

ESS++: Evolutionary Stochastic Search for Bayesian model exploration

DOCUMENTATION, V0.1

October 2010

Contents

1	Overview	2
2	Installation of <i>ESS++</i>	3
3	Contents of the <i>ESS++</i> package	4
3.1	Main directory	4
3.2	Classes directory	5
3.3	Routines directory	5
3.4	Example directory	5
3.5	Documentation directory	5
4	Prior specification	6
5	EMC implementation	7
5.1	Local moves	7
5.2	Global moves	9
5.2.1	Crossover moves	9
5.2.2	Exchange moves	11
5.3	Temperature placement	12
5.4	Sampling the selection coefficient	13
6	Overview of the algorithm	14

7	Running <i>ESS++</i>	17
7.1	Input Files	17
7.2	Output files	19
7.3	Further output files	21
7.4	Command line options	22
8	Example	24

Introduction

1 Overview

ESS++ is a C++ implementation of a fully Bayesian variable selection approach that can analyse single and multiple responses in an integrated way. In the following, for ease of notation and exposition, we only present the multiple responses case.

Let X denote the $n \times p$ predictor matrix, where n is the number of observations and p is the number of predictors and Y the $n \times q$ outcome matrix. *ESS++* searches for subsets of X that reliably explain the joint variation of Y . This problem, known as variable selection or subset selection, is particularly interesting when p is large and parsimonious models containing only a few predictors are sought to gain interpretability. *ESS++* works well for problems where $n \geq p$ and also $p < n$, including the case where $n \ll p$ (known as the “large p , small n ” case). In the current version, *ESS++* can handle several hundreds of observations ($n < 1,000$), thousands of predictors ($p < 10,000$) and a few responses simultaneously ($q < 10$). Since *ESS++* searches for subsets of covariates that jointly predict *all* the responses, the type and size q of the responses should be chosen carefully.

The model used by *ESS++* is the multivariate extension of the Gaussian linear model

$$Y - XB \sim \mathcal{N}(K, \Sigma) \tag{1}$$

with $\mathcal{N}(\cdot, \cdot)$ indicating the normal matrix variate and where:

- B is a $p \times q$ matrix of regression coefficients;
- K is a $n \times n$ matrix that controls the correlation among the observations;
- Σ is a $q \times q$ matrix that controls the correlation structure among the responses (the variance-covariance matrix of Y).

Given the latent binary vector $\gamma = (\gamma_1, \dots, \gamma_p)^T$, with $\gamma_j = 1$ if $\beta_j \neq 0$ and $\gamma_j = 0$ if $\beta_j = 0$, $j = 1, \dots, p$, the Gaussian linear model (1) becomes

$$Y - X_\gamma B_\gamma \sim \mathcal{N}(K, \Sigma), \quad (2)$$

where B_γ is the $p_\gamma \times q$ matrix of non-zero regression coefficients extracted from B , $p_\gamma \equiv 1_p^T \gamma$, and X_γ is the design matrix of dimension $n \times p_\gamma$ with columns corresponding to $\gamma_j = 1$.

ESS++ explores the 2^p -dimensional model space using an extension of Parallel Tempering called Evolutionary Monte Carlo that combines Monte Carlo Markov Chain (MCMC) and genetic algorithms. Specifically, *ESS++* relies on tempered multiple chains run in parallel that exchange information about the configuration of the latent binary vector γ through local and global moves.

In this documentation we first detail the prior settings of the model as well as the types of moves implemented in *ESS++*. We will then describe *ESS++* usage: input, parameters, options and outputs will be listed and commented. For further details about *ESS++* see:

Bottolo, L. and Richardson S. (2010). Evolutionary Stochastic Search for Bayesian model exploration. *Bayesian Analysis* **5**(3), 583–618.

Petretto, E., Bottolo, L., Langley, S.R., Heinig., M., McDermott-Roe, C., Sarwar, R., Pravenec, M., Hübner, N., Aitman, T.J., Cook, S.A. and Richardson, S. (2010). New insights into the genetic control of gene expression using a Bayesian multi-tissue approach. *PLoS Comput. Biol.*, **6**(4), e1000737.

ESS++ is free to use, distribute and modify, under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version. In particular *ESS++* comes WITHOUT ANY WARRANTY.

Please report any problem or bug to m.chadeau@imperial.ac.uk or d.hastie@imperial.ac.uk .

Please cite this article in any publication that uses *ESS++*.

2 Installation of *ESS++*

1. *ESS++* is a C++ program that uses packages from the GNU Scientific Library (GSL), available at:

`ftp://ftp.gnu.org/gnu/gsl/`

For Linux users, the installation is straightforward and documented in the downloaded archive.

For Mac users we suggest the following:

- Download and install MacPorts from

`http://www.macports.org/`

- Then type:

```
sudo port install gsl
```

- Change the library path in the command line (or in profile file, in the home directory, to get the correct path exported at each boot):

```
export PATH=$PATH:/opt/local/bin:~/local/bin
export LIBRARY_PATH=$LIBRARY_PATH:/opt/local/lib
export CPATH=$CPATH:/opt/local/include
```

For Windows users, we recommend emulating Linux using, for instance, Cygwin and then following the Linux instructions above.

ESS++ has been developed and tested using the `g++` compiler on several Linux and Mac platforms and Cygwin. Compilation using alternative compilers will require some modifications in the `makefile`.

2. Download `ESS_Release_0.1.tgz` and unpack it by typing `tar zxvf ESS_Release_0.1.tgz` to generate the `ESS` directory.
3. Type `cd ESS/Main` and then `make` to create the executable file `ESS`.
4. To check the installation, and create the example files mentioned below, move to the `ESS/Example` directory and run the example script by typing `./ESS_example.sh`

3 Contents of the *ESS++* package

3.1 Main directory

This directory contains the *ESS++* source code. The main program is `ESS.cc`. In this file (and many of the other `*.cc` files), a variable called `DEBUG` is defined and set to 0 by default. If the user changes

it to 1 and re-compiles the program, then step-by-step details of the program will be printed out on the standard output device (or `log` file). This option is useful to understand how *ESS++* works, but it usually generates very large `log` files.

3.2 Classes directory

This directory contains all the C++ classes used in *ESS++*. The classes typically relate to one feature of the model and contain the associated parameters. There is also a class (`Move_monitor`) to collate statistics in order to monitor the history of the MCMC run.

3.3 Routines directory

This directory contains:

- routines to read/write `xml` files (taken from the FREGENE program, see source files for details);
- GSL-based routines to enable random number generation, matrix handling and linear algebra.
- all functions called in *ESS++* main code.

3.4 Example directory

This directory illustrates how to use *ESS++*. It provides one example of each input file required by *ESS++* as well as a shell file to run *ESS++*. Having run this example the directory will also contain examples of the output files produced by *ESS++*.

3.5 Documentation directory

This directory contains a pdf copy of this document, along with a copy of the software license.

Features of the *ESS++* implementation

The current version of *ESS++* works with observations that do not exhibit any structure (e.g. familial data) and therefore it considers only independent observations: setting $K = I_n$, equation (2) becomes

$$Y - X_\gamma B_\gamma \sim \mathcal{N}(I_n, \Sigma).$$

4 Prior specification

Uncertainty is introduced by specifying suitable prior distributions for all unknowns. The prior density for the $p_\gamma \times q$ matrix of non-zero regression coefficients B_γ is

$$B_\gamma \sim \mathcal{N}(H_\gamma, \Sigma),$$

where the correlation structure between the q columns of B_γ is set equal to the variance-covariance matrix of Y for computational reasons. The prior density of Σ is

$$\Sigma \sim \mathcal{IW}(\delta, Q),$$

where $\mathcal{IW}(\cdot, \cdot)$ denotes the inverse Wishart distribution with $\delta = 3$ and $Q = kI_q$. The hyperparameter k is automatically elicited by *ESS++* and both scalar parameters are stored in the `Prior_param` class (see `Classes/Prior_param.cc`).

The matrix H_γ controls the correlation structure of the regression coefficients among the p_γ predictors. In *ESS++* we opt for the g -prior setting

$$H_\gamma = g (X_\gamma^T X_\gamma)^{-1},$$

where:

- g , the variable selection coefficient, is a scalar;
- g is either fixed or unknown. In the latter case, a prior density is specified, $g \sim \Gamma(1/2, n/2)$, and included in the MCMC sampling scheme, where $\Gamma(\cdot, \cdot)$ indicates the Gamma density.

The main assumption underlying g -priors is that the correlation structure of the regression coefficients among the p predictors replicates the covariance structure of the likelihood. The benefit is twofold: first, the calculation of the marginal likelihood is simplified; secondly, it enables the variance of each regression coefficient to adapt automatically to the scale of the covariates, which is of primary importance when the X matrix incorporates heterogeneous predictors.

H_γ could alternatively be modelled using an independent prior

$$H_\gamma = gI_p$$

assuming that there is no *a priori* correlation between the regression coefficients, with g either fixed or unknown.

All hyper-parameters are stored in the `Prior_param` class (see `Classes/Prior_param.cc`). Most of its members are automatically calculated from the two main user-defined parameters (see Section 7):

- `E_p_gam`, the *a priori* expected model size;
- `SD_p_gam`, the *a priori* standard deviation of the model size.

5 EMC implementation

The principle of *ESS++* is to explore the model space by iteratively sampling candidates for the binary latent vector γ . In the Gibbs move, candidates are sampled from their full conditional distribution, while in the Metropolis-Hastings move, candidates are sampled from their proposal distribution and then are accepted or rejected with a probability mechanism.

In order to reliably explore the huge model space, we run several chains in parallel, each of which is powered by a different inverse temperature (scalar) that is specified by the user, with the aim of achieving a trade-off between the speed and mixing of the MCMC algorithm. Higher temperatures flatten the posterior density, allowing the algorithm to escape from local modes and favouring changes in the chain specific γ configuration. The first chain is not heated, i.e. its temperature is set equal to 1. Heated chains are only included to improve mixing, providing good proposals for the first chain's γ configuration when its state is tentatively swapped with the latent binary vectors of the heated chains. For this reason only the information from the non-heated chain is recorded.

We describe below how the moves and the tempering scheme are implemented within *ESS++*. However, for full details about the underlying probabilistic framework, please refer to:

Bottolo, L. and Richardson S. (2010). Evolutionary Stochastic Search for Bayesian model exploration. *Bayesian Analysis* **5**(3), 583–618.

Making the most of the class-oriented structure of the C++ language, several of the moves and the temperature scheme (see below) are defined by specific classes. These classes contains all the parameters required to perform the move (or the temperature adjustment). A move monitoring class (see `Classes/Move_monitor.cc`) keeps a detailed record of the sequence of moves called during the MCMC run and the results produced by the move (or the temperature adjustment).

5.1 Local moves

We define a local move to be any MCMC move that is restricted to a single chain. Suppose that our local move occurs on a specific chain l , where $l \in \{1, 2, \dots, L\}$ and L is the number of chains run in parallel. For a given element j of γ_l , a local move consists in swapping the j th value of γ_l , from 1 to 0 or from 0 to 1, i.e. removing the variable j from the model if it was in at the previous step ($1 \rightarrow 0$), or adding it in if it was not ($0 \rightarrow 1$).

Two types of local moves are implemented in *ESS++*:

- *Gibbs move*

The Gibbs move is implemented in the function `Gibbs_move` (see file `Routines/moves.cc`): it samples, for each j in a random order, a new value for $\gamma_{j,l}$ from its full conditional distribution. An exhaustive scan of this nature is time-consuming (one Gibbs move corresponds to the evaluation of p models), and to control the computational time, the user can define the number of sweeps between two Gibbs moves (`GIBBS_N_BATCH`). For those sweeps where a Gibbs move is selected, it is only performed on the first non-heated chain. As the Gibbs move requires only one user-defined parameter, there is no specific associated C++ class. However, the number of transitions $0 \rightarrow 1$ and $1 \rightarrow 0$ at each Gibbs move call is recorded in the `Move_monitor` class (member `Gibbs_move_history`).

- *Fast-Scan Metropolis-Hastings (FSMH)*

FSMH is a Metropolis-Hastings type-move and it is implemented in the function `FSMH_move` (see file `Routines/moves.cc`). It proposes a swapping move ($1 \rightarrow 0$ or $0 \rightarrow 1$) for a randomly selected subset of elements of γ_l . The probability that a given element of γ_l is selected depends on:

- the current value of the selected element;
- the temperature of the chain;
- the current model size;
- ω , an hyper-parameter automatically calculated (and stored in the `Prior_param` class) which depends on `E_p_gam`, the user-defined *a priori* expected model size, `SD_p_gam`, the user-defined *a priori* standard deviation of the model size, and p , the number of predictors in X .

When the FSMH move is called, it is applied to all chains. As for the Gibbs move, no C++ classes are needed for the FSMH, and the following frequencies are updated and stored at each call of the FSMH move in the `Move_monitor` class (members `FSMH_*`):

- the total number of models evaluated and the number of accepted moves;
- the number of $0 \rightarrow 1$ moves proposed and the number of accepted moves;
- the number of $1 \rightarrow 0$ moves proposed and the number of accepted moves.

At each sweep of the sampler the FSMH move is performed with probability `Prob_mut`, which is user-defined (`P_MUTATION`) and stored in the `Prior_param` class (see Section 7).

If a local FSMH move is not selected for a particular sweep, then a global crossover move is attempted instead.

5.2 Global moves

Global moves are aimed at swapping part or all of the information contained in the latent binary vectors between the selected chains, allowing the algorithm to escape from local modes. In the following we detail the two classes of global moves that we have implemented in *ESS++*: crossover moves and exchange moves.

5.2.1 Crossover moves

Crossover moves are implemented in function `Crossover_move` (see file `Routines/moves.cc`). A crossover move relies on the three following steps:

1. *Selection of the chains*

The two chains are selected according to normalized “Boltzmann weights” (calculated in the function `computeAndSampleLogCondPostBoltz`, in the file `Routines/moves.cc`). These weights depend on the (conditional) posterior density and temperature of each chain. Among the population of chains, a group of chains with high weights are selected by applying a user-defined threshold `P_SEL` (see Section 7) to the cumulative distribution of the ordered Boltzmann weights. The sampling distribution is created by overweighting this group of chains and renormalizing all probabilities. Two chains are then drawn at random according to this distribution. This strategy is designed such that the two selected chains will give rise to a new configuration of the latent binary vectors with higher posterior density.

2. *Crossover breakpoints*

Two different ways of sampling the number and the position of the crossover breakpoints are implemented in *ESS++*. At each sweep, the algorithm selects one of the `k_max + 1` move types at random with equal probability:

- a) *k-points crossover*

In this case the number of breakpoints is uniformly sampled from 1 to `k_max`, the user-defined maximum number of breakpoints. The breakpoint location(s) is(are) sampled from a uniform distribution, and the chains’ latent binary vectors are swapped by shuffling the regions between two breakpoints (see function `recombine_chains` in `Routines/moves.cc`).

b) *Block crossover*

The block crossover swaps blocks of highly correlated variables between the two sampled chains. Firstly, a “reference” variable j is uniformly sampled. Then, all pairwise Pearson correlation coefficients $\rho(X_j, X_{j'})$ $j' = 1, \dots, p, j \neq j'$, are calculated (see function `define_haplotype_bkpts` in `Routines/moves.cc`). We retain for the block crossover all variables whose pairwise correlation coefficient $|\rho(X_j, X_{j'})| < \rho_0$, where the threshold ρ_0 is user-defined and stored in the prior setting class `Prior_param`, member `Prob_crsv_r`. Finally each of these selected elements of the two chains are swapped from one chain to another (see function `recombine_haplotype` in `Routines/moves.cc`).

3. *Acceptance probability*

Regardless of the way chains are swapped, the (conditional) posterior probability is calculated for each chain and the proposed move is accepted with a probability depending on:

- the difference in the (conditional) posterior probability induced by the move from the original to the swapped chains;
- the difference in the Boltzmann’s weight induced by the move from the original to the swapped chains;
- the temperature of the swapped chains.

Parameters of the crossover move are recorded in a C++ class (see `Classes/CM.cc` file) storing the maximum number of breakpoints (member `n_max_breakpoint`), the number of possible crossover moves (member `n_possible_CM_moves`) and the vector of cumulative selection probabilities for the `k_max + 1` move types (member `unit_move_pbt_y_cum`).

Crossover moves are monitored in the `Move_monitor` class. Each time a crossover move is selected, the following summary statistics are updated and recorded (see `Classes/Move_monitor.cc` file, member `CM_history`):

- the type of move: number from 1 to `k_max` code for the k -point crossover, and `k_max + 1` codes for the block crossover move;
- the number of sampled breakpoints;
- the two selected chains;
- acceptance frequencies for the crossover moves over the whole MCMC run and for each sweep.

5.2.2 Exchange moves

Exchange moves can be viewed as an extreme case of the crossover move as the two selected chains swap their whole latent binary vectors.

In *ESS++* two types of exchange moves are implemented:

1. *All Exchange move*

The All Exchange move is a Gibbs-type move, thus each pair of chains is selected according to a renormalised probability which depends on the (conditional) posterior probability of each chain. To guarantee the reversibility of the move, the case where no chains are swapped is also considered. (See function `All_exchange_move` in `Routines/moves.cc`).

2. *Delayed Rejection move*

The Delayed Rejection move is implemented in the function `DR_move` (file `Routines/moves.cc`). A first exchange move is proposed by selecting at random with equal probability two chains. If the first proposed move is rejected, a second move involving one of the two original chains and another chain which is adjacent in the temperature ladder is proposed. That second proposed move is then accepted or rejected according to the acceptance probability defined in the function `DR_move`.

A C++ class for the Delayed Rejection move is implemented (see `Classes/DR.cc`) and records:

- the number of times the move is called (member `nb_calls`);
- the number of times the Delayed Rejection move involves adjacent chains (member `nb_calls_adj`);
- the number of proposed and accepted moves for each chain combination (members `mat_moves_proposed` and `mat_moves_accepted`, respectively).

The optimal acceptance rate of the Delayed Rejection move is controlled during the burn-in period through the temperature placement procedure (see Section 5.3) and accordingly the above values are re-initialised every time a temperature placement takes place.

At each sweep, one of the above exchange moves is called: during the burn-in, only the Delayed Rejection is enabled, and once the burn-in is completed, the Delayed Rejection move is selected with a probability `P_DR`, which is user-defined and stored in the `Prior_param` class (member `Prob_DR`). Whatever exchange move is used, the `Move_monitor` class records at each call the two chains involved in the move.

5.3 Temperature placement

The temperature tuning during the burn in of the MCMC run is a key feature in *ESS++*: it aims at easing the convergence of the algorithm and improving chain mixing. The temperature placement is associated to a C++ class recording its parameter (see `Classes/Temperatures.cc`).

The temperature ladder implemented in *ESS++* is geometric. For a given chain $l \in \{1, \dots, L\}$, the l^{th} temperature is defined as

$$t_l = b^{a_l}, \quad (3)$$

where b is a scalar common to all chains, and a_l is the l^{th} elements of a vector, with $a_l = (l - 1)/a$, for some scalar a . Initial values for both scalars a and b can be user-defined and are stored in the `Temperatures` class (members `a_t_den` and `b_t`, respectively). The resulting temperatures are also stored in the member `t`.

We opted for an automatic placement of temperature ladder such that during the burn-in, the value of `b_t` is tuned in order to control the acceptance rate of the Delayed Rejection moves and keep it close to its optimal value (`TEMP_OPTIMAL`, stored in the `Temperatures` class, member `optimal`). The initial value of `b_t` is user-defined (`B_T`) and the l^{th} element of the vector `a_t`, corresponding to chain l , is initialised to $(l - 1)/a_t_den$, where `a_t_den` is chosen among three possible values entered by the user:

- `A_T_DEN_INF_5K`, the value of `a_t_den` if there are less than 5,000 predictors in X ;
- `A_T_DEN_5_10K`, the value of `a_t_den` if there are between 5,000 and 10,000 predictors in X ;
- `A_T_DEN_SUP_10K`, the value of `a_t_den` if there are more than 10,000 predictors in X .

Every time `TEMP_N_BATCH` Delayed Rejection moves have been performed, (where `TEMP_N_BATCH` is user-defined - see Section 7 - and stored in the `Temperatures` class, member `nbatch`) the temperature placement occurs and the parameter `b_t` is updated as follows (see function `temp_placement` in `Routines/moves.cc` for details):

- if the acceptance rate for Delayed Rejection moves involving the first (non-heated) chain is 0, or if the average model size in the most heated chain is greater than 10 times the number of observations, the updated value for `b_t`, `b_t*`, is

$$b_t^* = \max \{M_0, b_t - (b_t - 1) / 2\},$$

where M_0 is the user-defined lower bound for `b_t`, (`M_MIN`), stored in the `Temperatures` class, as the first element of member vector `M`;

- if the acceptance rate for Delayed Rejection moves is 1, the updated value for b_t , is

$$b_t^* = \min \{M_1, b_t - (b_t - 1) / 2\},$$

where M_1 is the user-defined upper bound for b_t , (M_MAX), stored in the `Temperatures` class, as the second element of member vector M ;

- if the acceptance rate for Delayed Rejection moves is neither 0 nor 1 but below the optimal value (`Temperatures.optimal`), the updated value for b_t , is

$$b_t^* = \max \{M_0, 2^{\log_2(b_t) - \delta_b}\},$$

where δ_b is automatically calculated at the initialisation of the `Temperatures` class using M_0 , M_1 , the length of the burn-in and `nbatch`, and stored as the member `delta_n` in the `Temperatures` class (see `Classes/Temperatures.cc`);

- if the acceptance rate for Delayed Rejection moves is neither 0 nor 1 but above the optimal value (`Temperatures.optimal`), the updated value for b_t , is

$$b_t^* = \min \{M_1, 2^{\log_2(b_t) + \delta_b}\}.$$

Temperatures are then updated, based on that new value for b_t , according to (3). The new temperature ladder is stored in the `Move_monitor` class (member `temperature_history`).

5.4 Sampling the selection coefficient

When the variable selection coefficient g is not fixed, it is included in the MCMC sampling scheme and its values are sampled using an adaptive Metropolis-within-Gibbs algorithm. At each sweep, a new candidate value for g , g' , is proposed. To ensure $g > 0$, $\log(g')$ is sampled from

$$\mathbb{N}(\log(g), e^{ls}),$$

where g is the value of the variable selection coefficient in the the previous sweep and ls is the log standard deviation of the proposal density for g . The value of ls is user-defined (`G_ADMH_LS`) and stored in the `g_AdMH` class (member `g_AdMH.ls`, see `Classes/g_AdMH.cc` file). The candidate value g' is then accepted/rejected with a probability which depends on the change in the (conditional) posterior probability over all chains induced by the proposed change in the value of g . The acceptance probability also depends on the temperature of each chain (see function `sample_g` in `Routines/moves.cc` file).

Along the MCMC run, the value of `ls` is adapted to control the acceptance rate of the Metropolis-within-Gibbs sampler and keep it close to an optimal values (which is user-defined `G_ADMH_OPTIMAL` and stored in the `g_AdMH` class, member `optimal`). Every `n_batch` sweeps – which is also defined by the user (`G_N_BATCH`) and stored in the `g_AdMH` class (member `n_batch`), depending on the acceptance rate over the past `n_batch` sweeps, the value of `ls` is updated as follows:

- if the acceptance rate for g is below the optimal value, the updated value for `ls`, ls^* is

$$ls^* = \max \{G_0, ls - \delta\},$$

where, G_0 , the lower bound for `ls`, is either specified by the user (`G_M_MIN`), or by default is automatically set to $-\log(p)/2$ at the initialisation of the `g_AdMH` class, and stored as the first element of a vector (member `M` of the `g_AdMH` class). Similarly, G_1 , the upper bound for `ls` is either user-specified (`G_M_MAX`) or set to $\log(p)/2$ and stored as the second element of the vector `M` in the `g_AdMH` class. The value of δ , depends on the current sweep of the sampler (to ensure ergodicity through diminishing adaptation) and also a constant δ_g which is calculated at the initialisation of the `g_AdMH` class using G_0 , G_1 , the length of the burn-in and `n_batch` and stored as the member `delta_n` (see `Classes/g_AdMH.cc`);

- if the acceptance rate for g is above the optimal value (`g_AdMH.optimal`), the updated value for `ls`, ls^* is

$$ls^* = \max \{G_1, ls + \delta\},$$

where, G_1 and δ are defined above.

The history of both g and its parameter `ls` are recorded in the `Move_monitor` class (members `g_sample_history` and `g_adapt_history`, respectively).

6 Overview of the algorithm

One MCMC run of *ESS++* is typically arranged as follows:

1. Load data and initialise hyper-parameters

During this first step, *ESS++* will read the input matrices and the arguments entered by the user (see Section 7.1 for a full listing of the input files). At that stage, all the C++ classes are initialised and their members are set to their default/starting values. The initial value for the latent binary vector for all chains is set equal to the variables selected by a stepwise regression whose parameters are user-defined (see Table 1). When multiple outcomes are considered,

stepwise regression is performed on each outcome. Results are then merged taking the union of the selected variables. It is possible to set the temperature of all chains equal to 1, in which case all chains start from a different and randomly sampled model (see Section 7.4, option `-iso_T` for details).

2. MCMC sampler

The MCMC sampler will run for `n_sweeps` sweeps. This value is user-defined through the command line option `-nsweep`. The user also defines `burn_in` (through `-burn_in` command line option), the number of sweeps to be discarded to allow for burn-in (see Section 7.4). At each sweep, the following sequences of moves will be repeated:

a) *Local move*

At every `Gibbs_n_batch` sweep, a full Gibbs scan will be performed, on the first chain only. This move is included to improve mixing. Next *ESS++* will perform a FSMH with probability `Prob_mut`, which is defined by the user and stored in the `Prior_param` class (member `Prob_mut`).

b) *Global move*

If a local FSMH was not selected, a Crossover move will be performed. Next a Delayed Rejection or All Exchange move will be attempted. During the burn-in, only the Delayed Rejection move is performed. Once the burn-in is completed, the All Exchange move is also enabled and selected with probability $1 - \text{Prob_DR}$, where `Prob_DR` is defined by the user and stored in the `Prior_param` class (member `Prob_DR`). Chains involved in the exchange move are subsequently updated.

c) *Sampling the selection coefficient g*

In cases where the user chooses to sample g , rather than fixing it, g is also updated at each sweep according to the scheme described in Section 5.4. Parameters of the proposal density are adapted along the MCMC run, specifically every `gAdObj.n_batch` sweeps, where `gAdObj` is an object of class `g_AdMH`.

d) *Temperature placement (during the burn-in only)*

As described in Section 5.3, every `TempObj.nbatch` sweeps (where `TempObj` is an object of class `Temperatures`), the temperature ladder is updated such that the acceptance rate of the Delayed Rejection move converges to its optimal value.

At each call of any of these operation, the relevant statistics are stored in the `Move_monitor` object and kept until the end of the run, to feed in the final step of the program.

3. Post-processing

During the post processing step, all the summary statistics from the MCMC run are calculated. All the functions needed for the post-processing are in the file `Routines/post_processing.cc`. These calculations can be decomposed in the following steps:

- a) *Getting the unique list of visited models (function `getUniqueList`)*
Using the history of all visited models during the MCMC run, this function returns the unique list of models visited together with the frequency for each model, i.e. the number of times each model was visited (in the first non-heated chain). If the null model (i.e. the model without any predictors) or any of the models with a single predictor were not visited during the MCMC run, these are added to the list.
- b) *Getting the marginal likelihood of each visited model (function `getLogPost`)*
For all the models in the unique list defined at the previous step, the marginal likelihood is calculated. In cases where g has been sampled, the marginal likelihood for each unique visited model is computed by using the mean value of g across the MCMC sample (discarding the burn in).
- c) *Getting the posterior probability for each visited model (function `getAndSortPostGam`)*
The posterior probability of a unique visited model γ_s , $s = 1, \dots, S^*$, is defined as

$$p_{\gamma_s} = \frac{\exp \{ \log p(\gamma_s | Y) \}}{\sum_{r=1}^{S^*} \exp \{ \log p(\gamma_r | Y) \}},$$

where $\log p(\gamma_s | Y)$ is the log (conditional) posterior probability for each unique visited model and S^* is the number of unique models including the null model and all models with a single covariate, even if they were not visited during the MCMC run.

- d) *Getting the posterior probability of inclusion for each predictor (function `getAndPrintMargGam`)*
The posterior probability of inclusion for each predictor j is defined as

$$\frac{\sum_{s=1}^S \mathbf{1}_{\{\gamma_{s,j}=1\}} p_{\gamma_s}}{\sum_{s=1}^S p_{\gamma_s}},$$

where the sums are over the S non-unique models visited by the sampler (S is equal to the number of sweeps) and $\gamma_{s,j}$ denotes the j^{th} element of latent binary vector associated with model s .

7 Running *ESS++*

ESS++ can be run in a terminal or using a shell file. The minimal command has the form

```
./ESS -X infile1 -Y infile2 -par infile3 -nsweep number1 -burn_in  
number2 -out outfile
```

Some additional features can be set via additional command line options (see Section 7.4). However *ESS++* can be used immediately without considering all the options in this document, by modifying the files specified by the `-X`, `-Y` and `-par` arguments of `Example/ESS_example.sh`.

7.1 Input Files

Predictor matrix (`-X file_name`)/required

This file contains the X matrix, the observed values of predictors (by columns) for each individuals (by rows). The first row of the file is a single scalar representing the number of rows (observations, n) and the second row, a scalar indicating the number of columns (predictors, p). See file `Example/Input/X_example.txt`.

Response matrix (`-Y file_name`)/required

This file contains the Y matrix, the observed values of outcome(s) which can be multi-dimensional (by columns) for each individuals (by rows). The first row of the file is a single scalar representing the number of rows (observations, n), and the second row, a scalar indicating the number of columns (responses, q). See file `Example/Input/Y_example.txt`.

Parameter file (`-par file_name`)

The parameter file (see `Example/Input/par_example.xml`) contains all the user specified parameters required to set up the move classes and options. It is an xml-formatted file containing the members detailed in Table 1. These parameters are not mandatory and, if not specified, they will be set to their default value, also given in Table 1.

Variable name	Default value	Description
General parameters		
<E_P_GAM>	2	The <i>a priori</i> expected model size.
<SD_P_GAM>	1	The <i>a priori</i> standard deviation of the model size.
<NB_CHAINS>	3	The number of chains in ESS algorithm.
Parameters of the stepwise regression		
<N_P_VALUE_ENTER>	0.01	The maximum nominal <i>p</i> -value for a term to be added.
<N_P_VALUE_REMOVE>	0.01	The minimum nominal <i>p</i> -value for a term to be removed.
Setup parameters for the moves		
<GIBBS_N_BATCH>	500	Number of sweeps between two full Gibbs scans.
<P_MUTATION>	0.5	The probability to perform the FSMH move at each sweep at each sweep (see Section 6).
<P_SEL>	0.5	The threshold on the cumulative Boltzmann weights (see Section 5.2.1).
<P_CSRV_R>	0.375	The threshold for the correlation coefficient ρ_0 to be considered in the block crossover move (see Section 5.2.1).
<P_DR>	0.5	The probability to perform a Delayed Rejection move among the two possible exchange moves (see Section 5.2.2).
<G_ADMH_OPTIMAL>	0.44	The optimal acceptance rate for <i>g</i> (see Section 5.4).
<G_N_BATCH>	100	The number of sweeps between two adaptations of the standard deviation of the proposal for <i>g</i> .
<G_ADMH_LS>	0	Initial value for the log standard deviation of <i>g</i> proposal.
<G_M_MIN>	$-\log p/2$	Lower bound for the log standard deviation of <i>g</i> proposal.
<G_M_MAX>	$\log p/2$	Upper bound for the log standard deviation of <i>g</i> proposal.
<K_MAX>	2	Maximum number of breakpoints in the crossover move. This number also defines the number of different crossover moves enabled (see Section 5.2.1)
<B_T>	2	Initial value for the argument <i>b</i> for the temperature ladder (see equation (3), Section 5.3).
<A_T_DEN_INF_5K>	2	Initial value for the argument <i>a</i> for the temperature ladder. This value is considered if $p < 5,000$.
<A_T_DEN_5_10K>	4	Initial value for the argument <i>a</i> for the temperature ladder. This value is considered if $5,000 \leq p < 10,000$.
<A_T_DEN_SUP_10K>	2	Initial value for the argument <i>a</i> for the temperature ladder. This value is considered if $p \geq 10,000$.
<TEMP_N_BATCH>	50	Number of Delayed Rejection moves between temperature placement (see Section 5.3).
<TEMP_OPTIMAL>	0.5	Optimal acceptance rate for the Delayed Rejection move. This value is used in the temperature placement.
<M_MIN>	1.0	Lower bound for the value of b_t in temperature placement (see Section 5.3).
<M_MAX>	4.0	Upper bound for the value of b_t in temperature placement (see Section 5.3).

Table 1: Tags that can appear in the `-par` parameter file.

7.2 Output files

Log file

When *ESS++* is running, summary information describing the initial parameters and the computational time are sent to standard output that can be redirected to a `log` file (see `Example/Output/Example_log` after successfully running the included example). Specifying an additional run time option (`-log`) will also add information to this output about the moves sampled at each sweep as well as information on each chain during the MCMC run. A Boolean variable `DEBUG` set to `false` by default can also be switched to `true` in all `*.cc` files. When `true`, if the program is recompiled, step-by-step monitoring of the MCMC run will be provided in the output. This option is useful to understand how *ESS++* works, but it generates very large files.

Main output files

By default, two main output files are provided by *ESS++*:

1. *Summary statistics about the best visited models*

This file summarizes the best models visited along the MCMC run according to their posterior probability, which is calculated during the post-processing (see Section 6). The number of models described in that file can be specified by the user through the `-top` command line option (see Section 7.4), and if not specified the whole list of unique models visited will be printed out (together with the null and all univariate models). The path and file name where this file is written is automatically defined from the argument of `-out` or `-out_full` option (one of which is required), and has the following form:

```
$1_$2_sweeps_output_best_visited_models.txt
```

where `$1` is the file name (including the full file path) entered by the user as the argument of the `-out` or `-out_full` option, and `$2` is the number of sweeps entered by the user as the argument of `-nsweep` option. The content of the file depends on which output option is chosen by the user, `-out` or `-out_full`. Whatever the output option, models are presented in lines and are sorted according to their posterior probabilities (descending order).

- `-out` option

The `-out` option outputs a simplified summary of the best model visited. As illustrated in the file

```
Example/Output/Example_short_11000_sweeps_output_best_visited_models.txt
```

for each model the following information is provided:

- Rank: the rank of the model according to its posterior probability;
 - #Visits: the number of times the model was visited during the entire MCMC run (including the burn-in);
 - Model_size: the number of variables in the model;
 - log_Post_Prob: the log posterior probability;
 - Model_Post_Prob: the posterior probability of the model;
 - Jeffreys_scale: the Jeffreys' scale value for the; model. This value is defined as $\log_{10}\text{BF}(\gamma_s; \emptyset)$, where $\text{BF}(\gamma_s; \emptyset) = \exp \{ \log p_{\gamma_s} - \log p_{\emptyset} \}$ is the null model Bayes factor associated to the model γ_s ;
 - Model: the last columns of the output lists the variables in the model.
- -out_full option
The -out_full option outputs a more detailed summary of the best models visited. See for example

Example/Output/Example_full_11000_sweeps_output_best_visited_models.txt,

where, for each model, the following information is provided:

- Rank: the rank of the model according to its posterior probability;
- #Visits: the number of times the model was visited during the entire MCMC run (including the burn-in);
- Sweep_1st_visit: the number of sweeps it took *ESS++* to first visit the model;
- #models_eval_before_1st_visit: the number of models evaluated before the first visit to the model;
- Model_size: the number of variables in the model;
- log_Post_Prob: the log posterior probability;
- Model_Post_Prob: the posterior probability of the model;
- Jeffreys_scale: the Jeffreys' scale value for the model;
- Model: the last columns of the output lists the variables in the model.

2. *The marginal posterior probability of inclusion for each predictor*

In this file the posterior probability of inclusion for each predictor (by lines) is presented (Marg_Prob_Incl). As above, the path and file name where this file is written is automatically generated and has the following form:

`$1_$2_iter_output_marg_prob_incl.txt`

where \$1 and \$2 are defined as above.

7.3 Further output files

Other output files can also be generated by ESS++. These will be invoked if the corresponding command line option(s), `-history` and/or `-time` are selected (see Section 7.4).

1. Output enabled by `-time` option

The `-time` option produces a detailed track of the computational time along the MCMC run, in the file `$1_$2_sweeps_output_time_monitor.txt`. Each line of the file represents a sweep of the sampler and shows the absolute computational time (`Time`) and the average computational time *per* model evaluated (`Time_per_eval_model`) during that sweep.

2. Outputs enabled by `-history` option

The `-history` option prints out 11 files whose path and file name are automatically generated and are of the form

`$1_$2_sweeps_output_$3_history.txt`

where \$1 and \$2 are defined as above. These files provide details useful to monitor the behaviour of ESS++ along the run. History files are:

a) *Gibbs move history* (\$3=gibbs)

For every Gibbs scan that was performed the file record the sweep of the sampler (`Sweep`), the number of variables that were added in (`n_0->1`) and the number of variables that were removed (`n_1->0`).

b) *FSMH move history* (\$3=fast_scan)

Each time a FSMH move is sampled along the MCMC run (`Sweep`), this file records: the number of models evaluated (`n_mod`), the number of accepted moves (`n_accept`), the number of proposed and accepted inclusion moves (`n_mod_0_1` and `n_accept_0_1`, respectively), the number of proposed and accepted exclusion moves (`n_mod_1_0` and `n_accept_1_0`, respectively).

c) *Crossover moves history* (\$3=cross_over)

For all sweeps where a crossover move was sampled (`Sweep`), this file reports the type of crossover move sampled (`Move_type`): as detailed in Section 5.2.1, numbers ranging

- from 1 to `k_max`, the maximum number of breakpoints, code for a k -point crossover, and `k_max+1` codes for a block crossover). The number of breakpoints is also reported (`#Breakpoints`), as well as the two selected chains (`Chain_l` and `Chain_r`).
- d) *All Exchange move history* (`$3=all_exchange`)
At each All Exchange move call (`Sweep`), this file simply records the two chains involved (`Chain_l` and `Chain_r`).
 - e) *Delayed Rejection move history* (`$3=delayed_rejection`)
As for the All Exchange history, this files records the sweeps at which a Delayed Rejection move was selected (`Sweep`) and the two chains involved (`Chain_l` and `Chain_r`).
 - f) *g sampling history* (`$3=g`)
This file provides the the sampled values for g (`g`) for each sweep (`Sweep`).
 - g) *g adaptation history* (`$3=g_adaptation`)
Each time the proposal for g is adaptively updated, this files records the current sweep (`Sweep`), the acceptance rate for g since the last update, (`Acceptance_rate`) and the the log-standard deviation of the proposal distribution of g (`log_proposal_std`).
 - h) *Temperature history*¹ (`$3=temperature`)
This file records the sweep at which temperature tuning took place (`Sweep`) and the resulting temperature for each chain.
 - i) *Model history* (`$3=models`)
This file describes the models visited along the MCMC run. For each sweep (`Sweep`), it presents: the number of variables in the model (`Model_size`), the log (conditional) marginal probability and the log (conditional) posterior probability (`log_marg` and `log_cond_post`) and the variables included in the model (`Model`). All values are as at the end of the sweep and only refer to the first non-heated chain.
 - j) *Model size history* (`$3=model_size`)
In this file the model size for each chain at the end of each sweep is reported.
 - k) *Log (conditional) posterior history* (`$3=log_cond_post_prob`)
This files reports the history of the log (conditional) posterior associated to each chain at the end of every sweep.

7.4 Command line options

We summarise the command line options that are implemented in *ESS++*:

¹This file is not produced if the `-iso_T` flag is used (see Section 7.4).

- `-X <filepath>` option (mandatory):
This option specifies the path where to read in the predictor matrix X file.
- `-Y <filepath>` option (mandatory):
This option specifies the path where to read in the outcome matrix Y file.
- `-par <filepath>` option (mandatory):
This option specifies the path where to read in the `xml`-formatted parameter file containing the members listed in Table 1.
- `-nsweep <int>` option (mandatory):
This argument sets the number of sweeps for the MCMC run. No default value is specified because the number of sweeps required for convergence may vary considerably between examples. To determine that the chain has converged, standard MCMC monitoring techniques should be employed, such as viewing trace plots and monitoring autocorrelation in the chains. See for instance Petretto et. al. (2010).
- `-burn_in <int>` option (mandatory):
This argument sets the number of sweeps to be discarded to account for the burn-in. For the same reason as for the `-nsweep` argument no default is specified. The value specified here must be less than the value specified for the `-nsweep` argument.
- `-seed <int>` option (default: current clock time):
This argument sets the random seed used for the pseudo-random number generation.
- `-out <filestem>` option (mandatory unless `-out_full` is used):
This option is an alternative to `-out_full`. It specifies the file stem (including full file path) for writing the standard version of the output files.
- `-out_full <filestem>` option (mandatory unless `-out` is used):
This option is an alternative to `-out`. It specifies the file stem (including full file path) for writing the extended version of the output files.
- `-history` flag (default: disabled):
This flag generates a number of additional output files that record the history of each move. The name of each of these additional files is automatically created. See Section 7.3 for further details.

- `-time` flag (default: disabled):
This flag generates a file that records the time each sweep took. See Section 7.3 for further details.
- `-log` flag (default: disabled):
This flag enables extended reporting in the log file at each sweep (type of the moves sampled, and for each chain, the model size, the posterior probability and marginal likelihood).
- `-top int` option (default: 100):
This option specifies the number of top models to be considered for the output. If not specified all visited models will be considered.
- `-iso_T` flag (default: disabled):
This flag disables the temperature placement within *ESS++* and sets the temperature of each chain to 1.
- `-g_set <double>` option (default: not used):
If specified this option fixes the value of g for duration of the MCMC run and disables the part of the algorithm related to the sampling of g .

8 Example

This section presents further details of the example included in the distributed code. Once the code has been installed and compiled (see Section 2), the example can be run by typing `./ESS_example.sh` in the `ESS/Example` directory. Running this example will generate samples of all of the output files referred to in Section 7.2.

The data for running this example is in the files `ESS/Example/Input/X.txt` and `ESS/Example/Input/Y.txt`. The covariate data in the design matrix X corresponds to $p = 770$ covariates measured on $n = 29$ individuals. The following note describes the data in more detail:

The HXB/BXH Genotype Database, X matrix, was assembled by Robert W. Williams and Michal Pravenec using a compendium of approximately 1100 markers that have been typed over the past decade (please see Pravenec *et al.* 1999 and Jirout *et al.* 2003 for additional details of marker selection and genotyping). The final accepted genotype database contains 556 markers covering all 20 autosomes and the X chromosome. This file was updated November 2007 by RWW to correct some serious errors of maker order and to update the physical positions to the November 2004 assembly of the rat genome

The specified references can be found at:

<http://www.genenetwork.org/dbdoc/HXBBXHGeno.html>.

The response data in the Y matrix records $q = 7$ measurements on the 29 individuals as described in the following references:

Hubner, N. *et al.* Integrated transcriptional profiling and linkage analysis for identification of genes underlying disease. *Nature Genet.* **37**, 243-253 (2005);

Petretto, E. *et al.* New insights into the genetic control of gene expression using a Bayesian multi-tissue approach. *PLoS Comput. Biol.* **6**, e1000737 (2010); and

Heinig, M. *et al.* A trans-acting locus regulates an anti-viral expression network and type 1 diabetes risk. *Nature* **467**, 460-464 (2010);

We are grateful for the use of this data.

Examining the command in the `ESS/Example/ESS_example.sh` script we can see the command line options that were invoked. As this is only an illustrative run we have used a very small number (11000) of sweeps (option `-nsweep 11000`). Of these, the first 1000 are discarded as burn in. (option `-burn_in 1000`). We have also restricted the output to the best 100 models (option `-top 100`). In order to produce complete examples of all the output files we have used `out_full` rather than `out` and enabled the `-history`, `-time`, and `-log` flags (see Section 7.4 for full details).

Finally we refer to the example parameter file `ESS/Example/Input/par_example.xml` that is used for this example. Comparing the values set in this file with Table 1, it is evident that we have set many of the parameters to take their default values. The exceptions to this are: the number of parallel chains (`<NB_CHAINS> 4` instead of 3); the stepwise regression p -values (`<N_P_VALUE_*> 0.1` instead of 0.01); the threshold for the crossover correlation coefficient (`<P_CRSV_R> 0.5` instead of 0.375); the maximum number of breakpoints (`<K_MAX> 3` instead of 2); and the number of delayed rejection attempts before the temperature should be adjusted (`<TEMP_N_BATCH> 100` instead of 50).