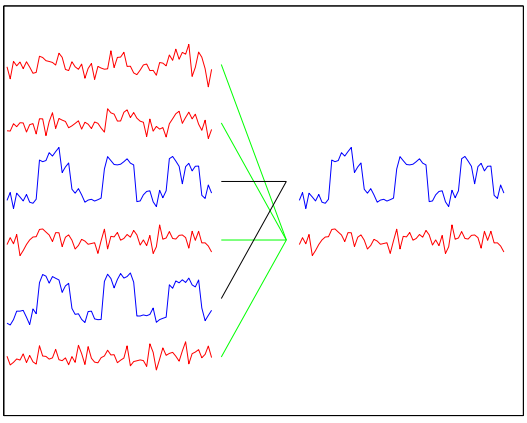


Figure 4.1: The time-series are each assigned to the cluster center with the best match.

In the K-means algorithm, K cluster centers have to be chosen (The dimension of the space is equal to the size of the time series length). Then the distance from each voxel to each cluster center is calculated. Each voxel is then assigned to the cluster which is the minimum distance away, as shown in figure 4.2; lastly (before iterating) the new cluster center is calculated as the mean of all the voxels assigned to that cluster.

1. Initialize K cluster centers $\mathbf{C}_k^{(0)}$ of same dimensionality as the time series, iteration $i = 0$.
2. Assign each data vector $\mathbf{x}_j$ to the cluster with the nearest center $\mathbf{C}_k^{(i)}$. Currently we use the ordinary Euclidean distance metric $\|\mathbf{C}_k^{(i)} - \mathbf{x}_j\|$.

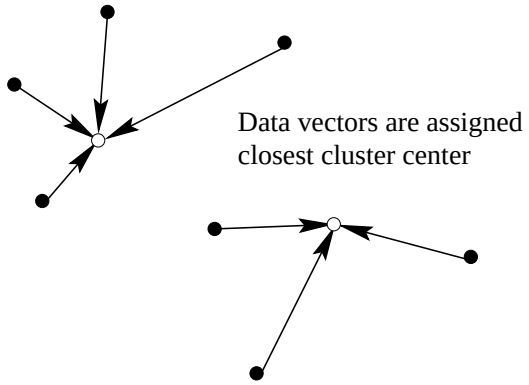


Figure 4.2: In a high dimensional space the data vectors are clustered to the nearest cluster center.

3. Set new cluster centers $\mathbf{C}_k^{(i+1)}$ to the center of gravity of each cluster:

$$\mathbf{C}_k^{(i+1)} = E\{\mathbf{x}_j\}_{\mathbf{x}_j \in \mathbf{C}_k^{(i)}} \tag{4.19}$$

   This formula can also be modified to use the median and/or to include an inertia term.

4. Goto step 2 until convergence.

4.6.1 Our implementation

Instead of using the time series directly as the input to the K-means algorithm, we have made it possible to use the cross-correlation of the fMRI time series and the paradigm. This has a major impact of the results of K-means algorithm, which will make it far easier to find the activated voxels (Toft et al., 1997).

When clustering on the cross-correlation function or the raw time series, the final clusters centers will be affected by the amplitude level and a possible lag. For instance, if two spatially separated regions have similar response strength but different delays, then they will be clustered into two different clusters according to their delay values.

The toolbox enables clustering using K-means or K-median (K-mediod). The range of the variables can be "standardized" (normalized) to unit variance or standardized according to the min-max-range. The cluster centers can be initialized randomly (In (Toft et al., 1997) it has been found that this strategy works rather well for the cross-correlation clustering) or according to the correlation with the paradigm function. The variable to cluster on can be set to the raw time series or the cross-correlation function.

It is implemented in the `lyngby_km_main` matlab function.

4.6.2 References

The K-means clustering algorithm was first described in (MacQueen, 1967). Other "non-neuroimaging" descriptions of this algorithm are available in (Ripley, 1996, section 9.3), (Sonka et al., 1993, section 7.2.4) or (Hartigan and Wong, 1979). Our approach in connection with functional neuroimaging is published in (Goutte et al., 1999) and described shortly in (Goutte et al., 1998;

Toft et al., 1997). In (Liptrot et al., 2003) it was applied to extract the time-activity curve from dynamic PET scans in connection with the $5HT_{2A}$-receptor ligand.

Different kinds of clustering methods have been applied in functional neuroimaging. Some of the references are collected at http://www.inrialpes.fr/is2/people/goutte/fmriclusterefs.html

4.7 Kolmogorov-Smirnov test

Most standard text books in statistics show the Kolmogorov Smirnov test, where the maximal difference between two histograms is used as a measure of match between the two signals. Hence a very small difference indicates that the two signals are nearly identical. Note that the method only uses the histograms, hence the temporal placement is not used here.

4.7.1 Our implementation

As mentioned in section 1.1.1, each of the time series is split into an activated part and a baseline part, determined from the

paradigm function. In order to compensate somewhat for the hemodynamic response time, we have made it possible to drop a number of samples after each transition from the baseline to an activated state and vice versa.

The Kolmogorov Smirnov test requires the histograms, hence some sorting of signal values is needed. This implies that the function might be slow if the fMRI data has many time elements.

4.7.2 References

A critique of the Kolmogorov-Smrnov test used for fMRI has been made in (Aguirre et al., 1998).

4.8 Lange-Zeger

The Lange-Zeger model (Lange and Zeger, 1997) fits a three parameter gamma density function h_{LZ} as the convolution filter:

$$h_{LZ}(u,t) = \beta(u)\theta_{2,u}(\theta_{2,u}t)^{\theta_{1,u}-1}\exp(-\theta_{2,u}t)/\Gamma(\theta_{1,u}) \tag{4.20}$$

The gamma density function is convolved with the paradigm to give the voxel value.

$$y(u,t) = \sum_{\tau} h(u,t-\tau)x(u,\tau) \tag{4.21}$$

Note that the Lange-Zeger kernel is normalized so that the β is the power amplitude of the hemodynamic response. The dependence on the two parameters can be seen from figures 4.3 and 4.4.

4.8.1 Estimation of the delay from the Lange Zeger model

The Lange Zeger model is in principle a constrained version of the FIR model, hence a delay can be estimated just as it was in subsection 4.2.2.

In the Lange Zeger case, the spectrum has a rather simple expression

$$H_{LZ}(\omega) = \left(1 + \frac{j\omega}{\theta_{2,u}}\right)^{-\theta_{1,u}} \tag{4.22}$$

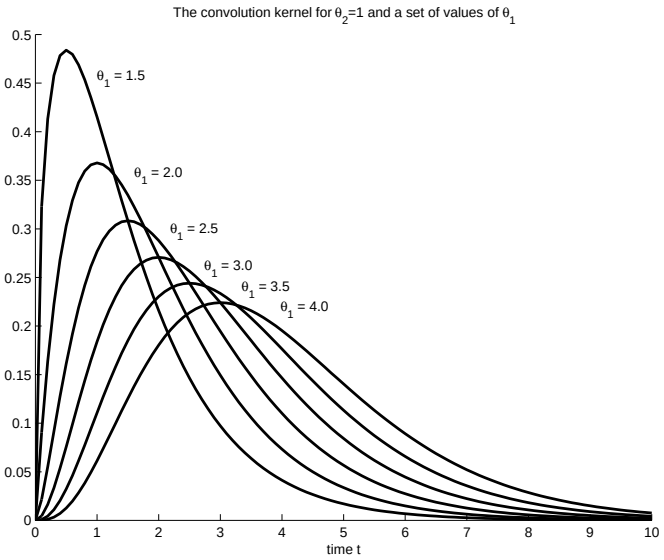


Figure 4.3: The gamma density kernel as a function of the time t for $\beta = 1$ and $\theta_2 = 1$ for a set of θ_1.

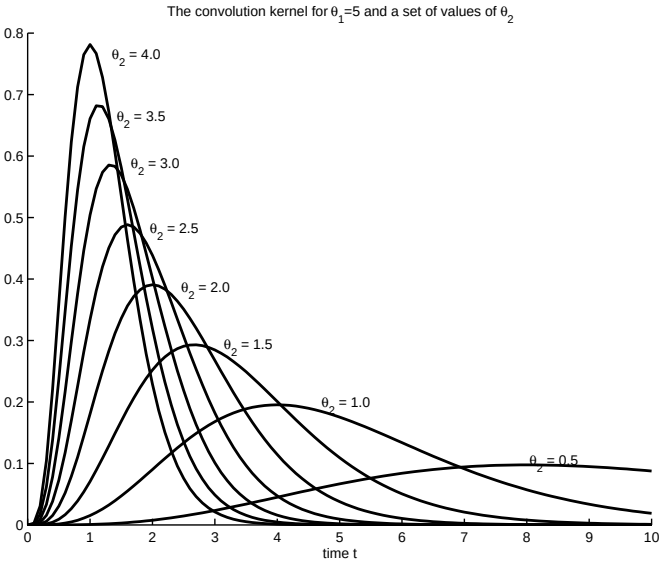


Figure 4.4: The gamma density kernel as a function of the time t for $\beta = 1$ and $\theta_1 = 5$ for a set of θ_2.

hence the delay for very low frequencies ($\omega = 0$) ends up in a very simple form

$$\tau_{LZ}(u) = \frac{\theta_{1,u}}{\theta_{2,u}} \tag{4.23}$$

4.8.2 Our implementation

Compared to the original algorithm, our algorithm does not perform any noise estimates as suggested in (Lange and Zeger, 1997). Furthermore we have experienced that the iterative way of estimating the model parameters suggested in (Lange and Zeger, 1997) is often unstable, which cannot be explained with the lack of a better noise model. In certain situations the noise makes the iterative algorithm diverge, resulting in extreme values of the model parameters.

In our implementation, the original iterative algorithm is available (though currently not in the GUI) as well as a version where the (θ_1, θ_2) space is grid searched and the value of β is determined directly in every grid point. Furthermore is it possible in the grid search version to use regularization on the value of β penalizing the extreme values.

It should also be mentioned that the Lange-Zeger kernel shown in equation 4.20 is a continuous function of time t, which needs to be sampled in time as indicated in equation 4.21. This implies that the number of samples taken from equation 4.20 should at least include the maximum point of the kernel, which is found at

$$t = \arg\max h_{LZ} = \frac{\theta_{1,u} - 1}{\theta_{2,u}} \tag{4.24}$$

4.9 Neural networks

By neural networks we mean *artificial* neural network. They have little to do with *biologic* neural network, — we are not trying to simulate the biological neural circuit. The name "neural network" has arisen because the first versions of these mathematical models was highly inspired by the biological neural circuits. We use neural networks solely as general purpose non-linear statistical models. In some sense the neural network is a non-linear generalization of the linear model (see the FIR model in section 4.2).

The main type of neural network we employ is the *two-layer feed-forward neural network*. It consists of two layers of weights (the neural network name for the model parameters) and two (or three) layers of "neurons" (or units). The first layer of neurons is not usually counted as a layer: It is the input to the neural network. The second layer is the hidden layer. The neurons in this layer have an activation function, and it is necessary for the non-linearity of neural network that this activation function is non-linear. The final layer is the output layer. These will also have an activation function. This might be linear or non-linear.

With x as the input, y as the output, with v as the first layer of weights (the input-to-hidden weights) and w as the second (the hidden-to-output weights) and with i, h, o and p as the indices for the input, hidden and output neurons, and the examples, respectively, we get the following *neural network function*:

$$y_o^p = g^o\left(\sum_h^{n_h} w_{ho} g^h\left(\sum_i^{n_i} v_{ih} x_i^p + v_{h0}\right) + w_{o0}\right) \tag{4.25}$$

Here, g^o and g^h are the activation functions and v_{h0} and w_{o0} are the *biases*.

The activation function is usually of the sigmoidal type and we use the hyperbolic tangent. In connection with classification the output activation function is this hyperbolic tangent to get a restricted output that can be interpreted as a probability. In connection with regression the output activation function is linear.

The neural network is optimized to predict to a target, i.e. to make the (numerical) difference between a desired target t and the computed output of the neural network y as small as possible.

As a measure for the discrepancy (how well we have optimized the network) we can use different kinds of *cost functions*. In regression we usually strive for uniformly minimum variance, thus we can use the mean square error for the measure:

$$E_q = \sum_p^{n_p} \sum_o^{n_o} (t_o^p - y_o^p)^2 \tag{4.26}$$

In the case of two-class classification a more appropriate measure is the two-class (cross-)entropy error:

$$E_e = \sum_p^{n_p} \sum_o^{n_o} \left[\frac{1}{2}(1 + t_o^p) \ln \frac{1 + t_o^p}{1 + y_o^p} + \frac{1}{2}(1 - t_o^p) \ln \frac{1 - t_o^p}{1 - y_o^p} \right] \tag{4.27}$$

The cross-entropic errorfunction of equation 4.27 requires the target to be $t = 1$ or $t = -1$.

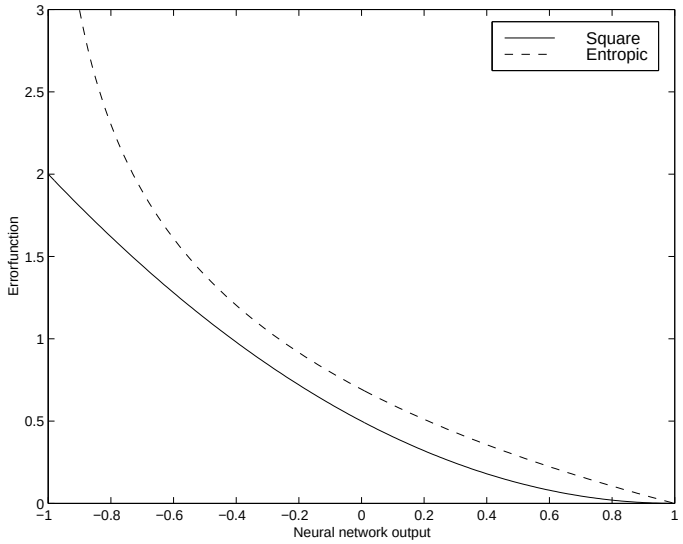


Figure 4.5: Comparison of the square and the cross-entropic errorfunction. The target is $t = 1$. The cross-entropic errorfunction is penalizing large deviation from the target value much more than the square errorfunction.

Another type of errorfunction is the cross-entropy for multiple classes (Bishop, 1995). We use a so-called "$c - 1$" variation where the number of output neurons in the neural network is one less than the number of classes (Andersen et al., 1997).

$$E_c = ln \left(1 + \sum_o^{n_o - 1} \exp(\phi_o) \right) - \sum_o^{n_o - 1} t_o \phi_o \tag{4.28}$$

This errorfunction is developed for a neural network with neural network function as follows. The hidden layer neurons have a hyperbolic tangent as the activation function:

$$h_h^p = \tanh \left(\sum_h^{n_i} v_{hi} x_i^p \right) + v_{h0} \tag{4.29}$$

The second layer has linear activation functions:

$$\phi_o = \sum_h^{n_h} w_{oh} h_h^p + w_{h0} \tag{4.30}$$

To be able to interprete the output as a probability we use a *softmax* layer:

$$y_o = \begin{cases} \frac{\exp(\phi_o)}{\sum_{o'}^{n_o} \exp(\phi_{o'})+1}, & \text{if } o < n_o \\ 1 - \sum_{o'}^{n_o} y_{o'}, & \text{if } o = n_o \end{cases} \tag{4.31}$$

The neural network is optimized by adjusting the weights in an iterative scheme, either with gradient or with Hessian based methods, see, e.g., (Ripley, 1996, appendix A.5). The term "back-propagation" is usually used to denote gradient-based optimization.

4.9.1 Our implementation

All neural network functions have the prefix `lyngby_nn`. In the present implementation there are two types of neural networks: A regression type with least square fitting and a classification type with a cross entropic cost function suitable for binomial distributions. The functions in connection with the regression neural network have the prefix `lyngby_nn_q` where the "q" stands for quadratic. The second group of functions has the prefix `lyngby_nn_e` where the "e" is for entropic. Functions that are common do only have the prefix `lyngby_nn`.

4.9.2 References

There are a number of good books for that introduce neural networks, e.g., Chris Bishop's *Neural Networks for Pattern Recognition* (Bishop, 1995) and Brian Ripley's *Pattern Recognition and Neural Networks* (Ripley, 1996). Others are (Haykin, 1994) and (Hertz et al., 1991).

The two-layer feed-forward neural network is described in several papers from our department, — the first important one being (Svarer et al., 1993a), followed by (Hintz-Madsen et al., 1995) (Hintz-Madsen et al., 1996a) (Hintz-Madsen et al., 1996b) and (Svarer et al., 1993b).

4.10 Neural network regression

The "Neural network regression" is a neural network generalization of the FIR model. For the input to the neural network we use the paradigm, and the weights are adjusted to target the output to the fMRI time series signal. One neural network is trained for every voxel.

The ability of the neural network to predict the fMRI time series is used

as a measure for the strength of the signal in a voxel: If this signal can be predicted from the paradigm signal then the voxel is activated.

4.10.1 Our implementation

The functions that belong to the neural network regression analysis method have the prefix `lyngby_nnr`.

The optimization of a single neural network is usually a tedious affair and when one neural network has to be optimized for every single voxel, the computational effort of this analysis method is enormous.

The neural network regression analysis is currently under implementation, and it does not provide any means for assuring that the neural network is generalizing well so it is presently dangerous to use the neural network regression.

4.10.2 References

To our knowledge using neural networks in connection with fMRI time series regression is not described anywhere, yet. A first application was in a comparison between models (Lange et al., 1996).

4.11 Neural network saliency

Contrary to the neural network regression analysis (section 4.10) the "Neural Network Saliency" does not analyze the fMRI signal as time series. Rather, the neural network treats each scan as an individual object that is used in a classification or regression, and the *saliency map* estimates each voxel's importance for this prediction.

Each scan is — after an SVD-projection or other linear transformation — used as the input to the neural network. The state of the paradigm corresponding to the scan is used as the target. The SVD-projection is usually necessary because the weight optimization problem will be highly ill-posed if the number of parameters is not heavily reduced.

After the neural network has been optimized for best generalization the actual saliency computation is performed. The saliency resembles the saliency from OBD (Le Cun et al., 1990), but where the OBD-saliency is for the weights the saliency in connection with the saliency map is for the variables, i.e. each individual voxel. The saliency describes the change in the generalization error when a voxel is deleted. The method to derive the change in generalization error is based on a Taylor expansion of the neural network costfunction and regarding the effect of the small perturbation when deleting a weight. There are several levels of approximation from the generalization error to the actual estimate.

Let the matrix $\mathbf{B}$ denote the linear transformation we apply to the voxel datamatrix $\mathbf{X}$ to project it onto a subspace containing the data described by the subspace datamatrix $\tilde{\mathbf{X}}$:

$$\tilde{\mathbf{X}} = \mathbf{XB} \tag{4.32}$$

The vectors $\tilde{\mathbf{x}}^p$ from the subspace matrix $\tilde{\mathbf{X}}$ are used as input to a two-layer feed-forward neural network, see section 4.9. When the neural network is fully optimized to best generalization, the full costfunction from the voxel to the output of the neural network is Taylor-expanded with respect to the components in the linear transformation $\mathbf{B}$. For an "entropic neural network" the first order derivative and the diagonal approximation of the second order derivative yield :

$$\frac{\partial C_e}{\partial b_{il}} = -\sum_p^{n_p}\sum_h^{n_h} (t^p - y^p)\, w_{oh} \left(1 - \left(h_h^p\right)^2\right) v_{hi} x_l^p \tag{4.33}$$

$$\frac{\partial^2 C_e}{\partial b_{il}^2} = \sum_p^{n_p} \left(1 - \left(y_o^p\right)^2\right) \left[\sum_h^{n_h} w_{oh} \left(1 - \left(h_h^p\right)^2\right) v_{hi}\right]^2 \left(x_l^p\right)^2 \tag{4.34}$$

The symbols are: $i = 1..n_i$ is indicing over inputs to the neural network, i.e., the output of the linear transformation and $l = 1..n_l$ indices over voxels. The rest of is the same as the symbols of section 4.9. Entering the derivatives in the Taylor expansion yields:

$$\begin{aligned} \delta C_{e,l} \approx & \sum_p^{n_p}\sum_h^{n_h}\sum_i^{n_i} (t^p - y^p)\, w_{oh} \left(1 - \left(h_h^p\right)^2\right) v_{hi} b_{il} x_l^p \\ & + \frac{1}{2}\sum_p^{n_p}\sum_i^{n_i} \left(1 - \left(y^p\right)^2\right) \left[\sum_h^{n_h} w_{oh} \left(1 - \left(h_h^p\right)^2\right) v_{hi}\right]^2 b_{il}^2 \left(x_l^p\right)^2 \end{aligned} \tag{4.35}$$

Similar calculations can be performed for the costfunction to the "quadratic neural network" C_q. Here the derivatives become:

$$\frac{\partial C_q}{\partial b_{il}} = -2\sum_p^{n_p}\sum_o^{n_o}\sum_h^{n_h}\left(t_o^p - y_o^p\right) w_{oh}\left(1-\left(h_h^p\right)^2\right) v_{hi} x_l^p \tag{4.36}$$

$$\frac{\partial^2 C_q}{\partial b_{il}^2} = 2\sum_p^{n_p}\sum_o^{n^o}\left[\sum_h^{n_h} w_{oh}\left(1-\left(h_h^p\right)^2\right) v_{hi}\right]^2 \left(x_l^p\right)^2 \tag{4.37}$$

Substituting these derivatives into the Taylor expanded saliency yields:

$$\begin{aligned}\delta C_{q,l} \approx\ & 2\sum_p^{n_p}\sum_o^{n_o}\sum_h^{n_h}\sum_i^{n_i}\left(t_o^p - y_o^p\right) w_{oh}\left(1-\left(h_h^p\right)^2\right) v_{hi} b_{il} x_l^p \\ & +\sum_p^{n_p}\sum_o^{n_o}\sum_i^{n_i}\left[\sum_h^{n_h} w_{oh}\left(1-\left(h_h^p\right)^2\right) v_{hi}\right]^2 b_{il}^2\left(x_l^p\right)^2\end{aligned} \tag{4.38}$$

4.11.1 Our implementation

The functions that are specific for the neural network saliency all have the infix `_nns_`. The optimization of the neural network is performed with the functions calls `_nn_`.

4.11.2 References

The first description of the neural network saliency was in Mørch (Mørch et al., 1995) and (Mørch et al., 1996). It has also been the subject of three master's thesis within our department, (Lundsager and Kristensen, 1996), (Mørch and Thomsen, 1994) and (Nielsen, 1996). The method is still under development, i.e. the mathematical form of the saliency is not yet fully justified.

4.12 The SCVA model: Strother Canonical Variate Analysis

The SCVA model is directly related to the SOP model (section 4.13: It uses the same designmatrix, — equation 4.51). Using the same terminology as Mardia (Mardia et al., 1979) the analysis method should not be called canonical variate analysis but canonical correlation analysis.

CVA together with SVD on a datamatrix with degenerate rank is ambiguous: The canonical correlation coefficients will all be one, thus the canonical correlation vectors will have no unique orientations. A technique to cope with this problem is *canonical ridge*, which is a variation of ridge regression within canonical variate analysis. With the ridge regression parameters on the highest level, the canonical ridge will become PLS (see section 4.13).

The ordinary canonical correlation analysis / canonical variate analysis is the singular value decomposition of a (normalized) cross-correlation matrix:

$$\left(\mathbf{G}^\mathsf{T}\mathbf{G}\right)^{-1/2}\mathbf{G}^\mathsf{T}\mathbf{X}^*\left(\mathbf{X}^{*\mathsf{T}}\mathbf{X}^*\right)^{-1/2} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\mathsf{T} \tag{4.39}$$

Here $\mathbf{X}^*$ is the SVD projection (see also the equivalent equation 4.53):

$$\mathbf{X}^* = \mathbf{X}\mathbf{V}^{*\mathsf{T}} = \mathbf{U}^*\mathbf{\Lambda}^* \tag{4.40}$$

The equality sign only holds if there are more variables than objects (more voxels than scans).

$\mathbf{V}^*$ is found through a normal singular value decomposition:

$$\mathbf{X} = \mathbf{U}^*\mathbf{\Lambda}^*\mathbf{V}^{*\mathsf{T}} \tag{4.41}$$

Ordinary canonical variate analysis can be extended to *canonical ridge*. Canonical ridge is usually written as:

$$\left(\mathbf{G}^\mathsf{T}\mathbf{G} + k_G\mathbf{I}\right)^{-1/2}\mathbf{G}^\mathsf{T}\mathbf{X}^*\left(\mathbf{X}^{*\mathsf{T}}\mathbf{X}^* + k_X\mathbf{I}\right)^{-1/2} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\mathsf{T} \tag{4.42}$$

A different way to write the canonical ridge equation is by using another form of scaling:

$$\left((1-k_G)\,\mathbf{G}^\mathsf{T}\mathbf{G} + k_G\mathbf{I}\right)^{-1/2}\mathbf{G}^\mathsf{T}\mathbf{X}^*\left((1-k_X)\,\mathbf{X}^{*\mathsf{T}}\mathbf{X}^* + k_X\mathbf{I}\right)^{-1/2} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\mathsf{T} \tag{4.43}$$

This has the advantage that the ridge parameters k_G and k_X are *morphing* between ordinary CVA and PLS via orthonormalized PLS.

	k_X	k_G
ordinary CVA	0	0
ordinary PLS	1	1
orthonormalized PLS	0	1

Equation 4.40 can be used to eliminate $\mathbf{X}^*$:

$$\left((1-k_G)\,\mathbf{G}^\mathsf{T}\mathbf{G} + k_G\mathbf{I}\right)^{-1/2}\mathbf{G}^\mathsf{T} \quad \mathbf{X}^*\left((1-k_X)\,\mathbf{X}^{*\mathsf{T}}\mathbf{X}^* + k_X\mathbf{I}\right)^{-1/2} \tag{4.44}$$

$$\mathbf{U}^*\mathbf{\Lambda}^*\left((1-k_X)\,(\mathbf{U}^*\mathbf{\Lambda}^*)^\mathsf{T}\,\mathbf{U}^*\mathbf{\Lambda}^* + k_X\mathbf{I}\right)^{-1/2} \tag{4.45}$$

$$\mathbf{U}^*\mathbf{\Lambda}^*\left((1-k_X)\,\mathbf{\Lambda}^{*\mathsf{T}}\mathbf{U}^{*\mathsf{T}}\mathbf{U}^*\mathbf{\Lambda}^* + k_X\mathbf{I}\right)^{-1/2} \tag{4.46}$$

$$\mathbf{U}^*\mathbf{\Lambda}^*\left((1-k_X)\,\mathbf{\Lambda}^{*\mathsf{T}}\mathbf{\Lambda}^* + k_X\mathbf{I}\right)^{-1/2} \tag{4.47}$$

If $k_X = 0$, as in ordinary CVA, we will get a simpler equation:

$$\left((1-k_G)\,\mathbf{G}^\mathsf{T}\mathbf{G} + k_G\mathbf{I}\right)^{-1/2}\mathbf{G}^\mathsf{T} \quad \mathbf{U}^*\mathbf{\Lambda}^*\left(\mathbf{\Lambda}^{*\mathsf{T}}\mathbf{\Lambda}^*\right)^{-1/2} \tag{4.48}$$

$$\mathbf{U}^*\mathbf{\Lambda}^*\mathbf{\Lambda}^{*-1} \tag{4.49}$$

$$\mathbf{U}^* \tag{4.50}$$

$\mathbf{U}^*$ is an orthonormal matrix: it is a matrix containing eigenvectors — eigensequences — of the original datamatrix. Such a matrix does not contain any principal direction. All directions are equally strong, i.e., the eigenvalues have the same magnitude. Multiplying with another orthonormalized matrix (e.g., the designmatrix part of equation 4.50) will not bring up any new principal direction. When there are no principal directions in the matrix, the eigenvectors — eigenimages — cannot be said to represent anything but an arbitrary direction.

4.12.1 Our implementation

The main function that implements the SCVA model is called `lyngby_scva_main`. Before the datamatrix is singular value decomposed it is doublecentered. The actual canonical variate analysis function is called `lyngby_cva`.

The number of components (singular values) maintained in the final singular value decomposition can be varied by the user. Usually only three or less components are significant. The function will only return eigenvectors with eigenvalues that are different from "zero", i.e. larger than a tolerance set to account for numerical round-off errors.

In the present implementation it is only possible to do the analysis on a run basis, — not within run (e.g. if you have a repeated stimulus function in a run), and not with longer periods than a run. The run specification is used to make the designmatrix.

4.12.2 References

SVD and CVA is usually explained in most multivariate analysis textbooks, e.g., Mardia (Mardia et al., 1979). An short overview of multivariate analyses is available in (Worsley et al., 1997) or (Worsley, 1997). The special canonical ridge technique is described shortly in (Mardia et al., 1979) which reference the original article (Vinod, 1976). Furthermore, the canonical ridge analysis has a special case in PLS (see section 4.13).

The SCVA model with the special designmatrix (which we here call the Strother's designmatrix) was used in (Strother et al., 1996). CVA has also been used in (Friston et al., 1995a) (Fletcher et al., 1996) and (Van Horn et al., 1996).

4.13 The SOP model: Strother Orthonormalized PLS

The SOP model consists of a preliminary SVD on the datamatrix, a orthonormalization of a designmatrix, followed by a partial least square analysis (PLS) between the SVD'ed datamatrix and the orthonormalized designmatrix. The outcome of this analysis is eigenimages, eigenvalues and the corresponding eigensequences are represent a dependency between the datamatrix and the designmatrix.

The designmatrix is constructed in a special way as in (Strother et al., 1996) (where it was used in connection with CVA — canonical variate analysis): All the scans that correspond to a specific period in a run are given their own class, e.g. the designmatrix of a tiny study consisting of 2 runs, each with four scans, will have the following structure:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.51}$$

It is of course important that the stimulus function (the paradigm) is the same for all runs.

We use partial least square, not as in introduced by McIntosh (McIntosh et al., 1996), but in the orthonormalized version as suggested by Worsley (Worsley et al., 1997) where the designmatrix is made invariant to linear transformations:

$$(\mathbf{G}^\mathsf{T}\mathbf{G})^{-1/2}\mathbf{G}^\mathsf{T}\mathbf{X}^* = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\mathsf{T} \tag{4.52}$$

Here $\mathbf{X}^*$ is the SVD projection:

$$\mathbf{X}^* = \mathbf{X}\mathbf{V}^{*\mathsf{T}} \tag{4.53}$$

And $\mathbf{V}^*$ is the eigenimage of $\mathbf{X}$:

$$\mathbf{X} = \mathbf{U}^*\mathbf{\Lambda}^*\mathbf{V}^{*\mathsf{T}} \tag{4.54}$$

The eigensequence $\mathbf{U}$ and the eigenimage of $\mathbf{V}$ of equation 4.52 are the interesting parts. These are sorted according to descending eigenvalues. The first eigenimage and eigensequence will (usually) have a direct relationship with the stimulus function.

4.13.1 Our implementation

The main function that implements the SOP model is called `lyngby_sop_main`. Before the datamatrix is singular value decomposed it is doublecentered.

The number of components (singular values) maintained in the final singular value decomposition can be varied by the user. Usually only three or less components are significant. The function will only return eigenvectors with eigenvalues that are different from "zero", i.e. larger than a tolerance set to account for numerical round-off errors.

In the present implementation it is only possible to do the analysis on a run basis, — not within a run (e.g. if you have a repeated stimulus function in a run), and not with longer periods than a run. The run specification is used to make the designmatrix.

4.13.2 References

SVD and CVA is usually explained in most multivariate analysis textbooks, e.g. Mardia (Mardia et al., 1979). For PLS, and orthonormalized PLS see the compact explanation in (Worsley et al., 1997).

The SOP model — orthonormalized PLS combined with Strother's designmatrix — is not described anywhere else than here.

4.14 The Ordinary t-test

A common test in functional brain modeling is the t-test. For a given voxel the fMRI signal is split into two sections: the activated part and a baseline part. The difference in means of the two parts divided by a measure of the standard deviation for the two parts can be modeled by the Student t-distribution. Hence a t-value (and/or a probability measure) for the two parts being identical can be derived.

If a Gaussian assumption is made about the noise and it is independent and identically distributed then a P-value can be derived. The noise from fMRI BOLD is usually not independent due to the hemodynamic response function and unmodelled confounds (autocorrelation in the noise).

Chapter 5

Post-Processing

$Revision: 1.8 $
$Date: 2002/08/14 12:44:33 $

This chapter covers the post-processing, or meta-analysis, stage within the toolbox. This is where the results of the previous analysis stage are then themselves analysed in an effort to gain more insight and robustness into the location of stimulations within functional images.

Currently, only one post processing algorithm has been implemented within the toolbox, although more are in the pipeline. Any suggestions for expanding this section will also be greatly received.

5.1 K-means Clustering on the Modelling Results

As described in section 4.6, the K-means clustering algorithm can be used to cluster from the time series or the cross-correlation function. It can also be taken a step further and be used as a post-processing tool.

Here, the result of the modelling method is fed into the K-means algorithm, regardless of the number of parameters generated. For the Kolmogorov Smirnov test the clustering could be made solely on the maximum test size, but with the FIR, for example, the whole filter could be fed to the clustering algorithm. The K-means will then cluster the similar results into the same bins, hence a label is generated for each of the voxels.

Note that this strategy can be questioned. If, for instance, a measure of activation and a delay measure are generated for each voxel and fed into the post-processing K-means clustering algorithms ("Meta K-Means"), then the magnitudes of the two parameters are very different, which naturally will bias the results. In our implementation, the Matlab matrix fed into K-means is called `RESULT` – this is generated by each of the modelling algorithms. If rescaling of the rows of the `RESULT` matrix is required before this clustering, e.g. multiplication of the first row by two, then, from the Matlab commandline, simply type:

```
>> global RESULT
>> RESULT(1,:) = 2*RESULT(1,:);
```

and then run the Meta K-means from the GUI as normal.

Appendix A

Glossary

$Revision: 1.3 $
$Date: 2002/08/14 13:20:17 $

Table A.1: An Explanation of `Lyngby` Toolbox, fMRI and Related Terms

Analysis	The core stage of the investigation of functional images within the toolbox, usually involving statistical algorithms acting upon a time series of volumes. It comes directly after the *pre-processing* stage, and before the *post-processing* one.
Axial slice	See *Transversal slice*
Coronal slice	A slice taken through a spatial volume. For the example of a person, this would be like looking at them from the front. As such, the slice index goes from posterior-anterior, or back-to-front.
Experiment	A term used to describe the entire set from which results are derived. An experiment can consist of several *trials* over different days, each of which may have several *runs*.
Frame	The spatial data (which could be 2D or 3D) available at each timepoint. Sometimes this is referred to as a *scan* or *volume*
	A portion of a window in the `Lyngby` toolbox that contains buttons that are related to each other. The main `Lyngby` window consists of a separate frame for each of the processing stages.
Layers	A term used in image processing to describe individual images put together to form a single composite image viewed on common axes. Part of each image will be transparent, allowing the images underneath to show through. It is rather akin to the building-up of a cartoon frame from different "cells" - a character or item is drawn on clear plastic. The majority of the layer is transparent allowing anything underneath to show through. However, individual items are not transparent and will obscure those directly underneath.
Pane	A portion of a window in the `Lyngby` toolbox that contains buttons that are related to each other. The *Load New Data...* window has three panes for the data parameters, external influences and the window control buttons.
Paradigm	The "activation" signal, usually binary, that indicates when the subject in a functional imaging experiment is performing some task.
Post-processing	The stage that comes after the analysis of the data. It usually consists of some form of analysis of the results sets, such as clustering of all the result parameters.

Pre-processing	The stage after the data has been loaded into the toolbox, but before the data analysis is started. It is an initialization stage, used to get the data into the correct form for analysis by the usual algorithms. It usually involves some form of mean-removal, normalisation and/or removal of any unwanted scans.
Run	The part of a time series of functional images where a certain task is performed. A run usually consists of several cycles of a paradigm. A single experiment can consist of several runs.
Sagittal slice	A slice taken through a spatial volume. For the example of a person, this would be like looking at them from the side. As such, the slice index goes from left-to-right. Derived from the Latin word for "arrow", due to the direction of the cutting plane indicated by the angle of an arrow striking a person from the front.
Slice	A spatial volume one voxel thick i.e. a 2-D image. Usually refers to an entire time-series (e.g. trial or run), and as such can be thought of as a volume with time as the third dimension.
Transversal slice	A slice taken through a spatial volume. For the example of a person, this would be like looking at them from above. As such, the slice index goes from inferior-superior, or bottom-to-top.
Trial (as in "single trial")	An experiment where only one task is performed, usually repeated several times.
Volume	A spatial volume of slices, usually spatially adjacent, with a time reference. It can be thought of as a 4-D set, with three spatial dimensions and one time dimension.
Voxel	An element of a spatial 3-D volume, usually with a time reference. It can be thought of as a 1-D time signal.

Appendix B

Functions

```
$Revision: 1.3 $
$Date: 2002/08/14 13:32:23 $
```

B.1 Available Functions

This appendix provides a list of all the functions that are available within the `Lyngby` package. This is meant for users who wish to write scripts to automatically process their data. Users of the GUI will not normally need to access these functions directly.

Help on each function can be obtained from the Matlab command line by typing:

```
>> help <functionname>
```

For instance, to obtain help on the "lyngby_getdata" function:

```
>> help lyngby_getdata
```

This will then return the following:

```
  lyngby_getdata       - Returns the masked datamatrix

        function X = lyngby_getdata(ROI, voxelMask, timeMask);

        Input:  ROI        (Optional) Region of Interests, 3x2 matrix
                voxelMask  (Optional) Sparse masking matrix
                timeMask   (Optional) Sparse masking matrix

        Output: X  The datamatrix

        This function loads the datamatrix. The voxel is in the
        horizontale direction (as columns) and the time is in the
        vertical direction (as rows).
        To save memory a 'Region of Interest' can be specified,
        and further a voxel or/and a time mask can be specified.
        If the input arguments are missing TIME_MASK, VOXEL_MASK and
        ROI_VOXELS are used.
```

```
        Which volumes is loaded is determined by global variables and
        through calls to consecutive calls lyngby_getvolume (for each
        volume).

        See also LYNGBY, LYNGBY_GETVOLUME, LYNGBY_UI_LOADFILE.

  $Id: function-appendix.tex,v 1.3 2002/08/14 13:32:23 fnielsen Exp $
```

The last line is the concurrent revision system (CVS) text. The first number indicates the revision number and the date is the date of the last revision.
You can also use the `helpwin` Matlab command:

```
>> helpwin('lyngby_getdata')
```

A new help window should popup, and you can click between related functions listed at the button next to "See also".

The following table lists all the functions available. Not that the list is not complete. Some will never need to be called directly as they will be called from other functions. All the functions with a "lyngby_ui_" stem are concerned with the GUI controls and as such are unlikely to be called from a script.

Table B.1: Functions Available in the `Lyngby` Toolbox

Function Name	Purpose
General `Lyngby` Functions	
lyngby	lyngby start, Calls lyngby_ui_main
lyngby_circle	Draw a circle
lyngby_classerr	Classification error
lyngby_corrlin	Lin's concordance correlation coefficient for reproducibility
lyngby_cumperiodo	Normalized cumulated periodogram
lyngby_cva	Canonical variate analysis (Canonical Ridge)
lyngby_design	n/a
lyngby_dropedge	Drop scans at the paradigm shifts
lyngby_filter	function for averaging volumes
lyngby_frame	Test function
lyngby_global	File defining global variables
lyngby_histeq	Perform histogram equalization
lyngby_image	Plot a slice
lyngby_init	Initialize the global variables
lyngby_kronadd	Kronecker add
lyngby_log	Log
lyngby_meangeo	Geometric mean
lyngby_meanhl	Hodges-Lehmann estimator
lyngby_mod	Modulus function
lyngby_msvd_main	Perform meta svd comparison of selected models.
lyngby_opls	Orthonormalized PLS
lyngby_plotmatch	Plot paradigm, data and model
Statistical Distributions	
lyngby_cdf_bin	Binomial distribution function
lyngby_cdf_chi2	$X\hat{2}$ cumulated distribution function
lyngby_cdf_gauss	Gaussian (normal) distribution function
lyngby_cdf_t	Student t distribution
lyngby_cdf_wilrank	"Wilcoxon sign rank" distribution
lyngby_fs_cdf	Returns the cumulative F disttribution
lyngby_fs_invcdf	Returns the inverse of the cumulative F dist.
lyngby_idf_chi2	Invers $X\hat{2}$ distribution function
lyngby_idf_t	Inverted Student t distribution
lyngby_pdf_bin	Binomial probability density function
lyngby_pdf_gauss	Gaussian (normal) density function
lyngby_pdf_poisson	Poisson probability density function
Data Loading and Writing	
lyngby_filefind	Find the position of a string in a file

lyngby_read_header	Reads header information of a volume file
lyngby_read_volume	Reads a volume from a recognized file
lyngby_readanahdr	Reads an ANALYZE header
lyngby_readanavol	Reads an ANALYZE img file
lyngby_readvahdr4	Reads the header information in a 4 dimensional VAPET file
lyngby_readvapethdr	Reads the header information in a VAPET file
lyngby_readvapetvol	Reads a VAPET volume from a file
lyngby_readvavol4	Reads a 4 dimensional VAPET structure from a file
lyngby_readxpihdr	Read header from EC flexible format
lyngby_readxpivol	Read data from EC flexible (Xprime) format
lyngby_vmask2index	Return Indices from voxel mask
lyngby_write_ana	Write a volume to ANALYZE files
lyngby_write_sdt	Write a volume/datamatrix to SDT/SPR file
lyngby_write_vapet	Write a volume to VAPET file
Data Formatting and Information	
lyngby_full2mask	Convert indices from Full volume to ROI volume
lyngby_getdata	Returns the masked datamatrix
lyngby_getinfo	Get volume info from a file
lyngby_getslice	Extracts a slice from a volume
lyngby_getvolume	Get a volume from a file
lyngby_index2tmask	Return time mask from indices
lyngby_index2vmask	Return voxel from indices
lyngby_mask2full	Convert indices from ROI to Full volume
lyngby_normalize	Normalize in vertical direction of a datamatrix
lyngby_normalize	Normalize (Preprocess) a datamatrix
lyngby_numscan	Returns the masked number of scans volume
lyngby_paradigm	Returns the paradigm
lyngby_prep_global	Global variables for preprocessing
lyngby_prep_init	Initialize global variables for preprocessing
lyngby_roi	Returns the begin and end index of the volume
lyngby_roi2vmask	Transform ROI definition to masking matrix
lyngby_run	Returns the run specification
lyngby_runinfo	Number of runs and scans within runs
lyngby_sliceindex	Returns indices for a specified slice
lyngby_swaporder	Swap the xyz order in a volume
lyngby_talaimg	Returns Talairach brain plot in the style of SPM
lyngby_tmask2index	Return time mask from indices
lyngby_tmp_files	Prints which temporary files was opened by lyngby
lyngby_tmp_getseq	Get sequence from temporary file of data matrix
lyngby_tmp_getvol	Get volume in temporary file for all voxels
lyngby_tmp_init	Initialize temp file for datamatrix (Read/Write)

lyngby_tmp_putseq	Store a sequence in a temporary file of data matrix
lyngby_tmp_putvol	Store volume in temporary file for all voxels
Matrix Operations	
lyngby_std	Take the standard deviation of a matrix
lyngby_test_wilrank	Wilcoxon sign rank test
Statistics Tests	
lyngby_ts_ftest	F-test for two samples.
lyngby_ts_global	Global variables for TS analysis method
lyngby_ts_init	Initialize globals for t-test analysis method
lyngby_ts_main	Whole volume Student's t-test for equal variances.
lyngby_ts_pttest	Student's paired t-test.
lyngby_ts_ttest	Two sample Student t test, equal variances.
lyngby_ts_uttest	Student's t-test for unequal variances.
lyngby_ui_ts_init	User interface for initialization of t-test
lyngby_ui_view_ts	t-test viewing function
lyngby_uis_ts	Script executed for t-test analysis
lyngby_uis_ts_v	Script for t-test view
General User Interface Functions	
lyngby_ui_credit	Display a credit
lyngby_ui_global	Defines global for the user interface
lyngby_ui_loadfile	Opens a window for specifying file parameters.
lyngby_ui_loadsetup	Opens a window for specifying file parameters.
lyngby_ui_main	Main function for the GUI for lyngby
lyngby_ui_message	Display a message
lyngby_ui_movie	User interface for movie
lyngby_ui_option	Option/setup for user interface
lyngby_ui_para_draw	User interface for paradigm viewing (and specification)
lyngby_ui_paradigm	User interface for paradigm viewing (and specification)
lyngby_ui_preproc	UI for setting up preprocessing parameters
lyngby_ui_resinfo	Defines global for the user interface
lyngby_ui_run	User interface for run viewing (and specification)
lyngby_ui_run_no	User interface for time mask specification
lyngby_ui_saveresult	Opens a window for specifying file parameters.
lyngby_ui_time	User interface for time mask specification
lyngby_ui_time_be	User interface for time mask specification
lyngby_ui_view	User interface for viewing results
lyngby_ui_view_dfl	Default viewing function
lyngby_ui_viewvol	View full volume timeserie and specify ROI and voxelmask
lyngby_ui_volume	View full volume timeserie and specify mask

lyngby_uis_none_v	Script for 'original' view
Analysis Functions and Modelling Algorithms	
Ardekani F-Test	
lyngby_baf_global	Global variables for Ardekani f-test analysis method
lyngby_baf_init	Initialize globals for Ardekani F-test
lyngby_baf_test	Barbak Ardekani's F-test
lyngby_ui_baf_init	Ardekani's F-test, GUI for init. of param.
lyngby_ui_view_baf	Viewing BAF filter results
lyngby_uis_baf	Script executed for Ardekani's F-test
lyngby_uis_baf_v	Script for Ardekani F-test view
Ardekani F-Test with Nuisance Subspace	
lyngby_baf2_global	Global variables for Ardekani nuisance signal f-test
lyngby_baf2_init	Initialize globals for Ardekani F-test with nuisance
lyngby_baf2_main	Barbak Ardekani's F-test with nuisance estimation
lyngby_baf2_test	Barbak Ardekani's F-test with nuisance estimation
lyngby_ui_baf2_init	Ardekani's nuisance F-test, init. of param.
lyngby_ui_view_baf2	Viewing function for Ardekani's nuisance F
lyngby_uis_baf2	Script executed for Ardekani's nuisance F
lyngby_uis_baf2_v	Script for Arkani's F-test with nuisance view
Ardekani T-Test	
lyngby_bat_global	Global variables for Ardekani t-test analysis method
lyngby_bat_init	Initialize globals for Ardekani t-test
lyngby_bat_test	t-test using the Ardekani t-method
lyngby_ui_bat_init	User interface for init. of Ardekani's t-test
lyngby_ui_view_bat	Viewing Ardekani t-test results
lyngby_uis_bat	Script executed for Ardekani's t-test
lyngby_uis_bat_v	Script for Ardekani't t-test view
Cross-correlation	
lyngby_cc_global	Global variables for cross-correlation analysis method
lyngby_xcorr	Cross correlation
lyngby_ui_cc_init	User interface for init. of cross-correlation
lyngby_ui_cc_view	User interface for viewing Cross-correlation results
lyngby_ui_view_cc	Viewing Cross-correlation results
lyngby_uis_cc	Script executed for cross-correlation anal.
lyngby_uis_cc_v	Script for cross-correlation view

Exhaustive FIR	
lyngby_efir_global	Global variables for Exhaustive FIR analysis method
lyngby_efir_init	Initialize global variables for EFIR analysis
lyngby_efir_main	Main exhautive FIR function
lyngby_ui_efir_init	UI for initialization of EFIR model
lyngby_ui_view_efir	Viewing EFIR filter results
lyngby_uis_efir	Script executed at the EFIR analysis
lyngby_uis_efir_v	Script for 'exhaustive FIR' viewing
lyngby_plotfilter	Plot FIR response
FIR	
lyngby_fir_convolve	Convolve the FIR kernel with the input
lyngby_fir_error	FIR residual signal (Quadratic error)
lyngby_fir_global	Global variables for FIR analysis method
lyngby_fir_init	Initialize global variables for FIR analysis
lyngby_fir_main	Regularized FIR filter, main function
lyngby_fir_masscent	Returns the masscenter of the FIR kernel
lyngby_fir_plot	Plot paradigm, data and FIR model
lyngby_ui_fir_init	User interface for FIR filter analysis
lyngby_ui_view_fir	Viewing FIR filter results
lyngby_uis_fir	Script executed at the FIR analysis
lyngby_uis_fir_v	Script for FIR view
lyngby_plotfilter	Plot FIR response
Gaussian Mixture Models	
lyngby_gmm_error	Gaussian mixture model, error
lyngby_gmm_fit	Gaussian mixture model, fitting
lyngby_gmm_plot	Plot gaussian mixture model
K-Means	
lyngby_km_error	K-means, error
lyngby_km_global	K-means, Global variables declarations
lyngby_km_init	K-means, Global variables initialization
lyngby_km_main	Main function for K-means clustering
lyngby_km_plot	K-means plot
lyngby_ui_km_init	GUI for K-means parameter initialization
lyngby_ui_view_km	Viewing K-means filter results
lyngby_uis_km	Script executed for K-means analysis
lyngby_uis_km_v	Script for K-means view
Kolmogorov-Smirnov	
lyngby_ks_global	Global variables for KS analysis method
lyngby_ks_init	Initialize globals for KS-test analysis

lyngby_ks_main	Kolmogorov Smirnov test
lyngby_ks_prob	Kolmogorov-Smirnov probability
lyngby_ks_test	Kolmogorov-Smirnov test of two arrays.
lyngby_ui_ks_init	KS, GUI for initialization of parameters
lyngby_ui_view_ks	Kolmogorov-Smirnov viewing function
lyngby_uis_ks	Script executed for Kolmogorov-Smirnoff
lyngby_uis_ks_v	Script for Kolmogorov-Smirnov view
Lange-Zeger	
lyngby_lz_error	Lange-Zeger residual signal (Quadratic error)
lyngby_lz_lambda	The Lange-Zeger hemodynamic response function
lyngby_lz_masscent	Returns the masscenter of the Lange Zeger Kernel
lyngby_lz_plot	Plot Paradigm, data and Lange-Zeger model
lyngby_lz_plottheta	Plot Lange-Zeger theta1 against theta2
Lange-Zeger Grid Search	
lyngby_lzgs_global	Global variables for LZ Grid Search analysis
lyngby_lzgs_init	Initialize global variables for LZGS
lyngby_lzgs_main	Lange-Zeger model with grid search
lyngby_lzgs_search	Lange Zeger main algorithm for grid search
lyngby_ui_lzgs_init	Lange-Zeger GUI init of parameters
lyngby_ui_view_lzgs	Viewing Lange Zeger results
lyngby_uis_lzgs	Script executed for Lange-Zeger Grid Search
lyngby_uis_lzgs_v	Script for Lange-Zeger gridsearch view
Lange-Zeger, Iterative	
lyngby_lzit_algo	Lange-Zeger, optimization of model
lyngby_lzit_beta	Lange-Zeger, Estimate beta parameter
lyngby_lzit_conv	Lange-Zeger, Convolve kernel with the input
lyngby_lzit_cost	The cost function used in the Lange-Zeger model
lyngby_lzit_ftlamb	Lange-Zeger, Fourier transformed of lambda
lyngby_lzit_global	Global variables for iterative LZ
lyngby_lzit_init	Initialize global variables for iterative LZ
lyngby_lzit_main	Main function for Lange-Zeger estimation
lyngby_lzit_noise	Noise estimate in the Lange-Zeger model
lyngby_lzit_pickf	Pick frequencies out from the data matrix
lyngby_lzit_thopt	Lange-Zeger, Optimize theta parameters
lyngby_ui_lzit_init	GUI for iterative Lange-Zeger initialization
lyngby_ui_view_lzit	Viewing Lange Zeger results
lyngby_uis_lzit	Script executed for Lange-Zeger, Iterative
lyngby_uis_lzit_v	Script for Lange-Zeger 'iterative' view
Meta K-Means	

lyngby_mkm_global	Meta K-means, Global variables declaration
lyngby_mkm_init	Meta K-means, Global variables initialization
lyngby_ui_mkm_init	GUI for Meta K-means initialization
lyngby_ui_view_mkm	Viewing of meta K-means results
lyngby_uis_mkm	Script executed for Meta K-means analysis
lyngby_uis_mkm_v	Script for Meta K-means viewing
Neural Network	
lyngby_nn_cddevds	2nd order, entropic, input, diag. sym.
lyngby_nn_cddewda	Classifier neural network, output, diag 2nd
lyngby_nn_cddewds	Classifier neural network, output, diag 2nd
lyngby_nn_cddru	2nd order der. of regularization for weights
lyngby_nn_cdev	Classifier neural network, input 1st der.
lyngby_nn_cdew	Classifier neural network, output 1st der.
lyngby_nn_cdru	Neural network, regularization, 1st der.
lyngby_nn_cerror	Classifier neural network error
lyngby_nn_cforward	Classifier neural network forward
lyngby_nn_cmain	Main functions for classifier neural network
lyngby_nn_cost	Cost function = error + regularization
lyngby_nn_cpruneobd	Classifier neural network, OBD pruning
lyngby_nn_csoftline	Classifier neural network soft linesearch
lyngby_nn_csoftmax	Softmax for neural network classifier
lyngby_nn_ctrain	Classifier neural network training
lyngby_nn_eddevds	2nd order, entropic, input, diag. sym.
lyngby_nn_eddewd	Entropic neural network, output, diag 2nd
lyngby_nn_eddru	2nd order der. of regularization for weights
lyngby_nn_edev	Entropic neural network, input 1st der.
lyngby_nn_edew	Entropic neural network, output, 1st der.
lyngby_nn_edru	First order derivative, Regularization
lyngby_nn_eehl	Neural network entropic error (Hodge-Lehmann)
lyngby_nn_eemedian	Neural network entropic error (median)
lyngby_nn_eerror	Calculate (cross-)entropic error
lyngby_nn_eforward	Entropic neural network, feed-forward
lyngby_nn_emain	Main function for Entropic Neural network
lyngby_nn_epruneobd	Pruning by Optimal Brain Damage (entropic)
lyngby_nn_esoftline	Soft linesearch with the entropic cost function
lyngby_nn_esoftline	Soft linesearch with the quadratic costfunction
lyngby_nn_etarget	Normalize target for entropic neural network
lyngby_nn_etrain	Entropic neural network, training
lyngby_nn_initvw	Initialize weights in neural network
lyngby_nn_plotnet	Neural network, plot the network weights
lyngby_nn_pruneobd	Pruning by Optimal Brain Damage
lyngby_nn_qddeus	2nd der., quad., all weights, gauss approx.

lyngby_nn_qddevds	2nd der., quadratic, input, diag gauss.
lyngby_nn_qddevs	2nd der., quadratic, input, gauss approx.
lyngby_nn_qddewds	2nd order der, quadratic, output, diag gauss
lyngby_nn_qddews	2nd. order der., quadratic, output, gauss approx.
lyngby_nn_qddru	2nd order der. of regularization for weights
lyngby_nn_qdev	Quadratic neural network, input, 1st der.
lyngby_nn_qdew	First order derivative, quadratic, Output weights
lyngby_nn_qdru	Quadratic neural network, 1st der. reg.
lyngby_nn_qerror	Quadratic error
lyngby_nn_qforward	Neural network forward with linear output function
lyngby_nn_qmain	Main function for quadratic neural network
lyngby_nn_qpruneobd	Pruning by Optimal Brain Damage (quadratic)
lyngby_nn_qtrain	Quadratic neural network, training
lyngby_nn_reg2reg	Standardize regularization
lyngby_nn_setsplit	Split the data into training and validation set
lyngby_nn_tanh	Faster hyperbolic tangent
lyngby_nn_u2vw	Vectorize neural network weights
lyngby_nn_vw2u	Vectorize neural network weights
Neural Network Regression	
lyngby_nnr_global	Global variables for NNR analysis method
lyngby_nnr_init	Initialize global variables for NNR analysis
lyngby_nnr_main	Main function for Neural network Regression
lyngby_ui_nnr_init	User interface for NNR method
lyngby_ui_view_nnr	Viewing NNR filter results
lyngby_uis_nnr	Script executed for Neural network regression
lyngby_uis_nnr_v	Script for neural network regression view
Neural Network Saliency	
lyngby_nns_esalmap	Saliency map with entropic cost function
lyngby_nns_global	Global variables for NNS analysis
lyngby_nns_init	Initialize global variables for NNS analysis
lyngby_nns_main	Main function for Neural network saliency
lyngby_ui_nns_init	GUI for Neural network saliency
lyngby_ui_view_nns	Viewing of NNS results
lyngby_uis_nns	Script executed for Neural network saliency
lyngby_uis_nns_v	Script for neural network saliency view
PCA Analysis	
lyngby_pca_eqtest	PCA test for equal eigenvalues
lyngby_pca_main	Main PCA analysis
lyngby_pcafilte	Reduce the dimension or filter a matrix with PCA filtering

Poisson	
lyngby_pois_error	Poisson residual signal (Quadratic error)
lyngby_pois_forward	Possion kernel forward (prediction)
lyngby_pois_forwn	Possion kernel forward (prediction)
lyngby_pois_global	Poisson filter, Global variables
lyngby_pois_init	Poisson filter, Initialize global variables
lyngby_pois_main	Main function for Possion filter
lyngby_pois_toptim	Possion kernel optimization
lyngby_ui_pois_init	GUI, Poisson filter, initialization
lyngby_ui_view_pois	Viewing poisson filter results
lyngby_uis_pois	Script executed for Poisson filter analysis
lyngby_uis_pois_v	Script for Poisson filter view
SCVA	
lyngby_scva_global	Global variables for SCVA analysis
lyngby_scva_init	Initialize global variables for SCVA analysis
lyngby_scva_main	Strother Canonical variate analysis
lyngby_ui_scva_init	GUI for init of Strother CVA
lyngby_ui_view_scva	Viewing function for Strother CVA
lyngby_uis_scva	Script executed for SCVA analysis
lyngby_uis_scva_v	Script for SCVA view
lyngby_sdesign	Strother design matrix
lyngby_set_and	set intersection: S1 / S2
lyngby_set_diff	set difference: S1 S2
lyngby_set_unique	set thining: all elements different
SOP	
lyngby_sop_global	Global variables for SOP analysis
lyngby_sop_init	Initialize global variables for SOP analysis
lyngby_sop_main	Strother OPLS
lyngby_sopressvd	SOP with SVD on the residual
lyngby_ui_sop_init	GUI of init. of SOP model
lyngby_ui_view_sop	Viewing for Strother OPLS
lyngby_uis_sop	Script executed for SOP analysis
lyngby_uis_sop_v	Script for SOP view
lyngby_splitonnoff	Function to find the indices of the on/off scans.
SVD	
lyngby_svd	Singular value decomposition
lyngby_svd_global	Global variables for SVD analysis
lyngby_svd_init	Initialize global variables for SVD analysis
lyngby_svdfilter	Filtering by SVD
lyngby_ui_svd_init	GUI for initialization of SVD model

lyngby_ui_view_svd	Viewing for Strother OPLS
lyngby_uis_svd	Script executed for SVD analysis
lyngby_uis_svd_v	Script for SVD view

Appendix C

Derivations

$Revision: 1.5 $
$Date: 2002/08/14 13:17:05 $

C.1 Neural Network

C.1.1 Symbol table

δ Kronecker's delta

ϕ Output of the neural network *before* the softmax is applied

E_c Errorfunction for the classifier neural network

h Index for hidden units

i Index for input units

n_o Number of outputs

n_p Number of patterns, i.e., examples or scans

o Index for output unit

p Index for pattern

$p(\ldots)$ Probability

$\mathcal{T}$ Training set

t Target output for the neural network

u Weights (parameters) in the neural network

v Input layer weights (first layer weights)

w Output layer weights (second layer weights)

x Input for the neural network

y (Predicted) Output of the neural network (*after* the softmax is applied)

z The hidden unit value after the activation function has been applied

In the following the index for pattern has been omitted, so that what should have been written as x_i^p, z_h^p, ϕ_o^p, y_o^p and t_o^p (or in an equivalent notation) is written as x_i, z_h, ϕ_o, y_o and t_o.

C.1.2 The errorfunction for the classifier neural network and its derivatives

The errorfunction for the classifier neural network is developed from the conditional probability:

$$p(\mathcal{T}|u) = \prod_p^{n_p} p(\mathbf{y}|\mathbf{x}, u) = \prod_p^{n_p} \prod_o^{n_o} y_o^{t_o} \tag{C.1}$$

The error function is constructed as the normalized negative log-likelihood of the parameters u and with the two-layer feedforward neural network with an added softmax layer and a 1-of-$(c-1)$ coding scheme — or rather 1-of-(n_o-1) coding scheme — we get:

$$\begin{aligned}
E_c &= -\frac{1}{n_p} \ln[p(\mathcal{T}|u)] && \text{(C.2)}\\
&= -\frac{1}{n_p} \sum^{n_p} \ln \prod_o^{n_o} y_o^{t_o} && \text{(C.3)}\\
&= -\frac{1}{n_p} \sum^{n_p} \sum_o^{n_o} t_o \ln y_o && \text{(C.4)}\\
&= -\frac{1}{n_p} \sum^{n_p} \left[t_{n_o} \ln \left(1 - \sum_o^{n_o-1} y_o \right) + \sum_o^{n_o-1} t_o \ln \left(\frac{\exp(\phi_o)}{\sum_{o'}^{n_o} \exp(\phi_{o'}) + 1} \right) \right] && \text{(C.5)}\\
&= && \text{(C.6)}\\
&= -\frac{1}{n_p} \sum^{n_p} \left[t_o \ln \left[1 - \sum_o^{n_o-1} \frac{\exp(\phi_o)}{\sum_{o'}^{n_o} \exp(\phi_{o'}) + 1} \right] + \sum_o^{n_o-1} t_o \phi_o \right. \\
&\qquad \left. - \sum_o^{n_o-1} t_o \ln \left[\sum_{o'}^{n_o} \exp(\phi_{o'}) + 1 \right] \right] && \text{(C.7)}\\
&= -\frac{1}{n_p} \sum^{n_p} \left[t_o \ln \left[\frac{1}{\sum_o^{n_o} \exp(\phi_o) + 1} \right] + \sum_o^{n_o-1} t_o \phi_o - \sum_o^{n_o-1} t_o \ln \left[\sum_{o'}^{n_o} \exp(\phi_{o'}) + 1 \right] \right] && \text{(C.8)}\\
&= \frac{1}{n_p} \sum^{n_p} \left[\sum_o^{n_o} t_o \ln \left[\sum_{o'}^{n_o-1} \exp(\phi_{o'}) + 1 \right] - \sum_o^{n_o-1} t_o \phi_o \right] && \text{(C.9)}\\
&= \frac{1}{n_p} \sum^{n_p} \left[\ln \left[\sum_o^{n_o-1} \exp(\phi_o) + 1 \right] - \sum_o^{n_o-1} t_o \phi_o \right] && \text{(C.10)}
\end{aligned}$$

In order to optimize the neural network with gradient or newton methods the derivatives have to be found. The first order derivative of the classifier neural network errorfunction with

respect to the weights is:

$$\frac{\partial E_c}{\partial u} = \frac{\partial}{\partial u}\left[\frac{1}{n_p}\sum^{n_p}\left[\ln\left[\sum_o^{n_o-1}\exp(\phi_o)+1\right]-\sum_o^{n_o-1}t_o\phi_o\right]\right] \tag{C.11}$$

$$= \frac{1}{n_p}\sum^{n_p}\left[\frac{1}{1+\sum_{o'}^{n_o-1}\exp(\phi_{o'})}\sum_o^{n_o-1}\exp(\phi_o)\frac{\partial\phi_o}{\partial u}-\sum_o^{n_o-1}t_o\frac{\phi_o}{\partial u}\right] \tag{C.12}$$

$$= \frac{1}{n_p}\sum^{n_p}\left[\sum_o^{n_o-1}\frac{\exp(\phi_o)}{1+\sum_{o'}^{n_o-1}\exp(\phi_{o'})}\frac{\partial\phi_o}{\partial u}-\sum_o^{n_o-1}t_o\frac{\phi_o}{\partial u}\right] \tag{C.13}$$

$$= \frac{1}{n_p}\sum^{n_p}\sum_o^{n_o-1}(y_o-t_o)\frac{\partial\phi_o}{\partial u} \tag{C.14}$$

The second order derivative of the classifier neural network errorfunction with respect to the weight is:

$$\frac{\partial^2 E_c}{\partial u_2\partial u_1} = \frac{\partial}{\partial u_2}\left[\frac{1}{n_p}\sum^{n_p}\sum_o^{n_o-1}(y_o-t_o)\frac{\partial\phi_o}{\partial u_1}\right] \tag{C.15}$$

$$= \frac{1}{n_p}\sum^{n_p}\sum_o^{n_o-1}\left[(y_o-t_o)\frac{\partial^2\phi_o}{\partial u_2\partial u_1}+\frac{\partial\phi_o}{\partial u_1}\frac{\partial y_o}{\partial u_2}\right] \tag{C.16}$$

For the first order derivative we only need to compute the derivative for the two ordinary layer ($\partial\phi/\partial u$) — not the softmax layer. However, for the second order derivative we need to have the derivative through the softmax:

$$\frac{\partial y_o}{\partial u} = \frac{\partial}{\partial u}\left[\frac{\exp(\phi_o)}{\sum_{o'}^{n_o-1}\exp(\phi_{o'})+1}\right] \tag{C.17}$$

$$= \frac{\exp(\phi_o)}{\sum_{o'}^{n_o-1}\exp(\phi_{o'})+1}\frac{\partial\phi_o}{\partial u}+\exp(\phi_o)\sum_{o'}^{n_o-1}\frac{\exp(\phi_{o'})}{\left(\sum_{o''}^{n_o-1}\exp(\phi_{o''})+1\right)^2}\frac{\partial\phi_{o'}}{\partial u} \tag{C.18}$$

$$= y_o\frac{\partial\phi_o}{\partial u}+\sum_{o'}^{n_o-1}\frac{\exp\phi_o\exp(\phi_{o'})}{\left(\sum_{o''}^{n_o-1}\exp(\phi_{o''})+1\right)^2}\frac{\partial\phi_{o'}}{\partial u} \tag{C.19}$$

$$= \sum_{o'}^{n_o-1}\delta_{oo'}y'_o\frac{\partial\phi'_o}{\partial u}+\sum_{o'}^{n_o-1}y_oy_{o'}\frac{\partial\phi_{o'}}{\partial u} \tag{C.20}$$

$$= \sum_{o'}^{n_o-1}y_{o'}(\delta_{oo'}-y_o)\frac{\partial\phi_{o'}}{\partial u} \tag{C.21}$$

We can now use equation C.21 in C.16 arriving at:

$$\frac{\partial^2 E_c}{\partial u_2\partial u_1} = \frac{1}{n_p}\sum^{n_p}\sum_o^{n_o-1}\left[(y_o-t_o)\frac{\partial^2\phi_o}{\partial u_2\partial u_1}+\sum_{o'}^{n_o-1}y_{o'}(\delta_{oo'}-y_o)\frac{\partial\phi_{o'}}{\partial u_2}\frac{\partial\phi_o}{\partial u_1}\right] \tag{C.22}$$

The partial derivatives for the ϕ output are the same as for the quadratic neural network. For the output weights w we have:

$$\frac{\partial\phi_o}{\partial w_{oh}} = z_h \tag{C.23}$$

Combining equations C.14 and C.23 makes the derivative for the errorfunction to:

$$\frac{\partial E_c}{\partial w_{oh}} = \frac{1}{n_p}\sum^{n_p}(y_o - t_o)\, z_h \tag{C.24}$$

For the input weights v we have:

$$\frac{\partial \phi_o}{\partial v_{hi}} = w_{oh}\left(1 - z_h^2\right) x_i \tag{C.25}$$

$$\frac{\partial E_c}{\partial v_{hi}} = \frac{1}{n_p}\sum^{n_p}(y_o - t_o)\, w_{oh}\left(1 - z_h^2\right) x_i \tag{C.26}$$

The second order derivative — the Hessian — is more complex, and it can unfortunately/luckily be approximated in a number of ways: using only the Gauss-Newton term and diagonalizing it. The derivative for the output weights is relatively simple:

$$\frac{\partial^2 \phi_{o_1 o_2}}{\partial w_{o_1 h_1} \partial w_{o_2 h_2}} = z_{h_1} z_{h_2} \tag{C.27}$$

$$\frac{\partial^2 E_c}{\partial w_{o_1 z_1} \partial w_{o_2 h_2}} = \frac{1}{n_p}\sum^{n_p}\left[\delta_{o_1 o_2}\,(y_o - t_o)\, z_{h_1} z_{h_2} + y_{o_1}(\delta_{o_1 o_2} - y_{o_2}) z_{h_1} z_{h_2}\right] \tag{C.28}$$

$$= \frac{1}{n_p}\sum^{n_p}\left[(-y_{o_1} y_{o_2} + 2\delta_{o_1 o_2} y_o - \delta_{o_1 o_2} t_o)\, z_{h_1} z_{h_2}\right] \tag{C.29}$$

The diagonal approximation is

$$\frac{\partial^2 E_c}{\partial w_{oh}^2} = \frac{1}{n_p}\sum^{n_p}\left[\left(-y_o^2 + 2y_o - t_o\right) z_h^2\right] \tag{C.30}$$

For the input weights we have:

$$\frac{\partial^2 \phi}{\partial v_{h_2 i_2} \partial v_{h_1 i_1}} = \frac{\partial}{\partial v_{h_2 i_2}}\left[w_{oh_1}\left(1 - (z_{h_1})^2\right) x_{i_1}\right] \tag{C.31}$$

$$= -w_{oh_1} x_{i_1} 2 z_{h_1}\left(1 - z_{h_1}^2\right)\delta_{h_1 h_2} x_{i_2} \tag{C.32}$$

$$= -w_{oh}\delta_{h_1 h_2} x_{i_1} x_{i_2} 2 z_h\left(1 - z_h^2\right) \tag{C.33}$$

The derivative for the errorfunction is:

$$\frac{\partial^2 E_c}{\partial v_{h_1 i_1} \partial v_{h_2 i_2}} = \frac{1}{n_p} \sum^{n_p} \sum_{o}^{n_o} \Bigg[(y_o - t_o)(-w_{oh}) \delta_{h_1 h_2} x_{i_1} x_{i_2} 2 z_h \left(1 - z_h^2\right) \tag{C.34}$$

$$+ \sum_{o'}^{n_o} y_{o'} \left(\delta_{oo'} - y_o\right) w_{o'h_2} \left(1 - z_{h_2}^2\right) x_{i_2} w_{oh_1} \left(1 - z_{h_1}^2\right) x_{i_1} \Bigg] \tag{C.35}$$

$$= \frac{1}{n_p} \sum^{n_p} x_{i_1} x_{i_2} \sum_{o}^{n_o} \Bigg[(y_o - t_o)(-w_{oh}) \delta_{h_1 h_2} 2 z_h \left(1 - z_h^2\right) \tag{C.36}$$

$$+ \sum_{o'}^{n_o} y_{o'} \left(\delta_{oo'} - y_o\right) w_{o'h_2} \left(1 - z_{h_2}^2\right) w_{oh_1} \left(1 - z_{h_1}^2\right) \Bigg] \tag{C.37}$$

$$= \frac{1}{n_p} \sum^{n_p} x_{i_1} x_{i_2} \sum_{o}^{n_o} \sum_{o'}^{n_o} \Bigg[2 \delta_{oo'} \delta_{h_1 h_2} (y_o - t_o)(-w_{oh}) z_h \left(1 - z_h^2\right) \tag{C.38}$$

$$+ y_{o'} \left(\delta_{oo'} - y_o\right) w_{o'h_2} \left(1 - z_{h_2}^2\right) w_{oh_1} \left(1 - z_{h_1}^2\right) \Bigg] \tag{C.39}$$

With diagonal approximation the derivative becomes:

$$\frac{\partial^2 E_c}{\partial v_{hi}^2} = \frac{1}{n_p} \sum^{n_p} x_i^2 \sum_{o}^{n_o} \sum_{o'}^{n_o} \Bigg[2 \delta_{oo'} (y_o - t_o)(-w_{oh}) z_h \left(1 - z_h^2\right) \tag{C.40}$$

$$+ y_{o'} \left(\delta_{oo'} - y_o\right) w_{o'h} w_{oh} \left(1 - z_h^2\right)^2 \Bigg] \tag{C.41}$$

C.1.3 Numerical stabil computation of softmax

The softmax function has an exponential divided by an exponential which makes it prone to congest the range of the data type and divide `+Inf` with `+Inf`. For 64 bit `double` data type this happens a bit over 700:

```
>> log(realmax)

ans =

  709.7827
```

To cope with this problem the softmax function can be reformulated:

$$\frac{\exp(\phi_o)}{\sum_o^{n_o} \exp(\phi_{o'}) + 1} = \frac{\exp(\phi_o)}{\sum_o^{n_o} \exp(\phi_{o'}) + 1} \frac{\exp(k)}{\exp(k)} = \frac{\exp(\phi_o + k)}{\sum_o^{n_o} \exp(\phi_{o'} + k) + \exp(k)} \tag{C.42}$$

And by setting $k = -\max(\phi)$ we can make sure that the exponential in the numerator does not saturate against `+Inf`, — and it does not matter that the denominator saturate against `-Inf` for computer with IEEE arithmetic since `exp(-Inf)` yields `0`.

Appendix D

Getting Started

$Revision: 1.3 $
$Date: 2002/08/14 13:03:16 $

Table D.1: Installed, Up and Running in Two Minutes!

Keyboard/mouse action	Explanation
Installation	
Click on the link to the toolbox file (lyngby.tar.gz). Choose a suitable place to save the file. Save the sample dataset (sampledata.tar.gz) in the same way.	Download the lyngby Toolbox and the sample dataset from the website at: http://hendrix.imm.dtu.dk-/software/lyngby/.
`> cp lyngby.tar.gz /usr/local/`	Copy the file to a suitable location, such as /usr/local, where a program directory will be created.
`> gunzip lyngby.tar.gz`	Uncompress the file.
`> tar -xvf lyngby.tar`	Unpack the file, which automatically creates a "lyngby" directory.
`> rm lyngby.tar.gz`	Remove the original distribution file to save space.
`> cp sampledata.tar.gz /usr/local/lyngby`	Copy the sample dataset to the lyngby directory.
`> gunzip sampledata.tar.gz`	Uncompress the sample dataset.
`> tar -xvf sampledata.tar`	Unpack the sample dataset, automatically creating a "sampledata" directory in the process.

`> rm sampledata.tar.gz`	Remove the original distribution file to save space.
Setup	

`> emacs startup.m`
Add the line:
path(path, '/usr/local/lyngby');
	Edit your Matlab "startup.m" file to inform Matlab where it can find the lyngby executable files.

Starting lyngby	
`> matlab`	Start Matlab as normal
`>> cd /usr/local/lyngby/sampledata`	Within Matlab, change to the directory containing the sample data.
`>> lyngby`	Start lyngby by simply typing lyngby from the command prompt. The lyngby main window should pop-up.

Appendix E

Example Analysis

```
$Revision: 1.2 $
$Date: 2002/08/14 13:08:46 $
```

How To Load and Analyse Data Using the Sample Dataset.

Table E.1: A Simple Example Analysis - How To Use The `Lyngby` GUI

Keyboard/mouse action	Explanation
Press [Load new data...]. A new window titled *Load data* will pop-up.	First, we need to load the sample dataset. The data parameters should all be set automatically, as lyngby will have read-in a conversion file specifying them.
Press the [Apply] button at the bottom of the new window	The values of the GUI will be written to global variables needed for the loading.
Press the [Load Data!] button. The window will close.	The data will now be read in from the files. In the main window it should display "Finished loading data!" at the bottom.
Press the [Create/Edit External Influences...] button bringing up a new window	This is were you, e.g., can specify a stimulus function.
Press [Setup design]	The first pre-processing step is to mask out those unwanted timepoints from the paradigm and run structures.
Press [Process design]	Next, the mean is removed from the paradigm structure (i.e. the paradigm signal is transformed from 0, 1, 0, 1, 0, 1... to -1/2, +1/2, -1/2, +1/2, -1/2, +1/2...) as required by some analysis algorithms.

Press Close. The window should disappear	
Press the Data setup... button in the main window. A new window titled *Data Setup* will pop-up.	The next stage is to perform any pre-processing of the data.
If not already highlighted, select the first options ("normalization") on the left-hand pane, and press "Setup design and run". A new window ("Pre-Processing") will appear.	The next stage is to select the pre-processing methods that will be applied to the data.
Select the first and third option and press the Apply Pre-Processing and Close button.	The *Pre-Proceesing* window will close, and the status of the pre-processing will be shown on the status bar within the main window.
Press Done! in the *Data setup* window.	This will close the window
Click on the top button of the *Data Analysis* frame, initally labeled *Original*. From the pop-up list, select the FIR Filter algorithm.	Next, the actual analysis of the data can take place. As an example, we will do a FIR Filter analysis of the paradigm and the fMRI data.
Press the Parameters... button to bring up a new window. Once you have finished examining the settings, press the Close button to shut the window.	The initial parameters for the FIR Filter can be set within a dedicated window. The *Parameters* window will be different for each algorithm, reflecting the different variables used in each case. Note that each algorithm will have a sensible set of defaults within its *Parameters* window. For this example, we can use default settings.
Press the Calculate button to start the calculation.	The calculation is then started, with feedback given on the status line within the main window. Note that once the calculation is finished, the symbol adjacent to the algorithm name changes to a "+", indicating that results for this algorithm are ready to be examined. A "-" indicates that no calculation has been performed, whilst a "!" indicates that the parameters have been changed, but not yet used in a calculation.

Press the View these results button. A set of three windows, labelled *Control*, *Volume* and *Time* will pop-up.	Once the calculations have finished (as shown in the status line), then the results can be viewed.
Click on different voxels in the *Volume* window. Note how the *Time* window displays the relevant time-series.	The control window initially shows two "layers". The lower one controls the right-hand time-based window, while the upper one is used to control the left-hand volume-based window. The *Time* window shows the time-series for whichever voxel is selected in the *Volume* window. The current voxel location is shown above the time-series in the form [x,y,z]. Note how the data in the top-left of the image [x = 32-34, y = 1, z = 34] correlates better with the paradigm than that at the bottom-right [x = 14-20, z = 40-45]. Note also how the FIR model (red-line) matches the data far more closely in these highly-activated regions.
Click on Sum of Coefficients in the Data Layer, and pick *Activation Strength* from the pop-up list.	For the FIR filter, there are several results for both the volume and time windows. The *Activation Strength* shows better contrast between the activated and non-activated regions.
Click on Time in the Time Layer, selecting the *Histogram of Data* option. The Time window will change from displaying a time-series to showing a histogram of the greylevels for the selected voxel.	The time display can also display other types of data that are voxel-dependent. For instance, the FIR filter algorithm generates a range of results displaying histograms of the greyscales of a voxel through the time-series.
Click on different voxels and note that the general shape of the histogram is different for the activated and non-activated areas.	The different areas in the volume have different greyscale distributions. Note that the distribution in activated areas has a bimodal distribution, reflecting the two distinct grey-levels that occur during the activated and non-activated states, whilst the distribution of non-activated areas is generally monomodal.

Click on the More... button to expand the control window to display an additional two rows. The upper row contains two new visual layers, *Contour* and *Background*, whilst the *Masking Layer* underneath is an intermediate processing layer that affects the data layer below it. See Fig. 2.11 for a graphical explanation.	The toolbox can also display concurrent spatial information by the use of *overlays* - layers that sit above or below the data.
In the Masking Layer, click on the second button from the left (showing None), and select the ">" option.	Now we can mask the Data layer by using the Masking layer above. This creates a mask from a given dataset and then applies it to the dataset in the Data layer.
Click on the Sum of Coefficients button in the Masking layer and select *Activation Strength* from the pop-up list.	We will choose to define the mask using the *Activation Strength* dataset, although any of the others could be used to give different masks.
Using the slider control in the Masking layer, adjust the threshold to around 0.96. You can use the edit box immediately to the left to type in 0.96 if you prefer.	We can now adjust the size of the mask by adjusting the thresholding limit. This can be done by specifying either an absolute or fractile value. We'll use the latter, which is the default.
In the Background Layer, click on the first button to the right of the *Background* label. This turns the layer on. Now click the Sum of Coefficients button and select the *Mean (data)* dataset to display the anatomical data.	It would be more useful if we could now see this thresholded data overlaid onto the original data.
Click on the button immediately to the right of the *Contour* label to activate the contour layer. Then click on the Sum of Coefficients button and select the *Activation strength* dataset from the pop-up list.	To see how far this threshold is from the rest of the data, we can now overlay a contour layer of the full *Activation strength* dataset. Note how you can now see that if you decreased the mask threshold, then the peak at [x = 31, y = 1, z = 27] will be the next to show up. The contour layer can also be used to compare different variables from the same result set. This allows a comparison of methods for highlighting the activated regions.

In the main window, below the Analysis pane, is the Post-processing section. Meta-K-means should already be selected. Click on the Parameters button to bring up the options dialog box. Make sure that the *Parameter Variables Set* is set to the FIR filter results. The other initial settings should should already be sensible. Click Close once you have finished.	Next, we can cluster the actual parameters of the FIR filter - a Meta K-means clustering of the results. This enables us to look for patterns in the result dataset.
Click on the Calculate button.	The post-processing analysis can then be started, with feedback on the progress given on the status line.
Once the calculation is finished, click on the View results button to bring up the triple-set of viewing windows.	The results are viewed in the same way as the main analysis results.
Click on the Time button in the Time layer and select the *Cluster mean seq.* from the pop-up list. The Time window will now show an additional red line representing how the mean of the cluster that the selected voxel is a member of varies with time.	The data can then be analysed using the same techniques as for the main anaylsis results. The Volume window shows the voxels in their correct locations, but with the colour indicating their cluster membership. Note how the voxels near the upper left of the image [around x = 31, y = 1, z = 34] are members of the same cluster, and that the mean of this cluster most closely follows the time series.
In the main window, click on the Save Worksheet button in the seventh pane. The entire workspace is then saved into the file `lyngby_workspace.mat`, located in the directory from which lyngby was started.	Once the results have been analysed, you will probably want to save them. In Matlab, you are able to save the entire variable and result set into a single file which can then be reloaded at a later date without having to re-specify all the data loading parameters.
Click on the Save Data... button to bring up the standard four layers plus the extra Save layer	Alternatively, you may want to save a subset of the results. The toolbox has a new save layer which can be used for this purpose.
Click on the first button on the Save layer and select the *Sequence* option. On the second button, select the *Matlab Binary* option. Type in a filename into the edit box and then press the Save here!.	As an example, we will save the time-sequence for the current voxel. Other options allow you to save the present slice, or the entire volume.

Click on the Exit! button and the lyngby main window will disappear.	Once you have finished using the GUI, just close it. All the variables will still be accessible from the command line.
On the command-line, type: `>> lyngby_ui_global`	To make the results and variables visible to the command-line you will need to make them all global. You can also do this whilst the GUI is still open.
In the command-line, type: `>> whos`	This will then display all the variables as used in the calculations. You can now access the individual results and variables.

Bibliography

Aguirre, G. K., Zarahn, E., and D'Esposito, M. (1998). A critique of the use of the Kolmogorov-Smirnov (KS) statistic for the analysis of BOLD fMRI data. *Magnetic Resonance in Medicine*, 39:500–505.

Alldredge, J. R. and Gilb, N. S. (1976). Ridge regression: An annotated bibliography. *International Statistical Review*, 44(3):355–360.

Andersen, L. N., Larsen, J., Hansen, L. K., and Hintz-Madsen, M. (1997). Adaptive regularization of neural classifiers. In Principe, J., editor, *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing VII, Florida, USA*, pages 24–33, Piscataway, New Jersey. IEEE.

Ardekani, B. A. and Kanno, I. (1998). Statistical methods for detecting activated regions in functional MRI of the brain. *Magnetic Resonance in Imaging*, 16(10):1217–1225.

Ardekani, B. A., Kershaw, J., Kashikura, K., and Kanno, I. (1999). Activation detection in functional MRI using subspace modeling and maximum likelihood estimation. *IEEE Transaction on Medical Imaging*, 18(2):101–114.

Belliveau, J., Fox, P., Kennedy, D., Rosen, B., and Ungeleider, L., editors (1996). *Second International Conference on Functional Mapping of the Human Brain*, volume 3. Academic Press.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK.

Fletcher, P. C., Dolan, R. J., Shallice, T., Frith, C. D., Franckowiak, R. S. J., and Friston, K. J. (1996). Is multivariate analysis of PET data more revealing than the univariate approach? Evidence from a study of episodic memory retrieval. *NeuroImage*, 3:209–215.

Friberg, L., Gjedde, A., Holm, S., Lassen, N. A., and Nowak, M., editors (1997). *Third International Conference on Functional Mapping of the Human Brain, NeuroImage*, volume 5. Academic Press.

Friston, K. J., Frith, C. D., Frackowiak, R. S. J., and Turner, R. (1995a). Characterizing dynamic brain responses with fMRI: A multivariate approach. *NeuroImage*, 2(2):166–172.

Friston, K. J., Holmes, A. P., Worsley, K. J., Poline, J.-B., Frith, C. D., and Frackowiak, R. S. J. (1995b). Statistical parametric maps in functional imaging: A general linear approach. *Human Brain Mapping*, 2(4):189–210.

Friston, K. J., Jezzard, P., and Turner, R. (1994). The analysis of functional MRI time-series. *Human Brain Mapping*, 1(2):153–171.

Gold, S., Christian, B., Arndt, S., Zeien, G., Cizadlo, T., Johnson, D. L., Flaum, M., and Andreasen, N. C. (1998). Functional MRI statistical software packages: A comparative analysis. *Human Brain Mapping*, 6(2):73–84.

Goutte, C., Nielsen, F. Å., and Hansen, L. K. (2000). Modeling the haemodynamic response in fMRI using smooth FIR filters. *IEEE Transactions on Medical Imaging*, 19(12):1188–1201.

Goutte, C., Nielsen, F. Å., Svarer, C., Rostrup, E., and Hansen, L. K. (1998). Space-time analysis of fMRI by feature space clustering. In Paus, T., Gjedde, A., and Evans, A. C., editors, *Fourth International Conference on Functional Mapping of the Human Brain. NeuroImage*, volume 7, page S610. Academic Press.

Goutte, C., Toft, P., Rostrup, E., Nielsen, F. Å., and Hansen, L. K. (1999). On clustering fMRI time series. *NeuroImage*, 9(3):298–310.

Hansen, P. C. (1996). *Rank-Deficient and Discrete Ill-Posed Problems.* Polyteknisk Forlag, Lyngby, Denmark. Doctoral Dissertation.

Hartigan, J. A. and Wong, M. A. (1979). Algorithm AS136. a K-means clustering algorithm. *Applied Statistics*, 28:100–108.

Haykin, S. (1994). *Neural Networks.* Macmillan College Publishing Company, New York.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, Califonia, 1st edition. Santa Fe Institute.

Hintz-Madsen, M., Hansen, L. K., Larsen, J., Olesen, E., and Drzewiechi, K. T. (1996a). Detection of malignant melanoma using neural classifiers. In A.B. Bulsari, S. K. and Tsaptsinos, D., editors, *Proceedings of International Conference on Engineerings Applications on Neural Networks*, pages 395–398.

Hintz-Madsen, M., Hansen, L. K., Larsen, J., Olesen, E., and Drzewiecki, K. T. (1995). Design and evaluation of neural classifiers, application to skin lesion classification. In *IEEE Workshop on Neural Networks for Signal Processing.* NNSP'95.

Hintz-Madsen, M., Pedersen, M. W., Hansen, L. K., and Larsen, J. (1996b). Design and evaluation of neural classifiers. In S. Usui, Y. Tohkura, S. K. and Wilson, E., editors, *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing VI*, pages 223–232.

Hoerl, A. E. and Kennard, R. W. (1970a). Ridge regression: Application to nonorthogonal problems. *Technometrics*, 12(1):69–82.

Hoerl, A. E. and Kennard, R. W. (1970b). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Jackson, J. E. (1991). *A user's guide to principal components.* Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. John Wiley & Sons, New York.

Lange, N., Hansen, L. K., Pedersen, M. W., Savoy, R. L., and Strother, S. C. (1996). A concordance correlation coefficient reproducibility of spatial activation patterns. In (Belliveau et al., 1996), page S75.

Lange, N. and Zeger, S. L. (1997). Non-linear fourier time series analysis for human brain mapping by functional magnetic resonance imaging (with discussion). *Applied Statistics, Journal of the Royal Statistical Society, Series C*, 46(1):1–29.

Le Cun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems: Proceedings of the 1989 Conference*, pages 598–605, San Mateo, CA. Morgan-Kaufmann. NIPS-2.

Liptrot, M., Adams, K. H., Matiny, L., Pinborg, L. H., Lonsdale, M. N., Olesen, N. V., Holm, S., Svarer, C., and Knudsen, G. M. (2003). Cluster analysis in kinetic modelling of the brain: A noninvasive alternative to arterial sampling. *NeuroImage*. In press.

Locascio, J. J., Jennings, P. J., Moore, C. I., and Corkin, S. (1997). Time series analysis in the time domain and resampling methods for studies of functional magnetic resonance brain imaging. *Human Brain Mapping*, 5(5):168–193.

Lundsager, B. and Kristensen, B. L. (1996). Lineær og ulineær modellering af positron emission tomografier. Master's thesis, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark. In Danish.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In Le Cam, L. M. and Neyman, J., editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, Califonia. University of California Press.

Mardia, K. V., Kent, J. T., and Bibby, J. M. (1979). *Multivariate Analysis*. Probability and Mathematical Statistics. Academic Press, London.

McIntosh, A. R., Bookstein, F. L., Haxby, J. V., and Grady, C. L. (1996). Spatial pattern analysis of functional brain images using Partial Least Square. *NeuroImage*, 3(3 part 1):143–157.

Mørch, N. J. S., Hansen, L. K., Strother, S. C., Law, I., Svarer, C., Lautrup, B., Anderson, J. R., Lange, N., and Paulson, O. B. (1996). Generalization performance of nonlinear vs. linear models for [^{15}O]water PET functional activation studies. In (Belliveau et al., 1996), page S258.

Mørch, N. J. S., Kjems, U., Hansen, L. K., Svarer, C., Law, I., Lautrup, B., Strother, S. C., and Rehm, K. (1995). Visualization of neural networks using saliency maps. In *Proceedings of 1995 IEEE International Conference on Neural Networks*, volume 4, pages 2085–2090.

Mørch, N. J. S. and Thomsen, J. R. (1994). Statistisk analyse af positron emission tomografier. Master's thesis, Electronics Institute, Technical University of Denmark, Lyngby, Denmark. In danish.

Nielsen, F. Å. (1996). Visualization and analysis of 3D functional brain images. Master's thesis, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark.

Nielsen, F. Å., Hansen, L. K., Toft, P., Goutte, C., Mørch, N. J. S., Svarer, C., Savoy, R., Rosen, B. R., Rostrup, E., and Born, P. (1997). Comparison of two convolution models for fMRI time series. In (Friberg et al., 1997), page S473.

Oppenheim, A. V. and Schafer, R. W. (1989). *Discrete-Time Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, United Kingdom.

Sonka, M., Hlavac, V., and Boyle, R. (1993). *Image, Processing, Analysis and Machine Vision*. Chapman & Hall Computing Series. Chapman & Hall, London, United Kingdom.

Strother, S. C., Lange, N., Savoy, R., Anderson, J. R., Sidtis, J. J., Hansen, L. K., Bandettini, P. A., O'Craven, K., Rezza, M., Rosen, B. R., and Rottenberg, D. A. (1996). Multidimensional state-spaces for fMRI and PET activation studies. In (Belliveau et al., 1996), page S98.

Strupp, J. P. (1996). Stimulate: A GUI based fMRI analysis software package. In (Belliveau et al., 1996), page S607.

Svarer, C., Hansen, L. K., and Larsen, J. (1993a). On the design and evaluation of tapped-delay lines neural networks. In *Proceedings of the IEEE International Conference on Neural Networks, San Francisco, California, USA*, volume 1, pages 46–51.

Svarer, C., Hansen, L. K., Larsen, J., and Rasmussen, C. E. (1993b). Designer networks for time series processing. In Kamm, C. et al., editors, *Neural Networks for Signal Processing III*, pages 78–87, Piscataway, NJ. (Baltimore 1993), IEEE.

Tikhonov, A. N. (1963). Solution of incorrectly formulated problems and the regularization method. *Soviet Math. Dokl.*, 4:1035–1038.

Tikhonov, A. N. and Arsenin, V. Y. (1977). *Solution of Ill-Posed Problems*. Winston, Washington D. C.

Toft, P., Hansen, L. K., Nielsen, F. Å., Strother, S. C., Lange, N., Mørch, N. J. S., Svarer, C., Paulson, O. B., Savoy, R., Rosen, B. R., Rostrup, E., and Born, P. (1997). On clustering of fMRI time series. In (Friberg et al., 1997), page S456.

Van Horn, J. D., Esposito, G., Weinberger, D. R., and Berman, K. F. (1996). Volume-based multivariate discriminant and canonical correlation analysis of neurophysiological measures of brain function. In (Belliveau et al., 1996), page S103.

Vinod, H. D. (1976). Canonical ridge and econometrics of joint production. *Journal of Econometrics*, 4:147–166.

Worsley, K. J. (1997). An overview and some new developments in the statistical analysis of PET and fMRI data. *Human Brain Mapping*, 5(4):254–258.

Worsley, K. J., Poline, J.-B., Friston, K. J., and Evans, A. C. (1997). Characterizing the response of PET and fMRI data using multivariate linear models. *NeuroImage*, 6(4):305–319.