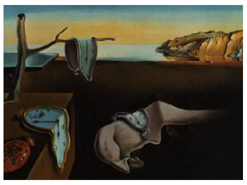

Journal of Statistical Software

October 2011, Volume 44, Issue 4. *http://www.jstatsoft.org/*

cudaBayesreg: Parallel Implementation of a Bayesian Multilevel Model for fMRI Data Analysis

Adelino R. Ferreira da Silva
Universidade Nova de Lisboa

Abstract

Graphic processing units (GPUs) are rapidly gaining maturity as powerful general parallel computing devices. A key feature in the development of modern GPUs has been the advancement of the programming model and programming tools. Compute Unified Device Architecture (CUDA) is a software platform for massively parallel high-performance computing on Nvidia many-core GPUs. In functional magnetic resonance imaging (fMRI), the volume of the data to be processed, and the type of statistical analysis to perform call for high-performance computing strategies. In this work, we present the main features of the R-CUDA package **cudaBayesreg** which implements in CUDA the core of a Bayesian multilevel model for the analysis of brain fMRI data. The statistical model implements a Gibbs sampler for multilevel/hierarchical linear models with a normal prior. The main contribution for the increased performance comes from the use of separate threads for fitting the linear regression model at each voxel in parallel. The R-CUDA implementation of the Bayesian model proposed here has been able to reduce significantly the run-time processing of Markov chain Monte Carlo (MCMC) simulations used in Bayesian fMRI data analyses. Presently, **cudaBayesreg** is only configured for Linux systems with Nvidia CUDA support.

Keywords: Bayesian multilevel methods, fMRI, R, GPU, CUDA.

1. Introduction

Currently, the statistical method used by the vast majority of functional magnetic resonance imaging (fMRI) data researchers and neuroscientists is the general linear model (GLM, Lazar 2008). The GLM procedure is often said to be 'massively univariate', since data for each voxel are independently fit with the same model. In this paper, we adopt Bayesian methodologies for the statistics that comprise GLM inference. However, since (non-variational) Bayesian models draw on Markov chain Monte Carlo (MCMC) simulations, Bayesian estimates involve a heavy computational burden. The volume of the data to be processed and the type of statistical analysis to perform in fMRI analysis call for high-performance computing strategies.

Traditionally, parallel processing has been performed on shared-memory systems with multiple CPUs, or on distributed-memory clusters made up of smaller shared-memory systems or single-CPU systems. However, using these systems for high-performance computing suffers from serious limitations. They are difficult to manage, require vast resources, do not scale well due to communication and/or synchronization constraints, and require specialized parallel programming expertise. A less costly alternative is to use multicore processors. Today's multicore architectures are making scalability more affordable. The current microprocessor development effort in today's computer architectures is to increase the number of cores in order to maintain performance growth. This trend comes from the necessity to overcome the physical constraints on CPU frequency growth and high power consumption. Virtually all present day CPU processors are multicore processors. Therefore, parallel computing is becoming a new trend in mainstream computing. However, CPUs are optimized for high performance on sequential code, and use multiple-instructions-multiple-data (MIMD) architectures designed to extracting instruction-level parallelism. On the other hand, graphics hardware performance is increasing more rapidly than that of CPUs. The highly data-parallel nature of graphics computations enables graphics processing units to achieve higher arithmetic intensity (floating-point horsepower) with the same transistor count. It is thus not surprising that considerable efforts have been made to harness the tremendous power of GPUs, enabling them to function as general parallel computing devices for a wide range of applications (Owens *et al.* 2007). The programmable GPU has evolved from a graphics engine into a powerful highly parallel, multithreaded, manycore processor. GPUs allow advanced scientific and engineering applications to scale transparently to hundreds of processor cores and thousands of concurrent threads.

Modern graphic processing units (GPUs) are built around a scalable array of multithreaded streaming multiprocessors (SMs). Compute Unified Device Architecture (CUDA) (Nvidia Corporation 2010b), is a software platform for massively parallel high-performance computing on Nvidia many-core GPUs. Just as important in the widespread use of the GPU as a general-purpose computing engine has been the advancement of software development tools (Owens *et al.* 2008). Current GPU implementations enable scheduling thousands of concurrently executing threads. However, without proper hardware abstraction mechanisms and software development tools, parallel programming becomes extremely challenging. The CUDA programming model follows the standard single-program multiple-data (SPMD) model. CUDA greatly simplifies the task of parallel programming by providing thread management tools that work as extensions of conventional C/C++ constructions. Automatic thread management removes the burden of handling the scheduling of thousands of lightweight threads, and enables straightforward programming of the GPU cores. Finally, the wide availability of CUDA tools in inexpensive laptops and desktops are favouring the rapid development of GPU parallel computing applications (Fatahalian and Houston 2008).

The purpose of the present paper is to highlight the main features of the R package **cudaBayesreg** (Ferreira da Silva 2011b), available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=cudaBayesreg`. The package implements a parallel Bayesian multilevel model for the analysis of brain fMRI data written in the R system for statistical computing (R Development Core Team 2011), with an interface to C-CUDA procedures. The package serves a twofold purpose. First, the Bayesian multilevel model overcomes several limitations of the classical SPM methodology (Ferreira da Silva 2011a). Second, the proposed framework significantly reduces the run-time processing of MCMC simulations (see Section 5). The main contribution for the increased performance comes from the use of separate threads for fitting the linear regression model at each voxel in parallel. Apart from the MCMC simulation itself, all other pre-processing and post-processing functions in **cudaBayesreg** are implemented in R. Therefore, the package contributes to establish a research environment dedicated to the analysis of fMRI experiments, and benefits from algorithmic implementations already available in the R environment. In particular, **cudaBayesreg** functions depend on the R package **oro.nifti** (Whitcher *et al.* 2011a) for input/output of NIFTI formatted fMRI data sets. In a similar vein, the R packages **bayesm** (Rossi 2011), **boa** (Smith 2007), and **fmri** (Tabelow and Polzehl 2011) could profitably be used in conjunction with **cudaBayesreg** to further process or analyse fMRI data.

2. Bayesian multilevel modelling

2.1. Multivariate regression

Consider a multivariate regression model for the fMRI time series in which the regression equations are related through common predictor variables, and the errors are correlated across equations. For a general linear model fit at a set of m voxels we have,

$$y_i = X\beta_i + \epsilon_i, \quad \epsilon_i \overset{iid}{\sim} N(0, \sigma^2 I_n), \quad i = 1, \ldots, m, \tag{1}$$

where y_i is a vector of n time series observations for voxel i, X is a matrix of predictor variables, β_i is a vector of unknown parameters, and ϵ_i are unknown error vectors with the Normal distribution $N(0, \sigma^2 I_n)$. In matrix notation, model (1) assumes the equivalent standard multivariate regression form,

$$Y = XB + U, \quad u_j \overset{iid}{\sim} N(0, \Sigma), \quad j = 1, \ldots, n, \tag{2}$$

where Y is a $n \times m$ matrix of observations, X is a $n \times k$ matrix on k independent variables, B is $k \times m$, with each column of B containing the regression coefficients for one of the m equations, and U is a $n \times m$ matrix of random disturbances whose rows u_j for given X are not autocorrelated. There is, however, contemporaneous correlation between corresponding errors in different equations, with mean 0 and common variance-covariance matrix Σ. Model (2) involves the study of several regressions taken simultaneously, because the disturbance terms in the several regressions are mutually correlated. Thus, information in one regression can be used to estimate coefficients in other regressions. The standard multivariate regression model assumes that the regression equations are related through common X variables, and that the errors are correlated across equations (contemporaneous correlation). In statistics, when X is a design matrix, model (2) is called the general linear model (Mardia *et al.* 1979).

In Bayesian inference for the linear model (Box and Tiao 1973; Judge *et al.* 1988; Gelman 2006; Rossi *et al.* 2005), the standard priors for model (2) are the natural conjugate priors (multivariate Normal-Wishart prior),

$$\begin{aligned} p(B, \Sigma) &= p(\Sigma)p(B|\Sigma), \\ \Sigma &\sim \mathcal{IW}(\nu, V), \\ \beta = vec(B)|\Sigma &\sim N(vec(\bar{B}), \Sigma \otimes A^{-1}), \end{aligned} \tag{3}$$

where $\mathcal{IW}(\nu, V)$ is the expression for an inverted Wishart density with degrees of freedom ν and scale matrix V, which is used for the prior on Σ, and A is the $k \times k$ precision matrix specified for the prior on β. In (3), $\bar{\beta} = vec(\bar{B})$ is the unknown prior mean vector of the regression parameters β, which is then used to draw the posterior estimates specified in (4) and (5). In (3), $vec(\cdot)$ is the operator which transforms a matrix into a vector by stacking the columns of the matrix one underneath the other, and $\otimes$ means the Kronecker product of two matrices (Judge *et al.* 1988).

Simulations from the multivariate regression model (2) can be obtained by considering the posterior as the product of the conjugate priors and the conditional Normal likelihood, $p(B, \Sigma)p(Y|X, B, \Sigma)$. Expressions for the posterior density of the multivariate regression model have been derived by several authors, e.g., Lindley and Smith (1972); Box and Tiao (1973); Press (2003); Rossi *et al.* (2005). In this work, we follow the notation and derivations used in the last of these references. To draw from the posterior, we first draw Σ and then draw B given Σ,

$$\begin{aligned} \Sigma|Y, X &\sim \mathcal{IW}(\nu + n, V + S), \\ \beta|Y, X, \Sigma &\sim N\big(\tilde{\beta}, \Sigma \otimes (X^\top X + A)^{-1}\big), \end{aligned} \tag{4}$$

where

$$\begin{aligned} \tilde{\beta} &= vec(\tilde{B}), \\ \tilde{B} &= (X^\top X + A)^{-1}(X^\top Y + A\bar{B}), \\ S &= (Y - X\tilde{B})^\top (Y - X\tilde{B}) + (\tilde{B} - \bar{B})^\top A(\tilde{B} - \bar{B}). \end{aligned} \tag{5}$$

and $X^\top$ means the transpose of X.

2.2. Multilevel modeling

Now, we generalize the standard linear model presented in Section 2.1 in two directions. Firstly, we lift the restriction on the common structure of the regression equations to improve estimation efficiency. We introduce a multivariate regression prior to estimate correlations between the regression coefficient vectors. Secondly, we present an empirical Bayes approach for second-stage priors to obviate the difficulties with prior elicitation.

In fMRI data analysis, the disturbances in the voxel regression equations at a given time are likely to reflect some common unmeasurable factors or structure, and hence are correlated. By taking into account the correlation structure of the disturbances across voxel equations, and by jointly estimating the regression equations, it is generally possible to obtain more precise estimates. We may improve estimation by pooling time series and cross-sectional fMRI data, assuming cross-sectional voxels with different coefficient vectors. In econometrics, a widely used method for joint regression estimation is the seemingly unrelated regression (SUR) method proposed in (Zellner 1962), which can be regarded as a generalization of the standard linear model (Geweke 2005). An alternative approach has been proposed in (Rossi *et al.* 2005).

Consider the general linear model (1) with different regressors in each equation, and a different error variance for each voxel,

$$y_i = X_i\beta_i + \epsilon_i, \quad \epsilon_i \overset{iid}{\sim} N(0, \sigma_i^2 I_{n_i}), \quad i = 1, \ldots, m. \tag{6}$$

In order to tie together the voxels' regression equations, we assume that the $\{\beta_i\}$ have a common prior distribution. To build the Bayesian regression model we need to specify a prior on the $\{\beta_i\}$ coefficients, and a prior on the regression error variances $\{\sigma_i^2\}$. Following (Rossi *et al.* 2005), we specify a Normal regression prior with mean $\Delta^\top z_i$ for each β_i,

$$\beta_i = \Delta^\top z_i + \eta_i, \quad \eta_i \overset{iid}{\sim} N(0, V_\beta), \tag{7}$$

where z is a vector of n_z elements, representing characteristics of each of the m regression equations. A special case of (7) is to consider a common mean vector for all betas by doing $z_i = \mathbf{1}$ and centering the matrix Δ. The prior (7) can be written using the matrix form of the multivariate regression model for k regression coefficients,

$$B = Z\Delta + V, \quad B = \begin{bmatrix} \beta_1^\top \\ \vdots \\ \beta_m^\top \end{bmatrix}, \ Z = \begin{bmatrix} z_1^\top \\ \vdots \\ z_m^\top \end{bmatrix}, \ V = \begin{bmatrix} \eta_1^\top \\ \vdots \\ \eta_m^\top \end{bmatrix}, \ \Delta = [\delta_1 \ldots \delta_k], \tag{8}$$

where B and V are $m \times k$ matrices, Z is a $m \times n_z$ matrix, Δ is a $n_z \times k$ matrix. Interestingly, the prior (8) assumes the form of a second-stage regression, where each column of Δ has coefficients which describe how the mean of the k regression coefficients varies as a function of the variables in z. In (8), Z assumes the role of a prior design matrix.

Assuming that each of the error variances is independent, a commonly used prior for the regression error variances $\{\sigma_i^2\}$ is the standard inverse gamma with parameters $a = \nu_i/2$ and $b = (\nu_i s_i^2)/2$, where $\nu_i s_i^2 = (y_i - X_i\hat{\beta}_i)^\top(y_i - X_i\hat{\beta}_i)$, $\nu_i = n - k$, and $\hat{\beta}_i = (X_i^\top X_i)^{-1} X_i^\top y_i$ (Box and Tiao 1973). In terms of the relationship between the inverse gamma form and the inverse of a chi-square random variable,

$$\sigma_i^2 \sim \frac{\nu_i s_i^2}{\chi^2_{\nu_i}}. \tag{9}$$

In order to alleviate the difficulties with the assessment of the priors in (8), we specify a second-stage of priors on Δ and V_β. As in (3), we specify natural conjugate priors for the multivariate regression model (8),

$$\begin{aligned} V_\beta &\sim \mathcal{IW}(\nu, V), \\ vec(\Delta)|V_\beta &\sim N(vec(\bar{\Delta}), V_\beta \otimes A^{-1}). \end{aligned} \tag{10}$$

In summary, the proposed model can be written down as a sequence of conditional distributions,

$$\begin{aligned} &y_i|X_i, \beta_i, \sigma_i^2 \\ &\beta_i|z_i, \Delta, V_\beta \\ &\sigma_i^2|\nu_i, s_i^2 \\ &V_\beta|\nu, V \\ &\Delta|V_\beta, \bar{\Delta}, A. \end{aligned} \tag{11}$$

The prior on the set of regression coefficients β is specified in two stages. First, we specify a Normal prior on β, and then we specify a second-stage prior on the parameters of this distribution. From a practical point of view, the key feature of model (11) is that it converts the problem of assessing a prior on the $(m \times k)$-dimensional joint distribution of the β_i into the problem of assessing hyperparameters $\bar{\Delta}, A, \nu$, and V. At each stage, the prior parameters are being projected onto lower-dimensional subspaces.

Model (6) is formulated in a generalized form, which allows for the specification of different design matrices in parcelled brain areas, or regions of interest, for instance. However, the present implementation in **cudaBayesreg**, and the examples reported in Section 5, use the same design matrix X for the whole brain ($X_i \equiv X, \forall i$), as commonly practised in standard GLM approaches.

3. GPU computation

Modern GPUs are well-suited to address problems that can be expressed as data-parallel computations (Nickolls *et al.* 2008). CUDA's parallel programming model provides three key abstractions: a hierarchy of thread groups, shared memories, and barrier synchronization (Nvidia Corporation 2010a,b). These abstractions are encapsulated in a minimal set of language extensions, allowing developers to define C functions, called kernels, that are executed N times in parallel by N different CUDA threads. A kernel executes in parallel across a set of parallel threads. The task of the programmer is to specify how these threads are organized in a hierarchy of grids of thread blocks. Once the dimensions of a grid and its thread blocks when launching a kernel have been specified, parallel execution and thread management will be automatic. The system automatically manages the tasks of thread creation, scheduling, and termination. This programming model is typically much simpler to implement than writing traditional parallel code.

The maximum number of concurrent, coresident threads on the GPU may be calculated by multiplying the maximum of resident threads per multiprocessor on the GPU by the number of multiprocessors. The program `deviceQuery`, distributed by Nvidia as part of the SDK toolkit ("NVIDIA_GPU_Computing_SDK") (Nvidia Corporation 2010b), can be used to query the characteristics of the available GPU in the system. However, the most important point is not so much this "hard thread partition", but rather the "logical thread partition" in terms of the kernel parameters. The user can specify a much higher number of threads to be allocated to the kernel via the kernel parameters. The CUDA environment transparently and automatically manages and allocates the resources to be handled by the kernel, according to the type of GPU available in the system. This feature is known as "automatic scalability". If the system runs out of resources, the kernel threads are scheduled for sequential execution. It is important to note that these threads, as opposed to CPU threads, are extremely lightweight, with negligible context switches. Typically, thousands of threads are queued up for work, in warps of 32 threads each. If the GPU must wait on one warp of threads, it simply begins executing work on another. Separate registers are allocated to all active threads. Therefore, no swapping of registers or state need occur between GPU threads (Nvidia Corporation 2010a). There are several other considerations that may limit the number of concurrently executing threads, namely the core allocated memory. Therefore, specifying a grid of thread blocks of dimension 64, for instance, represents a balance between maximum number of threads per block, and available resources. This number may be calibrated by the programmer for optimization purposes, depending on the problem at hand.

We investigated the application of the CUDA programming model in parallelizing fMRI data analysis. The multilevel Gibbs sampler model specified in Section 2 was implemented in

CUDA. Bayesian techniques typically rely on MCMC simulations for sampling from the posterior distribution of the parameters. However, MCMC simulations for large data sets are computationally demanding. Using CUDA, high-performance gains compared to strictly sequential analyses have been achieved, enabling us to significantly reduce the time complexity of computing for Bayesian inferences. The computational model has been specified as a grid of thread blocks of dimension 64, in which a separate thread is used for fitting a linear regression model at each voxel in parallel. Maximum efficiency is expected to be achieved when the total number of required threads to execute in parallel equals the number of voxels in the fMRI data set, after appropriate masking has been done. However, this approach typically calls for the parallel execution of several thousands of threads. To keep computational resources low, while maintaining significant high efficiency it is generally preferable to process fMRI data slice-by-slice. In this approach, slices are processed in sequence. Voxels in slices are processed in parallel. Thus, for slices of dimension 64×64, the required number of parallel executing threads does not exceed 4096 at a time.

The main computational bottleneck in sequential code comes from the necessity of performing Gibbs sampling, using a univariate regression model for all voxels time series. We coded this part of the MCMC computation as device code, i.e., a kernel to be executed by the `CUDA threads`. CUDA threads execute on the GPU device that operates as a coprocessor to the host running the MCMC simulation. Following the model presented in Section 2, each thread implements a Gibbs sampler to draw from posterior of a univariate regression with a conditionally conjugate prior. The host code is responsible for controlling the MCMC simulation. At each iteration, the threads perform one Gibbs iteration for all voxels in parallel, to draw the threads' estimators for the regression coefficients β_i as specified in (11). In turn, the host, based on the simulated β_i values, draws from the posterior of a multivariate regression model to estimate V_β and Δ (see (11)). These values are then used to drive the next iteration.

One important aspect of the device code simulation relates to the random number generation (RNG) process. We have to ensure that different threads do not generate the same sequence of random numbers and use different seeds, even if the number of threads is large. For this purpose, we implemented random number generation in device code. The package includes three optional CUDA-based RNGs. Marsaglia's multicarry RNG (Marsaglia 2003) follows the R implementation, is the fastest one, and is used by default; Brent's RNG (Brent 2006) has higher quality but is not-so-fast; Matsumoto's Mersenne Twister (Matsumoto and Nishimura 1998) is slow. In addition, we have to ensure that different threads receive different random seeds. We generated random seeds for the threads by combining random seeds generated by the host with the threads' unique identification numbers. Random deviates from the Normal (Gaussian) distribution and χ^2 distribution had to be implemented in device code as well. Random deviates from the Normal distribution were generated using the Box-Muller method. In a similar vein, random deviates from the χ^2 distribution with ν number of degrees of freedom, $\chi^2(\nu)$, were generated from Gamma deviates, $\Gamma(\nu/2, 1/2)$, following the method of Marsaglia and Tsang specified in (Press *et al.* 2007).

Additional design considerations underlying the CUDA implementation in **cudaBayesreg**, and the options taken for processing fMRI data in parallel have appeared recently in (Ferreira da Silva 2010).

4. Program installation

To use **cudaBayesreg** the following requirements must be met:

1. CUDA-capable GPU and CUDA software, which includes the `nvcc` (release 3.1 or higher) Nvidia CUDA Compiler driver (available at no cost from `http://www.nvidia.com/cuda`);

2. A supported version of Linux with `gcc/g++` GNU compiler (`nvcc` releases before version 3.1 are not fully compatible with current versions of `gcc/g++`);

3. The R system for statistical computing (R Development Core Team 2011).

The package's `configure.ac` file tests the environment variables `R_HOME` and `CUDA_HOME`, for detecting the root locations of R and CUDA installations, respectively. It is advisable to have these environment variables set for package installation. To reconfigure the source `Makefile` with different options, one may edit `configure.ac` and run `autoconf` followed by `configure` in the shell. The package requires the library `libRmath` that comes with the R-devel version of the R release. In case this library is not installed in the system, the user should install the R development version R-devel, available from the R Subversion repository. Alternatively, the user can install the `R-devel.tar.gz` tarball for her/his specific Linux distribution. A third alternative, is to build the library as standalone, as detailed in the manual for the 'R Installation and Administration' (R Development Core Team 2011), and adapt the file `Makefile` in the package directory `src`, accordingly.

Presently, the CUDA runtime API has some tough restrictions on device dynamic memory allocation. The currently version of **cudaBayesreg** does not use dynamic memory allocation on the device. Instead, the source file `cudaMultireg.cu` uses three constants which control the amount of CUDA allocated space: REGDIM = 4096, OBSDIM = 128, and XLIM = 5. These constants impose limits on the maximum number of parallel regressions, on the maximum length of voxel time-series, and on the maximum number of regression variables used in the simulations, respectively. These limits entail very low hardware requirements. In fact, all the tests described in the paper were performed on a standard off-the-shelf notebook equipped with a Nvidia "GeForce 8400M GS" card with just 2 multiprocessors. This device has Compute Capability 1.1 (Nvidia Corporation 2010b), and delivers single-precision performance. The CUDA installation should be tested by running the examples in the Nvidia SDK toolkit, before using **cudaBayesreg**.

5. Using cudaBayesreg

The next sections provide details on how to use **cudaBayesreg** for fMRI data analysis. For demonstration purposes, two fMRI data sets are included in the R package **cudaBayesregData** (Ferreira da Silva 2011c). For convenience, data sets for the examples used in **cudaBayesreg** have been separated from the main package. The two experiments reported in this work use the fMRI volumes included in **cudaBayesregData**. The fMRI volume `fmri.nii.gz` for the first experiment, may be retrieved from the FMRIB/FSL site (`http://www.fmrib.ox.ac.uk/fsl/`). The raw fMRI volume for the second experiment, may be retrieved from the **SPM** site (`http://www.fil.ion.ucl.ac.uk/spm/`, MoAEpilot example epoch (block)

Pre-stats	Stats: EVs
Motion correction: MCFLIRT	Number of original EVs: 2
B0 unwarping: No	Basic shape: Square
Slice Timing correction: None	Convolution: Gamma
Spatial smoothing FWHM (mm): 5	Orthogonalize: No
Intensity normalization: No	Add temporal derivative: Yes
Temporal Filtering: Highpass	Apply temporal filtering: Yes

Table 1: **FSL/FEAT** parameters used in the example `fmri`.

fMRI data set). In both cases, the data sets were preprocessed using the default parameter settings used in those software packages. This orientation has the following advantages: (a) reproducible research is facilitated, since the potential user does not have to care for ad-hoc parameterizations, (b) the results of the application of the proposed approach is directly seen without additional filtering or regularization effects.

The data set `fmri.nii.gz` is from an auditory-visual experiment. Auditory stimulation was applied as an alternating "boxcar" with 45s-on-45s-off and visual stimulation was applied as an alternating "boxcar" with 30s-on-30s-off. The data set includes just 45 time-points and 5 slices from the original 4D data. The file `fmri_filtered_func_data.nii` included in the package was obtained from `fmri.nii.gz` by applying **FSL/FEAT** pre-processing tools. We have followed the indications for data preparation published by the FMRIB Centre. The main operations involved in data preparation are motion correction and highpass filtering. In this case, the FSL/BET tool for brain extraction is not directly used, as we only have a few slices of data. However, since the application of the **FEAT** tool generates a mask data set from the main structural image, we have included this mask in **cudaBayesregData**. It turns out that the reduced number of slices produces an enlarged, poorly fit mask. The result is a mask that extends well beyond the brain area proper. As a consequence, artifacts tend to appear on the border of the brain areas. These artifacts could be removed by calibrating the mask size. Nevertheless, in the `fmri` example we decided to use the originally generated mask, and default test conditions. The design matrix file `fmri_design.txt` in **cudaBayesregData** was generated by the **FSL/FEAT** tool as well, assuming a Gamma Hemodynamic Response Function (HRF), and a design with temporal derivatives. Table 1, presents a summary of the main parameters used in data preparation, and model setup.

The file `swrfM_filtered_func_data.nii.gz` is a pre-processed volume in NIFTI format of an auditory fMRI data set reported in the **SPM** manual ('MoAEpilot example') (Ashburner *et al.* 2008). The original data set has been analysed by several researchers and is often used as a reference. The data set comprises whole brain BOLD/EPI images, acquired as successive blocks alternating between rest and auditory stimulation, starting with rest. Auditory stimulation was bi-syllabic words presented binaurally at a rate of 60 per minute. Each acquisition consisted of 64 contiguous slices for each volume. The auditory data set was pre-processed by the **SPM** software for realignment, co-registration and brain extraction, following the procedures outlined in (Ashburner *et al.* 2008). All pre-processing step were executed with the parameterizations specified in the **SPM** manual for this specific data set. The package's data directory also includes mask files associated with the partition of the auditory data set (prefixed by `fbase = "swrfM"`), in 3 classes: cerebrospinal fluid (CSF), grey matter (GM) and white matter (WM).

We performed MCMC simulations on these data sets using three types of code implementations for the Bayesian multilevel model specified before: a (sequential) R language version, a (sequential) C language version, and a CUDA implementation. Comparative runtimes for 3000 iterations in these three situations, for the data sets `fmri` and `swrfM`, are as follows.

```
Runtimes in  seconds for 3000 iterations:

      slice R-code C-code CUDA
fmri      3   1327    224   22
swrfM    21   2534    309   41
```

Speed-up factors between the sequential versions and the parallel CUDA implementation are summarized next.

```
Comparative speedup factors:

      C-vs-R CUDA-vs-C CUDA-vs-R
fmri     6.0      10.0      60.0
swrfM    8.2       7.5      61.8
```

In these tests, the C implementation provided, approximately, a 7.6$\times$ mean speedup factor relative to the equivalent R implementation. The CUDA implementation provided a 8.8$\times$ mean speedup factor relative to the equivalent C implementation. Overall, the CUDA implementation yielded a significant 60$\times$ speedup factor. The tests were performed on a notebook equipped with a (low-end) graphics card: a 'GeForce 8400M GS' Nvidia device. This GPU device has just 2 multiprocessors, *Compute Capability* 1.1, and delivers single-precision performance. The compiler flags used in compiling the source code are detailed in the package's `Makefile`. In particular, the optimization flag `-O3` is set there.

The function `read.fmrislice()` is the main function used for reading fMRI information to be processed by **cudaBayesreg**. This function expects three pieces of information to be available: (a) a pre-processed fMRI data set in gzipped NIFTI format; (b) a data set specifying the mask to be used, in gzipped NIFTI format; (c) a file in **FSL/FEAT** design format (`design.mat`) defining the design matrix X.

In the current version of **cudaBayesreg** (version 0.3-12), the argument `fbase` in the function `read.fmrislice()` enables the user to handle user defined data sets, as well as to process the example data sets included in the complementary package **cudaBayesregData**. If `fbase` is left unspecified (default `NULL`), then user data sets need to be provided as input. Otherwise, `fbase` specifies the data set prefix of one of the two demo fMRI data sets to use. User specified data files must have the names generated by the **FSL/FEAT** pre-processing tool, namely `filtered_func_data.nii.gz`, `mask.nii.gz`, and `design.mat`. The file `filtered_func_data.nii.gz` specifies the data set to be analyzed, `mask.nii.gz` specifies the data set to be used as mask, and `design.mat` specifies the data file to be used as design matrix. Typically, these data sets may be obtained using the **FSL/FEAT** pre-processing tool, or other similar tool like **SPM**. In **cudaBayesreg** versions 0.3-10+, `read.fmrislice()` uses the `design.mat` format from **FSL/FEAT**. The **FSL**-design format `design.mat` is simply an ASCII textfile comprising the fields `/NumWaves`, `/NumPoints`, `/PPheights`, and `/Matrix` (please refer to the package documentation for further information).

When specified, the prefix `fbase` applies to the demo data files *fbase*`_filtered_func.nii.gz`, *fbase*`_mask.nii.gz`, and *fbase*`_design.mat`. Two test data sets are included in **cudaBayesregData**: one with prefix `fmri`, the other with prefix `swrfM`. The function `read.Zsegslice()` builds the Z matrix of the statistical model, based on the brain segmented regions CSF/GM/WM for a given fMRI dataset. As for `read.fmrislice()`, the **FSL** tools may be used to obtain the segmented masks. If `fbase` has been left unspecified in `read.fmrislice()`, then three user specified segmented datasets in gzipped NIFTI format must be provided with the names `csf.nii.gz`, `gry.nii.gz`, and `wht.nii.gz`. Otherwise, `fbase` indicates the dataset prefix of one of the three segmented mask provided for the group effects example in **cudaBayesregData**. In all these cases, the R package **oro.nifti** (Whitcher *et al.* 2011a,b) is required for reading gzipped NIFTI files.

5.1. Posterior probability maps

In **cudaBayesreg**, fMRI volume data is processed on a slice-by-slice basis. The function `cudaMultireg.slice()` is the main function which provides the interface to the CUDA implementation of the Bayesian multilevel model for the analysis of brain fMRI data. This function processes a single slice in a fMRI data volume. The following code runs `R = 2000` iterations of the MCMC simulation for slice 3 of the `fmri` data set, and saves the result. Slice data and slice mask data are read by the function `read.fmrislice()`. The function `premask()` applies a pre-defined mask to a fMRI slice in order to select regions of interest (ROIs) for processing:

```
R> library("cudaBayesreg")
R> slicedata <- read.fmrislice(fbase = "fmri", slice = 3, swap = FALSE)
R> ymaskdata <- premask(slicedata)
R> fsave <- paste(tempdir(), "/simultest1", fileext = ".sav", sep = "")
R> out <- cudaMultireg.slice(slicedata, ymaskdata, R = 2000, fsave = fsave)
```

In this example, we have followed the practice of including the HRF waveforms (regressor coefficients β_2 and β_4) as well as their derivatives (regressor coefficients β_3 and β_5) to account for variability in the shape of the response. The regressor coefficients β_2 and β_4 are associated with the visual and auditory regressors, respectively. The regressor coefficient β_1 represents the intercept term. In the code snippets, the variables `vreg`*i* are used to select the regressor coefficients β_i. We may visualise the posterior probability map (PPM) images for the visual (`vreg = 2`) and auditory (`vreg = 4`) stimulation using the function `post.ppm()` (see Figure 1) as follows: Highest probability density (HPD) 95% intervals of the β_2 and β_4 distributions are used to define the thresholds of voxel activations associated with the visual and auditory cortex areas, respectively.

```
R> post.ppm(out = out, slicedata = slicedata, ymaskdata = ymaskdata,
+    vreg = 2, col = heat.colors(256))
R> post.ppm(out = out, slicedata = slicedata, ymaskdata = ymaskdata,
+    vreg = 4, col = heat.colors(256))
```

The function `cudaMultireg.volume()` processes all slices included in a given fMRI data set sequentially, by calling the function `cudaMultireg.slice()` repeatedly. Alternatively, a user specified range of slices in the data volume may be specified by the user for MCMC

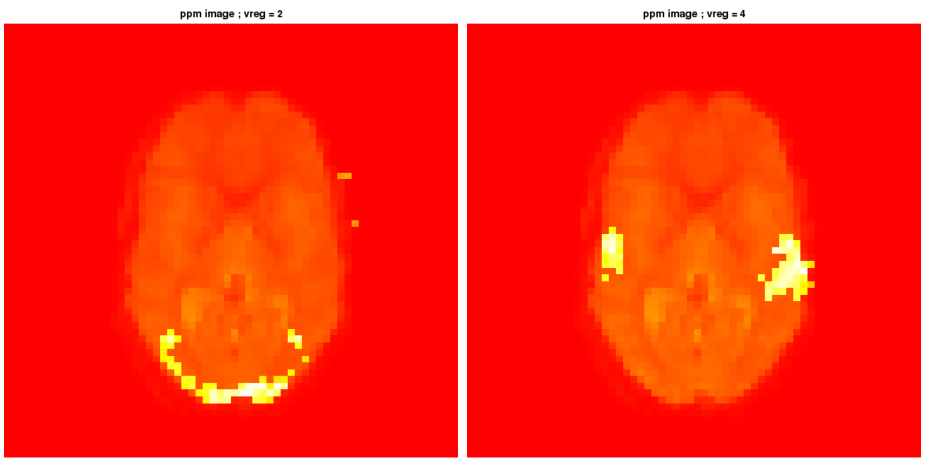

Figure 1: PPM images for slice 3 of the `fmri` data set: (a) areas of visual stimulation; (b) areas of auditory stimulation.

simulation purposes. The following piece of code runs the Bayesian simulation for slices 13 to 24 of the `swrfM` data set, and builds a NIFTI volume of statistical PPMs images via `buildzstat.volume`:

```
R> cudaMultireg.volume(fbase = "swrfM", R = 3000, rg = c(13, 24))
R> buildzstat.volume(fbase = "swrfM", rg = c(13, 24))
```

The statistical PPM volume of voxel activations may now be overlayed on the original fMRI data volume to yield a visualisation of the areas of auditory activation estimated by the Bayesian multilevel method, as depicted in Figure 2:

```
R> post.overlay(fbase = "swrfM", vreg = 2, rg = c(13, 24), view = "axial")
```

The experimental analyses and the PPM images presented in Section 5.1 show that the Bayesian approach proposed in this work achieves good results in terms of true activations and reduced number of artifacts, in comparison with the 'state-of-the-art' approaches incorporated in the **SPM** and **FSL** software packages. We consider that the techniques reflected in these packages are representative of the field, since they are used by the vast majority of researchers in neuroimaging and cognitive neuroscience. A detailed comparison of the quality of the neuroscientific results achieved using **cudaBayesreg** is beyond the scope of the present manuscript. We refer the interested reader to reference (Ferreira da Silva 2011a), where the **SPM** and **FSL** packages were applied to the same data sets considered in this work. As explained before for the `fmri` example, a proper calibration of brain masks and other pre-processing parameters could further improve the quality of **cudaBayesreg** estimates.

5.2. Bayesian posterior analyses

The function `cudaBayesreg.slice()` drives the MCMC simulations to estimate the regression coefficients at each voxel. In Bayesian analysis, PPMs are used for representing the 'activated' cortex areas in response to some experimental stimulation. Similarly to classical

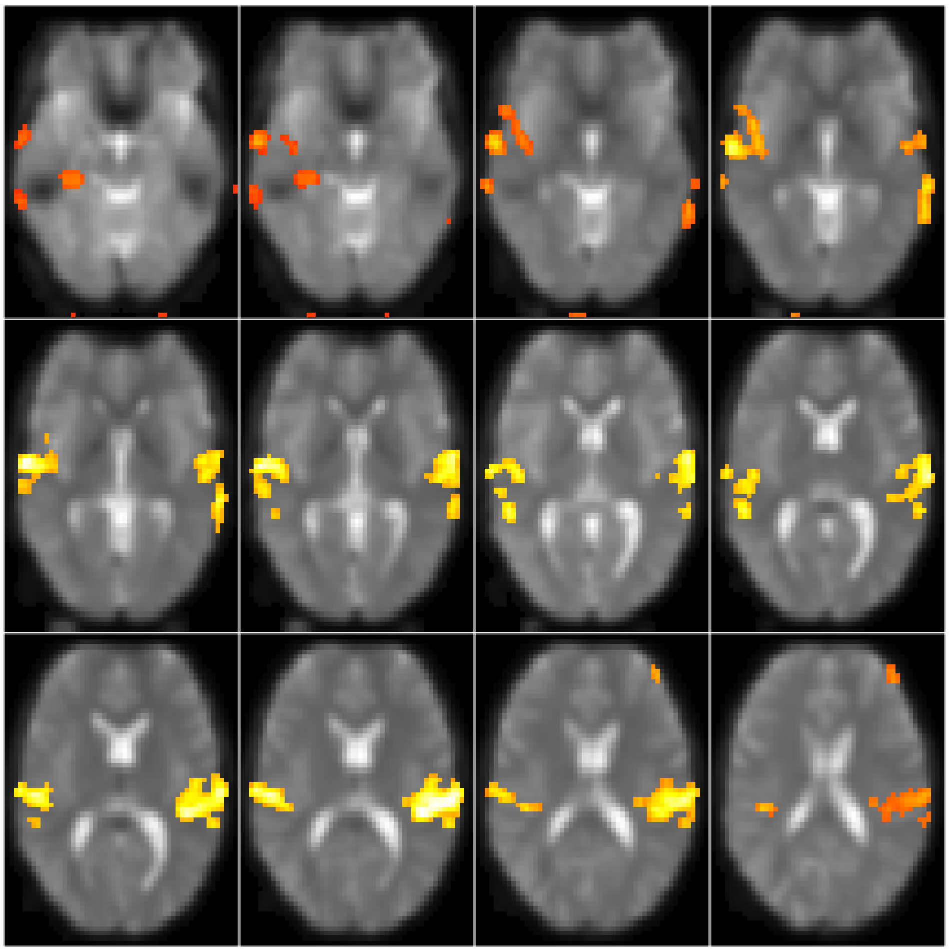

Figure 2: PPMs representing areas of auditory activation, estimated by the multilevel method for axial slices 13–24 of the `swrfM` data set.

.

SPMs, PPMs can be obtained by thresholding the regression coefficients. Bayesian thresholds are selected using the posterior distributions of the regressor coefficients. We may rely on highest probability intervals to select the thresholds, instead of using frequentist quantiles as in SPM-based approaches.

As an example, consider slice 21 of the `swrfM` data set, and define a design matrix with three regressors as commonly specified by standard fMRI packages such as **FSL** or **SPM** (run `?swrfM_design` in R). Regressor β_1 represents the intercept term, regressor β_2 is the main regressor for the auditory time-series, and β_3 is the temporal derivative regressor (Ashburner *et al.* 2008). By default, we use the highest probability density (HPD) 95% interval of the β_2 distribution to define the thresholds of voxel activations associated with the auditory cortex areas. The function `post.simul.hist()` outputs summary statistics for the posterior mean values of the auditory regression coefficient β_2 (`vreg = 2`), and plots the histogram of the posterior distribution, as represented on the left panel of Figure 3:

```
R> load(paste(tempdir(), "/swrfM_s21_nu3.sav", sep = ""))
R> post.simul.hist(out = out, vreg = 2)
```

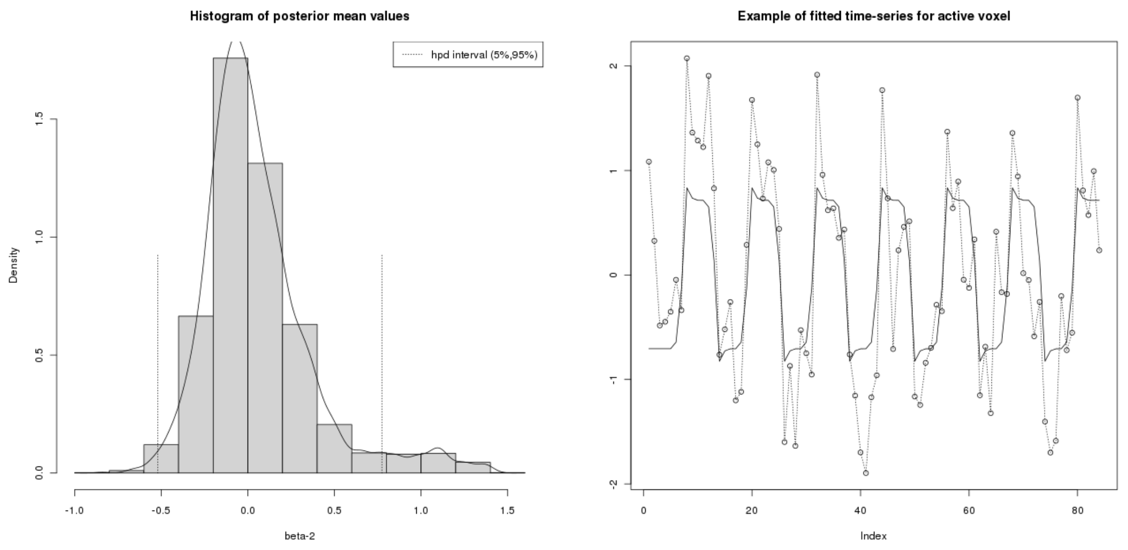

Figure 3: (a) Histogram of the posterior distribution of the regression coefficient β_2 (slice 21); (b) Fitted time-series for a randomly selected "active" voxel in slice 21 of the `swrfM` data set.

```
Call:
        density.default(x = pm2)

Data: pm2 (2872 obs.);        Bandwidth 'bw' = 0.04294

       x                  y
 Min.   :-0.9674   Min.   :3.731e-05
 1st Qu.:-0.3381   1st Qu.:3.974e-02
 Median : 0.2912   Median :9.467e-02
 Mean   : 0.2912   Mean   :3.969e-01
 3rd Qu.: 0.9205   3rd Qu.:5.749e-01
 Max.   : 1.5498   Max.   :1.849e+00
[1] "active range:"
[1] 0.7736182 1.4209423
[1] "non-active range:"
[1] -0.8385900  0.7729416
hpd (95%)=               -0.5199595 0.7736182
```

The vertical dotted lines in the histogram reference the HPD 95% interval values for the β_2 distribution given in the summary statistics. The upper value of the HPD interval is used as a threshold estimate to produce posterior probability maps.

To show the fitted time series for a randomly selected "active" voxel in slice 21, as depicted on the right panel of Figure 3, we use the code:

```
R> slicedata <- read.fmrislice(fbase = "swrfM", slice = 21)
R> ymaskdata <- premask(slicedata)
```

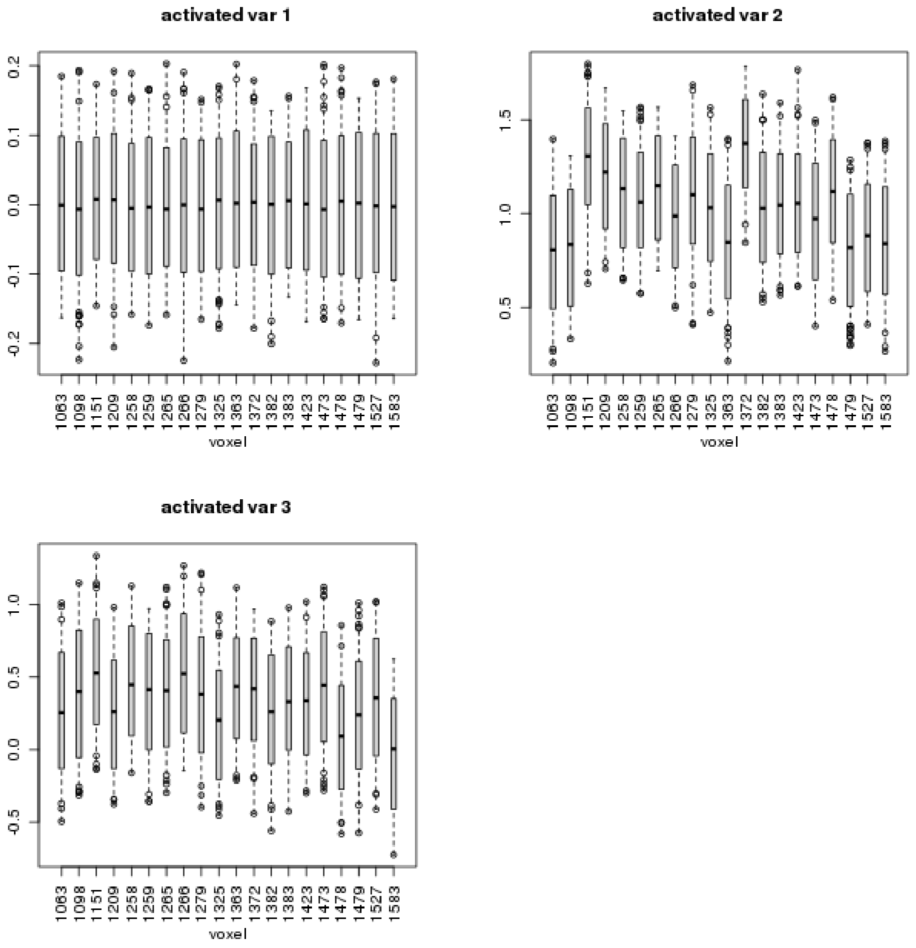

Figure 4: Boxplots of posterior distributions for the regressor coefficients $\{\beta_1, \beta_2, \beta_3\}$, based on 20 randomly selected "active" voxels.

```
R> post.tseries(out = out, slicedata = slicedata, ymaskdata = ymaskdata,
+    vreg = 2)
```

To illustrate the variability of the estimates generated by the parallel RNG procedure outlined in Section 2, we show in Figure 4 boxplots of the posterior distributions of the regression coefficients $\{\beta_1, \beta_2, \beta_3\}$ for voxels in "activated" cortex areas. Figure 4 shows 20 random boxplots of the β distributions, when **cudaBayesreg** is used to fit time series of voxels in estimated "active" visual cortex areas:

```
R> vreg <- 2
R> pmeans <- pmeans.hcoef(out$betadraw)
R> px <- regpostsim(pmeans, vreg = vreg)
R> spma <- px$spma
R> plot(out$betadraw, spmname = "activated", spm = spma, nsamp = 20)
```

The boxes' lower and upper hinges summarise HPD intervals. A similar procedure has been

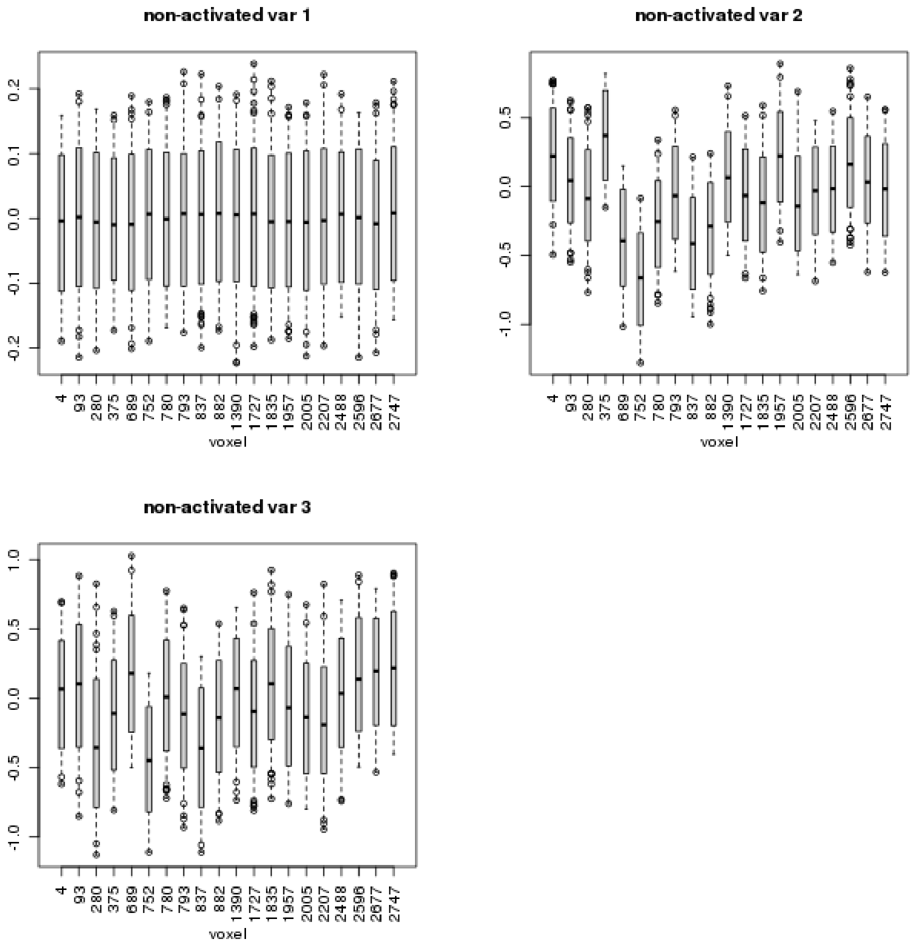

Figure 5: Boxplots of posterior distributions for the regressor coefficients $\{\beta_1, \beta_2, \beta_3\}$, based on 20 randomly selected "non-active" voxels.

used in Figure 5, for time series of voxels estimated as "non-activated":

```
R> spmn <- px$spmn
R> plot(out$betadraw, spmname = "non-activated", spm = spmn, nsamp = 20)
```

5.3. Adaptive shrinkage

An important feature of the Bayesian model used in **cudaBayesreg** and outlined in Section 2, is the shrinkage induced by the hyperprior ν in (10) (Ferreira da Silva 2011a). To illustrate the influence of the hyperparameter ν on the shrinking properties of the multilevel approach, Figure 6 compares the variability of the posterior predictive values of $y_i = X\hat{\beta}_i$, using the voxels of slice 21 of the auditory data set for MCMC simulations with two different values of the hyperparameter ν ($\nu = \{3, 168\}$). The predictive y_i values were obtained using the estimated $\hat{\beta}_i$ values at each voxel. Figure 6, clearly illustrates the phenomenon of "shrinkage" in the Bayesian multilevel approach, and the influence of the hyperparameter ν on the variability of the fitted values. The hyperparameter ν is a shrinkage parameter which works as a regularisation parameter in a data-adaptive way. We may assess the adaptive shrinkage properties of the Bayesian multilevel model for two different values of ν as detailed next:

```
R> slicedata <- read.fmrislice(fbase = "swrfM", slice = 21)
R> ymaskdata <- premask(slicedata)
R> rng <- 1
R> nu1 <- 3
R> f1 <- paste(tempdir(), "/swrfM_s21_", nu1, "_rng", rng, ".sav", sep = "")
R> out1 <- cudaMultireg.slice(slicedata, ymaskdata, R = 3000, nu.e = nu1,
+    fsave = f1, rng = 1)
R> nu2 <- 168
R> f2 <- paste(tempdir(), "/swrfM_s21_", nu2, "_rng", rng, ".sav", sep = "")
R> out2 <- cudaMultireg.slice(slicedata, ymaskdata, R = 3000, nu.e = nu2,
+    fsave = f2, rng = 1)
R> vreg <- 2
R> x1 <- post.shrinkage.mean(out = out1, slicedata$X, vreg = vreg,
+    plot = FALSE)
R> x2 <- post.shrinkage.mean(out = out2, slicedata$X, vreg = vreg,
+    plot = FALSE)
R> par(mfrow = c(1, 2), mar = c(4, 4, 1, 1) + 0.1)
R> xlim = range(c(x1$beta, x2$beta))
R> ylim = range(c(x1$yrecmean, x2$yrecmean))
R> plot(x1$beta, x1$yrecmean, type = "p", pch = "+", col = "violet",
+    ylim = ylim, xlim = xlim, xlab = expression(beta), ylab = "y")
R> legend("topright", expression(paste(nu, "=3")), bg = "seashell")
R> plot(x2$beta, x2$yrecmean, type = "p", pch = "+", col = "blue",
+    ylim = ylim, xlim = xlim, xlab = expression(beta), ylab = "y")
R> legend("topright", expression(paste(nu, "=168")), bg = "seashell")
R> par(mfrow = c(1, 1))
```

The code above also demonstrates how to select Brent's random number generator (RNG) (Brent 2006) to run the simulations, by specifying the argument `rng=1` in `cudaMultireg.slice`. The **cudaBayesreg** package includes three optional CUDA-based RNGs: Marsaglia's multicarry RNG (Marsaglia 2003), Brent's RNG and Matsumoto's Mersenne Twister (Matsumoto and Nishimura 1998). Marsaglia's multicarry RNG follows the R implementation, and is selected by default (with the argument `rng=0`) since it is the fastest one.

5.4. Analysis of random effects for the SPM auditory data set

The Bayesian multilevel statistical model allows for the analysis of random effects through the specification of the Z matrix for the prior in (8). We exemplify the analysis of the random effects distribution Δ (see (8)), following the specification of cross-sectional units (group information) in the Z matrix of the statistical model. The **FSL** tools (Smith *et al.* 2004) were used to obtain the segmented masks associated with the partition of the **SPM** auditory data set in three classes: cerebrospinal fluid (CSF), grey matter (GM), and white matter (WM). The segmented masks were obtained by applying FSL/FAST to the structural high-resolution fMRI image, followed by FSL/FLIRT for low-resolution registration

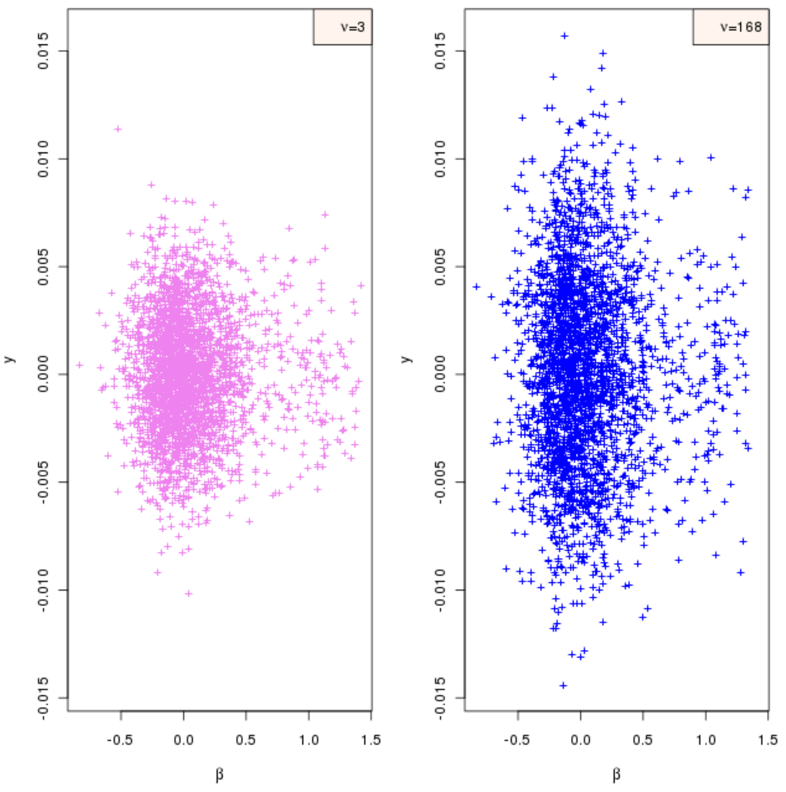

Figure 6: Variability of the posterior predictive y values as a function of the estimated $\hat{\beta}$ values in voxels of slice 21 of the auditory data set, for MCMC simulations with two different values of the hyperparameter ν: (a) left panel: $\nu = 3$; (b) right panel: $\nu = 168$.

(see `cudaBayesreg::read.Zsegslice`). The segmented images (CSF/GM/WM) were then used to build the Z matrix in (8). In addition, to account for variability in the shape of the voxels' time series response, we included the derivative of the HRF as a third regressor in the specification of the design matrix (see `cudaBayesreg::swrfM_design`). Thus, the MCMC simulation uses three regression variables, in which the first represents the intercept and the second is the main regressor.

As before, we begin by loading the data and running the simulation. This time, however, we call `cudaMultireg.slice` with the argument `zprior = TRUE`. This argument will launch `read.Zsegslice`, that reads the segmented images (CSF/GM/WM) to build the Z matrix:

```
R> fbase <- "swrfM"
R> slice <- 21
R> slicedata <- read.fmrislice(fbase = fbase, slice = slice)
R> ymaskdata <- premask(slicedata)
R> f3 <- paste(tempdir(), "/swrfM_s21_zprior.sav", sep = "")
R> out <- cudaMultireg.slice(slicedata, ymaskdata, R = 3000, keep = 5,
+    nu.e = 3, fsave = f3, zprior = TRUE, rng = 1)
```

Plots of the draws of the mean of the random effects distribution for each one of the three regression variables used in the design matrix X are presented in Figure 7(a), as generated

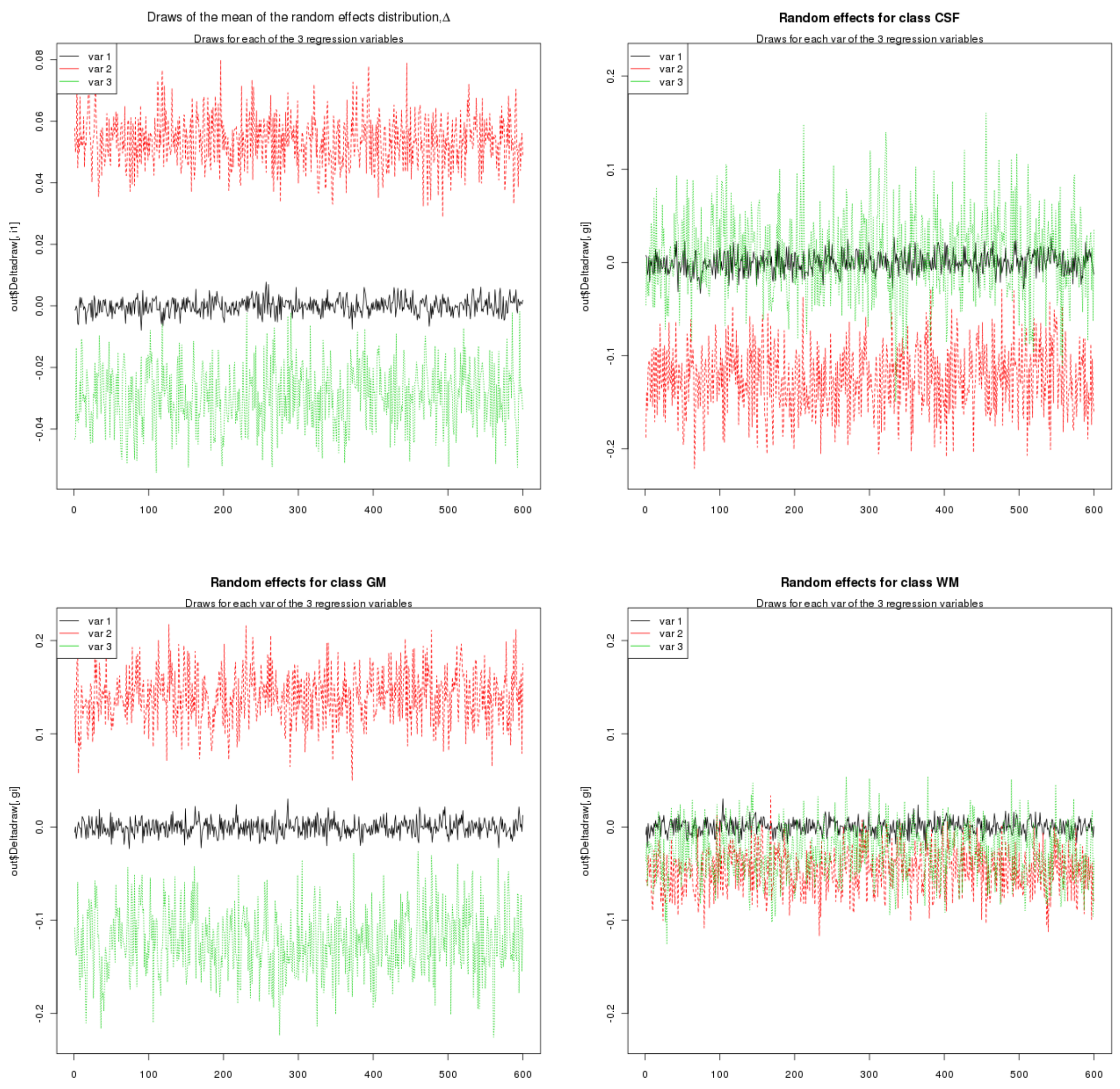

Figure 7: (a) Draws of the mean of the random effects distribution; (b) Draws of the random effects distribution associated with class CSF:(c) idem for class GM; (d) idem for class WM.

by `post.randeff`:

```
R> post.randeff(out)
```

The same function may be used to represent the draws of the random effects distribution associated with each one of the three segmentation classes (CSF/GM/WM), as shown in Figure 7(b–d):

```
R> post.randeff(out, classnames = c("CSF", "GM", "WM"), climits = TRUE)
```

These plots clarify the influence of different tissue types on the mean values of the regression estimates. In general, the information provided by these analyses might be useful in identifying factors influencing estimation, and suggest ways for model improvement.

5.5. Convergence issues

To assess convergence of the MCMC simulation, it is in general advisable to visualise the evolution of the simulated values and autocorrelation functions (ACFs) generated by the multilevel simulation. As an example, we show in Figure 8 the evolution of the simulated values and autocorrelation functions (ACFs) generated by the multilevel simulation of the Δ values in (8–10) for slice 21 of the `swrfM` data set, using a sub-sampling approach which keeps every 5 iteration of the MCMC chain:

```
R> library("cudaBayesreg")
R> fbase <- "swrfM"
R> slice <- 21
R> slicedata <- read.fmrislice(fbase = fbase, slice = slice, swap = FALSE)
R> ymaskdata <- premask(slicedata)
R> fsave <- paste(tempdir(), "/simultest20.sav", sep = "")
R> out <- cudaMultireg.slice(slicedata, ymaskdata, R = 5000, keep = 5,
+    nu.e = 3, fsave = fsave, zprior = FALSE, rng = 1)
R> plot(out$Deltadraw)
```

To plot the sequence plots of MCMC draws and ACFs in Figure 8 we used the **cudaBayesreg** $S3$ method `plot.bayesm.mat`. For compatibility with the equivalent summary and plot functions in **bayesm**, the output of the MCMC simulations in **cudaBayesreg** adopts the class attributes used in **bayesm**, e.g., `attributes(Deltadraw)$class = c("bayesm.mat", "mcmc")` (see `cudaMultireg.slice`). This means that equivalent plots (and summary statistics) may be obtained using **bayesm**, e.g.:

```
R> library("bayesm")
R> plot(out$Deltadraw)
```

A more comprehensive approach to study convergence issues is to rely on the convergence tests included in **boa** (Smith 2007), by submitting the output of MCMC simulation to **boa** functions as follows:

```
R> resdata <- out$Deltadraw
R> z <- matrix(resdata, dim(resdata))
R> library("boa")
R> boa.quit()
R> boa.init()
R> boa.chain.add(z, "Deltadraw")
R> boa.print.acf()
R> boa.print.randl()
R> boa.plot("acf")
R> boa.plot("trace")
R> boa.quit()
```

Figure 8 and **boa** analyses show good mixing properties, and no significant autocorrelations for simulations with 5000 iterations.

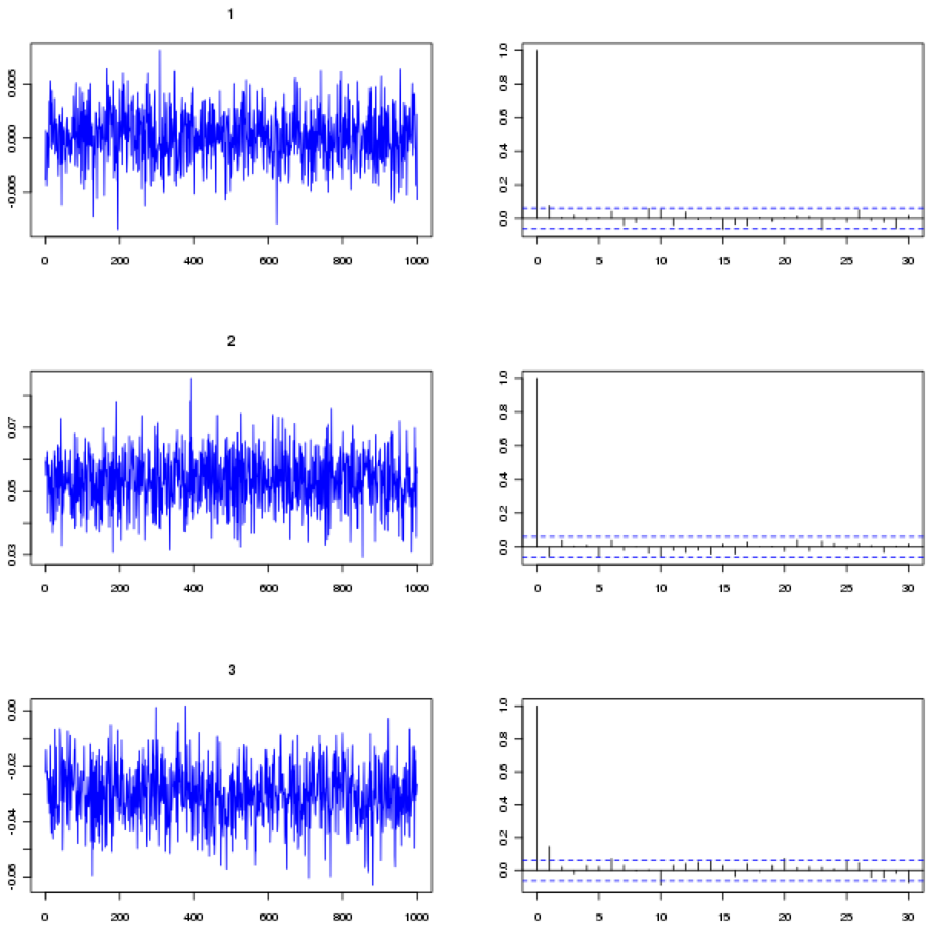

Figure 8: Evolution of the simulated values and autocorrelation functions generated by the cudaBayesreg simulation of the Δ values (8–10) to draw the voxel regression coefficients $\{\beta_i\},\ i = 1, 2, 3$, using a sub-sampling approach with keep factor 5.

6. Conclusion

The GPU is rapidly gaining maturity as a powerful general parallel computing device. The **cudaBayesreg** package shows how the combination of R and GPU programming tools can be used to improve the performance of Bayesian fMRI data analyses. The implemented code may easily be modified to process all voxels of a fMRI volume in parallel, instead of processing data slice-by-slice. Moreover, a desirable extension of the Bayesian multilevel techniques implemented in **cudaBayesreg** is the support for multi-subject and multi-session fMRI data analyses. In these cases, a more powerful GPU that the one with just 2 multiprocessors and 16 CUDA cores used in this work is highly recommended. GPUs with 16 multiprocessors and 512 CUDA cores are now available at affordable prices. The use of more sophisticated GPUs would also enable the programmer to exploit additional programming constructs for performance improvement. However, incorporating these constructs in **cudaBayesreg** would entail additional hardware requirements to be met by the would-be user.

Acknowledgments

The author thanks the anonymous reviewers and the associated editors for their constructive comments and valuable recommendations.

References

Ashburner J, Chen C, Flandin G, Henson R, Kiebel S, Kilner J, Litvak V, Moran R, Penny W, Stephan K, Hutton C, Glauche V, Mattout J, Phillips C (2008). *The SPM8 Manual.* Functional Imaging Laboratory, Wellcome Trust Centre for Neuroimaging, Institute of Neurology, UCL, London. URL `http://www.fil.ion.ucl.ac.uk/spm/`.

Box GEP, Tiao GC (1973). *Bayesian Inference in Statistical Analysis.* John Wiley & Sons.

Brent RP (2006). "Some Long-Period Random Number Generators Using Shifts and Xors." 13th Biennial Computational Techniques and Applications Conference (CTAC06), Townsville. URL `http://gan.anu.edu.au/~brent/pd/rpb224.pdf`.

Fatahalian K, Houston M (2008). "A Closer Look at GPUs." *Communications of the ACM*, **51**(10), 50–57.

Ferreira da Silva AR (2010). "**cudaBayesreg**: Bayesian Computation in CUDA." *The R Journal*, **2**(2), 48–55. URL `http://journal.R-project.org/archive/2010-2/RJournal_2010-2_Ferreira~da~Silva.pdf`.

Ferreira da Silva AR (2011a). "A Bayesian Multilevel Model for fMRI Data Analysis." *Computer Methods and Programs in Biomedicine*, **102**, 238–252.

Ferreira da Silva AR (2011b). ***cudaBayesreg**: CUDA Parallel Implementation of a Bayesian Multilevel Model for fMRI Data Analysis.* R package version 0.3-10, URL `http://CRAN.R-project.org/package=cudaBayesreg`.

Ferreira da Silva AR (2011c). ***cudaBayesregData**: Data Sets for the Examples Used in the Package **cudaBayesreg**.* R package version 0.3-10, URL `http://CRAN.R-project.org/package=cudaBayesregData`.

Gelman A (2006). "Multilevel (Hierarchical) Modeling: What It Can and Cannot Do." *Technometrics*, **48**(3), 432–435.

Geweke J (2005). *Contemporary Bayesian Econometrics and Statistics.* John Wiley & Sons.

Judge GG, Hill RC, Griffiths WE, Lütkephol H, Lee TC (1988). *Introduction to the Theory and Practice of Econometrics.* 2nd edition. John Wiley & Sons.

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

Lindley DV, Smith AFM (1972). "Bayes Estimates for the Linear Model." *Journal of the Royal Statistical Society B*, **34**(1), 1–41.

Mardia KV, Kent JT, Bibby JM (1979). *Multivariate Analysis.* Academic Press.

Marsaglia G (2003). "Random Number Generators." *Journal of Modern Applied Statistical Methods*, **2**(1), 2–13.

Matsumoto M, Nishimura T (1998). "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudorandom Number Generator." *ACM Transactions on Modeling and Computer Simulation*, **8**, 3–30.

Nickolls J, Buck I, Garland M, Skadron K (2008). "Scalable Parallel Programming." *ACM Queue*, **6**(2), 40–53.

Nvidia Corporation (2010a). *CUDA C Best Practices Guide Version 3.2*. URL http://www.nvidia.com/CUDA.

Nvidia Corporation (2010b). *Nvidia CUDA Programming Guide, Version 3.2*. URL http://www.nvidia.com/CUDA.

Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC (2008). "GPU Computing." *Proceedings of the IEEE*, **96**(5), 879–899.

Owens JD, Luebke D, Govindaraju N, Harris M, Krueger J, Lefohn AE, Purcell TJ (2007). "A Survey of General-Purpose Computation on Graphics Hardware." *Computer Graphics Forum*, **26**(1), 80–113.

Press J (2003). *Subjective and Objective Bayesian Statistics*. 2nd edition. John Wiley & Sons.

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007). *Numerical Recipes, The Art of Scientific Computing*. 3rd edition. Cambridge University Press.

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

Rossi P (2011). ***bayesm**: Bayesian Inference for Marketing/Micro-Econometrics*. R package version 2.2-4, URL http://CRAN.R-project.org/package=bayesm.

Rossi PE, Allenby GM, McCulloch R (2005). *Bayesian Statistics and Marketing*. John Wiley & Sons.

Smith BJ (2007). "**boa**: An R Package for MCMC Output Convergence Assessment and Posterior Inference." *Journal of Statistical Software*, **21**(11), 1–37. URL http://www.jstatsoft.org/v21/i11/.

Smith SM, Jenkinson M, Woolrich MW, Beckmann CF, Behrens TEJ, Johansen-Berg H, Bannister PR, Luca MD, Drobnjak I, Flitney DE, Niazy RK, Saunders J, Vickers J, Zhang Y, Stefano ND, Brad JM, Matthews PM (2004). "Advances in Functional and Structural MR Image Analysis and Implementation as FSL." *Technical Report TR04SS2*, FMRIB (Oxford Centre for Functional Magnetic Resonance Imaging of the Brain). URL http://www.fmrib.ox.ac.uk/fsl.

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

Whitcher B, Schmid V, Thornton A (2011a). ***oro.nifti**: Rigorous – NIfTI+ANALYZE+AFNI Input / Output.* R Package Version 0.2.6, URL http://CRAN.R-project.org/package=oro.nifti.

Whitcher B, Schmid VJ, Thornton A (2011b). "Working with the DICOM and NIfTI Data Standards in R." *Journal of Statistical Software*, **44**(6), 1–28. URL http://www.jstatsoft.org/v44/i06/.

Zellner A (1962). "An Efficient Method of Estimating Seemingly Unrelated Regressions and Tests for Aggregation Bias." *Journal of the American Statistical Association*, **57**, 348–368.

Affiliation:

Adelino R. Ferreira da Silva
Departamento de Engenharia Electrotécnica
Faculdade de Ciências e Tecnologia (FCT)
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
E-mail: afs@fct.unl.pt

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 44, Issue 4 *Submitted:* 2010-10-20
October 2011 *Accepted:* 2011-06-16