GUESS: Graphical Unit Evolutionary Stochastic Search for Bayesian model exploration

Documentation, V1.1

June 2013

Contents

1 Overview

GUESS is an extension of *ESS++* which enables to redirect computationally intensive linear algebra operations to the Graphics Processing Unit (GPU). As such it represents a computationally optimised C++ implementation of a fully Bayesian variable selection approach that can analyse, in a genome-wide context, single and multiple responses in an integrated way.

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. *GUESS* 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. *GUESS* works well for problems where $n > p$ and also $n < p$, including the case where $n << p$ (known as the "large p, small n"case). In the current version, *GUESS* can handle several thousands of observations ($n < 20,000$), hundreds of thousands of predictors ($p < 1,000,000$) and a few responses simultaneously ($q < 10$). Since *GUESS* searches for subsets of covariates that jointly predict *all* the responses, the type and size q of the responses should be chosen carefully. The significant gain in computational efficiency gained in this novel implementation compared to *ESS++* was achieved by using GPU-based and optimised linear algebra libraries (CULA). While these libraries yield faster (over one order of magnitude) matrix operations such as multiplication or decomposition, they imply extensive data transfer from the memory/CPU to the GPU, which in-turn can be computationally expensive. As a consequence, for smaller X matrices ($i.e.\ n < 1,000$, $p < 10,000$), for which the matrix operations are not rate-limiting, the CPU version of *GUESS* may be faster. Hence, to ensure both an optimal use of the algorithm, and to enable running *GUESS* on non-CULA compatible systems the call to GPU-based calculations within *GUESS* can easily be switched off by the user (see below).

The model used by *GUESS* 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 $\gamma = (\gamma_1, \ldots, \gamma_p)^T$, with $\gamma_j = 1$ if $\beta_j \neq 0$ and $\gamma_j = 0$ if $\beta_j = 0$, $j = 1, \ldots, p$, the Gaussian linear model (1) becomes

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

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 $\gamma_j = 1$.

GUESS 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 genetics algorithms. Specifically, *GUESS* 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.

For further details about *GUESS* see:

Bottolo L, Chadeau-Hyam M, Hastie DI, Langley SR, Petretto E, Tiret L, Tregouet D and Richardson S. (2011). *ESS++*: a *C++* objected-oriented algorithm for Bayesian stochastic search model exploration. *Bioinformatics* **27**(4), 587–588.

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.

***

Please report any problem or bug to m.chadeau@imperial.ac.uk or l.bottolo@imperial.ac.uk.

***

2 Installation of *GUESS*

Prior to the installation of *GUESS* please check the two following points:

1. *GUESS* 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
     ```

2. *GUESS* also offers the possibility to use proprietary GPU-optimised linear algebra libraries from CULA (version $R10$ and more recent version as CULA dense):

   `http://www.culatools.com/`

   The installation of the library is platform-dependent and is documented on the website. It may also require the installation of $nvidia$ CUDA drivers from:

   `http://developer.nvidia.com/cuda-downloads`

   Users who do not wish to purchase and install CULA libraries can still use GUESS in its CPU version (see below).

To install *GUESS*, please follow the steps listed below

1. Download `GUESS_Release_1.1.tgz` and unpack it by typing `tar zxvf GUESS_Release_1.1.tgz` to generate the `GUESS_Release_1.1` directory.

2. Type `cd GUESS_Release_1.1/Main`. The `makefile` provided has been developed to compile `GUESS` using the 32-bits version of CULA version *R10*. Compilation using different version may require some edits in the `makefile`:

   (a) for non-CULA users, open the `makefile` and set the variable `CUDA` to `CUDA=0`

   (b) for CULA users, checks the features of your CULA installation:

   - for the 64 bits version of CULA change the `LFLAGS` to:

     ```
     LFLAGS = $(LFLAGSTMP) -L/usr/local/cuda/lib64
                  -L/usr/local/cula/lib64
     ```

     i.e. replace `lib` with `lib64`.

   - For the *R11*-CULA dense version change the `LIBS` to:

     ```
     LIBS = $(LIBSTMP) -lpthread -llapack -lcuda -lcublas
         -lcudart -lgslcblas -lcula_core -lcula_lapack
     ```

     i.e. replace `-lcula` with `-lcula_core -lcula_lapack`

   (c) *GUESS* has been developed and tested using the `g++` and `icc` compilers on several Linux and Mac platforms. Compiling it on other systems may require further edits in the `makefile`. For instance the path to the GSL library may need to be changed.

3. Once the `makefile` has been appropriately updated, compile *GUESS* by simply typing `make` in the `GUESS_Release_1.1/Main/` directory.

4. To check the installation, and create the example files mentioned below, move to the `GUESS_Release_1.1/Example` directory and run the example script by typing `./GUESS_example.sh`

   (a) If the code has been compiled with `CUDA=1` option, *GUESS* will only use the GPU-resources if the `-cuda` option is in the command line. To disable GPU calculations from GUESS, which is recommended for small data sets (as the example files provided), simply remove the `-cuda` option from the command line, or the script file used to run *GUESS* (e.g. `GUESS_example.sh`)

   (b) If the code has been compiled with `CUDA=0` option, the `-cuda` option in the command line will be ignored.

3 Contents of the *GUESS* package

3.1 `Main` directory

This directory contains the *GUESS* source code. The main program is `GUESS.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 *GUESS* works, but it usually generates very large `log` files.

3.2 `Classes` directory

This directory contains all the C++ classes used in *GUESS*. The classes typically relate to one feature of the model and contain the associated parameters. There is also a class (`Move_monitor`) to compile 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 *GUESS* main code.

3.4 `Example` directory

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

4 Running *GUESS*

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

```
./GUESS -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 4.3). However *GUESS* 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/GUESS_example.sh`.

4.1 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`. *GUESS* enables to include confounders in the model, these variables will be included in all models visited. The number of confounders has to be specified through `-nconf` command line option, and the first `nconf` columns of the predictor matrix must contain observed values for these covariates in the population of interest.

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 values, also given in Table 1.

Init file (`-init file_name`)

This file (see `Example/Input/Init_example.txt`) specifies which variable to include at the first run of the MCMC algorithm. The first row of the file is a single scalar representing the number of rows (# variables to include). Subsequent rows indicate the position of the covariates to include. This file is optional and if not specified, initial guesses of the MCMC algorithm will be derived from a step-wise regression approach.

4.2 Main Output files

Log file

When *GUESS* 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 *GUESS* works, but it generates very large files.

Main output files

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

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 7). The number of models described in that file can be specified by the user through the `-top` command line option (see Section 4.3), and if not specified the whole list of unique models visited will be printed out (together with the null and all univarite 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. 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 $\log_{10}\text{BF}(\gamma_s;\varnothing)$, where $\text{BF}(\gamma_s;\varnothing) = \exp\{\log p_{\gamma_s} - \log p_{\varnothing}\}$ 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/GUESS_example_11000_sweeps_output_best_visited_models.txt`,

  where, for each model, the following information is added:
  - `Sweep_1st_visit`: the number of sweeps it took *GUESS* to first visit the model;
  - `#models_eval_before_1st_visit`: the number of models evaluated before the first visit to 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.

Other output files may be obtained from command lines options (see 5).

4.3 Command line options

1. **Mandatory options**

   - `-X` option:
     This option specifies the path where to read in the predictor matrix X file.
   - `-Y` option:
     This option specifies the path where to read in the outcome matrix Y file.
   - `-nsweep int` option:
     The argument of this option is an integer specifying the number of sweeps for the MCMC run (including the burn-in).
   - `-burn_in int` option:
     The argument of this option is an integer specifying the number of sweeps to be discarded to account for the burn-in.
   - `-out/out_full file_name` option:
     This option is an alternative to `-out_full`. It specifies the file stem for writing the standard/extended version of the output file respectively.

2. **Running options**

   - `-Egam int` option:
     this option specifies the (un-truncated) 'a priori' average model size.
   - `-Sgam int` option:
     this option specifies the (un-truncated) 'a priori' standard deviation of the model size.
   - `-n_chain int` option:
     this option specifies the number of parallel chains to include in the evolutionary algorithm.
   - `-cuda` option:
     If selected, this option will enable GPU-calculations implemented in *GUESS*. If *GUESS* has been compiled with `CUDA=0` option, `-cuda` will be ignored.
   - `-seed int` option:
     The argument of this option is an integer specifying the random seed used for the pseudo-random number generation. If not specified it will be set to the current clock time.
   - `-timeLimit double` option:
     The argument of this option is an scalar defining the maximum duration (in hours) of the run. If the run time reaches this limit, every information enabling the run to be resumed (e.g. the status of the random number generator, etc ...) will be written out.

- `-resume` option:
  This flag will resume a run which has been halted after reaching the `timeLimit` specified in a previous run. In practice to resume a halted flag, simply change the `log` file name to prevent overwriting, and add the `-resume` option to the original command line.
- `-postProcess` option:
  This option enables the post-processing from a halted run.
- `-extend int` option:
  This flag will extend a previous run for an additional number of iterations (specified by the argument of the option). In practice to extend for `N` iterations an already finished run, simply change the `log` file name to prevent overwriting, and add the `-extend N` option to the original command line.
- `-noRescaleX` option:
  If chosen, this option disables the standardisation of the X matrix.

3. **Modelling options**

- `-par` option:
  This option specifies the path where to read in the `xml`–formatted parameter file containing the members listed in Table 1. If unspecified, parameters will be set to the default values also listed in Table 1.
- `-lambda double` option:
  If selected, this option disbales the g-prior setting implemented by default. Any negative value for the argument of this option switches on the independent prior model (see section 6).
- `-init` option:
  This option specifies the path where to read in the position of the variable to include in the first model. If unspecified, the first model will be derived from a step-wise regression.
- `-nconf int` option:
  This option specifies the number of confounders to be considered. The first `nconf` columns of `X` will be included in all model visited.
- `-iso_T` flag:
  The use of this flag disables the temperature placement within *GUESS* and sets the temperature of each chain to 1.

- `-g_set double` option:
  This option specifies the value of g to be considered along the whole run and disables the part of the algorithm related to the sampling of g.

4. **Output options**

- `-history` flag:
  This flag generates a number of additional output files that record the history of each move. See Section 5 for further details.
- `-time` flag:
  This flag generates a file that records the time each sweep took (see Section 5).
- `-log` flag:
  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:
  This option specifies the number of top models to be considered for the output. If not specified all visited models will be considered.

Additional Information

5 Further output files

Other output files can also be generated by GUESS. These will be invoked if the corresponding command line option(s), -history and/or -time are selected (see Section 4.3).

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 *GUESS* 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): 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.

[1] This file is not produced if the `-iso_T` flag is used (see Section 4.3).

6 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

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

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

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

where $\mathcal{IW}\left(\cdot,\cdot\right)$ denotes the inverse Wishart distribution with $\delta = 3$ and $Q = kI_q$. The hyperparameter k is automatically elicited by *GUESS* 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 *GUESS* we opt for the g-prior setting

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

where:

- g, the variable selection coefficient, is a scalar;
- g is either fixed (`-g_set option`) or unknown and sampled (implemented by default). In the latter case, a prior density is specified, $g \sim \Gamma\left(1/2, n/2\right)$, and included in the MCMC sampling scheme, where $\Gamma\left(\cdot,\cdot\right)$ 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 (enabled by using `-lambda` option with a null or negative argument)

$$H_\gamma = gI_p$$

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 4):

- `E_p_gam` (denoted E_{p_γ} in eq. 3), the *a priori* expected model size for the un-truncated model distribution;
- `SD_p_gam` (denoted σ_{p_γ} in eq. 3), the *a priori* standard deviation of the model size for the un-truncated distribution.
- `MAX_P_GAM_FACTOR` (denoted F_{p_γ} in eq. 3), the factor defining the maximal model size to be considered $p_{X_{max}}$ from

$$p_{X_{max}} = E_{p_\gamma} + \sigma_{p_\gamma} \times F_{p_\gamma} \tag{3}$$

7 Overview of the MCMC algorithm

1. **Data loading and hyper-parameters initialisation**
   During this first step, *GUESS* will read the input matrices and the arguments entered by the user (see Section 4.1 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 4.3, 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 4.3). At each sweep, the following sequences of moves will be repeated:

   a) *Local moves (see section 8.1)*
   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 *GUESS* 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 moves (see section 8.2)*
   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-`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 (see section 8.4)*
In cases where the user chooses to sample g, rather than fixing it, g is also updated at each sweep. 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)* (see section 8.3)
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 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 time 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, $s = 1, \ldots, S^*$, 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.

8 EMC implementation

The principle of *GUESS* 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 (the number of chains is specified through `NB_CHAINS` in the parameter file), 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 *GUESS*.

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).

8.1 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 $l \in \{1, 2, \ldots, L\}$ 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 jth 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 *GUESS*:

1. *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`).

2. *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 current model size;
   - ω, an hyper-parameter automatically calculated (and stored in the `Prior_param` class) which dependents 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 $0 \to 1$ moves proposed and the number of accepted moves;
- the number of $1 \to 0$ 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 4). If a local FSMH move is not selected for a particular sweep, then a global crossover move is attempted instead.

8.2 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 *GUESS*: crossover moves and exchange moves.

8.2.1 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 4) 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 *GUESS* . 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 $\rho(X_j, X'_j)$ $j' = 1, \ldots, p$, $j \neq j'$, 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_pbty_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.

8.2.2 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 *GUESS* 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 8.3) 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.

8.3 Temperature placement

The temperature tuning during the burn in of the MCMC run is a key feature in *GUESS*: 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 *GUESS* is geometric. For a given chain $l \in \{1, \ldots, L\}$, the l^{th} temperature is defined as

$$t_l = b^{a_l}, \tag{4}$$

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 4 - 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, $\texttt{b\_t}^*$, is

$$\texttt{b\_t}^* = \max\left\{M_0, \texttt{b\_t} - (\texttt{b\_t} - 1)/2\right\},$$

  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

$$\texttt{b\_t}^* = \min\left\{M_1, \texttt{b\_t} - (\texttt{b\_t} - 1)/2\right\},$$

  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 neither 0 nor 1 but below the optimal value (Temperatures.optimal), the updated value for b_t, is

$$\texttt{b\_t}^* = \max\left\{M_0, 2^{\log_2(\texttt{b\_t}) - \delta_b}\right\},$$

  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

$$\texttt{b\_t}^* = \min\left\{M_1, 2^{\log_2(\texttt{b\_t}) + \delta_b}\right\}.$$

Temperatures are then updated, based on that new value for b_t, according to (4). The new temperature ladder is stored in the Move_monitor class (member temperature_history).

8.4 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

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

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`, $\texttt{ls}^*$ is

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

  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`, $\texttt{ls}^*$ is

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

  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).

Variable name	Default value	Description	Ref.
General parameters			6
<MAX_P_GAM_FACTOR>	10	The factor specifying the maximal model size from equation (3).	6
Parameters of the stepwise regression			
<N_P_VALUE_ENTER>	0.01	The maximum nominal p-value for a term to be added.	
<N_P_VALUE_REMOVE>	0.01	The minimum nominal p-value for a term to be removed.	
Setup parameters for the moves			
<GIBBS_N_BATCH>	500	Number of sweeps between two full Gibbs scans.	8.1-1
<P_MUTATION>	0.5	The probability to perform the FSMH move at each sweep.	8.1-2
<P_SEL>	0.5	The threshold on the cumulative Boltzmann weights.	8.2.1
<P_CSRV_R>	0.375	The threshold for the correlation coefficient ρ_0 to be considered in the block crossover move.	8.2.1
<K_MAX>	2	Maximum number of breakpoints in the crossover move. This number also defines the number of different crossover moves enabled.	8.2.1
<P_DR>	0.5	The probability to perform a Delayed Rejection move among the two possible exchange moves.	8.2.2
<G_ADMH_OPTIMAL>	0.44	The optimal acceptance rate for g.	8.4
<G_N_BATCH>	100	The number of sweeps between two adaptations of the standard deviation of the proposal for g.	8.4
<G_ADMH_LS>	0	Initial value for the log standard deviation of g proposal.	8.4
<G_M_MIN>	$-\log p/2$	Lower bound for the log standard deviation of g proposal.	8.4
<G_M_MAX>	$\log p/2$	Upper bound for the log standard deviation of g proposal.	8.4
<B_T>	2	Initial value for the argument b for the temperature ladder.	8.3
<A_T_DEN_INF_5K>	2	Initial value for the argument a for the temperature ladder. This value is considered if $p < 5,000$.	8.3
<A_T_DEN_5_10K>	4	Initial value for the argument a for the temperature ladder. This value is considered if $5,000 \leq p < 10,000$.	8.3
<A_T_DEN_SUP_10K>	2	Initial value for the argument a for the temperature ladder. This value is considered if $p \geq 10,000$.	8.3
<TEMP_N_BATCH>	50	Number of Delayed Rejection moves between temperature placement.	8.3
<TEMP_OPTIMAL>	0.5	Optimal acceptance rate for the Delayed Rejection move. This value is used in the temperature placement.	8.3
<M_MIN>	1.0	Lower bound for the value of b_t in temperature placement.	8.3
<M_MAX>	4.0	Upper bound for the value of b_t in temperature placement.	8.3

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