Chapter 12

Genome-Enabled Prediction Using the BLR (Bayesian Linear Regression) R-Package

Gustavo de los Campos, Paulino Pérez, Ana I. Vazquez, and José Crossa

Abstract

The BLR (Bayesian linear regression) package of R implements several Bayesian regression models for continuous traits. The package was originally developed for implementing the Bayesian LASSO (BL) of Park and Casella (J Am Stat Assoc 103(482):681–686, 2008), extended to accommodate fixed effects and regressions on pedigree using methods described by de los Campos et al. (Genetics 182(1):375–385, 2009). In 2010 we further developed the code into an R-package, reprogrammed some internal aspects of the algorithm in the C language to increase computational speed, and further documented the package (Plant Genome J 3(2):106–116, 2010). The first version of BLR was launched in 2010 and since then the package has been used for multiple publications and is being routinely used for genomic evaluations in some animal and plant breeding programs. In this article we review the models implemented by BLR and illustrate the use of the package with examples.

Key words Genomic selection, Whole-genome prediction, Bayesian, Shrinkage and genomic prediction

1 Models, Algorithms, and Data

In whole-genome regression (WGR, e.g., [1–4]) models, phenotypes $\mathbf{y} = \{y_i\}_{i=1}^{n}$ are regressed on large number of markers concurrently [1–3]. The number of markers (p) can vastly exceed the number of individuals with records (n) and implementing these "*large-p-with-small-n*" regressions requires using estimation methods that perform variable selection, shrinkage of estimates, or a combination of both. Bayesian procedures are commonly used for variable selection and for deriving shrunken estimates of effects in regressions models (*see* ref. 5 for a review of methods commonly used in WGR). The BLR package of R ([6], http://www.r-project.org/) implements several types of Bayesian regressions. In this section we describe the models implemented and the algorithms used by BLR.

Bayesian inference is based on the posterior density of the parameters (θ) given the data ($\mathbf{y}$). Following Bayes' theorem, this

Cedric Gondro et al. (eds.), *Genome-Wide Association Studies and Genomic Prediction*, Methods in Molecular Biology, vol. 1019, DOI 10.1007/978-1-62703-447-0_12,

is proportional to the product of the conditional distribution of the data given the unknowns (or Bayesian likelihood) times the prior density assigned to θ, these objects are described next.

Bayesian Likelihood. In BLR the conditional density of the phenotypes is a multivariate normal density. The data equation is:

$$\begin{aligned} y_i &= \mu + \sum_{j=1}^{P_F} x_{Fij}\beta_{Fj} + \sum_{j=1}^{p_R} x_{Rij}\beta_{Rj} + \sum_{j=1}^{p_L} x_{Lij}\beta_{Lj} + u_i + \varepsilon_i \\ &= \eta_i + \varepsilon_i \quad (i = 1, \ldots, n) \end{aligned} \tag{1}$$

where

- y_i is a phenotypic record.
- μ is an intercept.
- x_{Fij}, x_{Rij}, and x_{Lij} are covariates whose effects are assigned flat priors (β_{Fj}), normal identical-independent priors (β_{Rj}), and identical-independent double exponential priors (β_{Lj}), respectively.
- u_i is a Gaussian random effect with mean zero and covariance matrix given by the user (e.g., a numerator relationship matrix).
- ε_i are model residuals, each following independent normal densities with mean zero and variance equal to $w_i^2\sigma_\varepsilon^2$, where w_i is a user-defined weight used to accommodate heterogeneous residual variances, and σ_ε^2 is a variance parameter (if weights are not provided, by default, the w_i's are all set equal to one).
- $\eta_i = \mu + \sum_{j=1}^{p_F} x_{Fij}\beta_{Fj} + \sum_{j=1}^{p_R} x_{Rij}\beta_{Rj} + \sum_{j=1}^{p_L} x_{Lij}\beta_{Lj} + u_i$ is the linear predictor used to model the conditional expectation function. Any of the variables in the linear predictor can be omitted, except for the intercept, which is included by default.

From the above assumptions, it follows that the conditional density of the data given the unknowns is:

$$p(\mathbf{y}|\mu, \beta_F, \beta_R, \beta_L, \mathbf{u}, \sigma_\varepsilon^2) = \prod_{i=1}^{n} N(y_i|\eta_i, w_i^2\sigma_\varepsilon^2) \tag{2}$$

where $\beta_F = \{\beta_{Fj}\}$, $\beta_R = \{\beta_{Rj}\}$, $\beta_L = \{\beta_{Lj}\}$, and $\mathbf{u} = \{u_i\}$ are vectors of effects.

Prior. In BLR, μ and β_F are assigned flat priors, $p(\mu, \beta_F) \propto 1$; therefore, these unknowns are estimated using information from the likelihood solely. The remaining effects, β_R, β_L, and $\mathbf{u}$, are assigned informative priors, yielding shrinkage of estimates of these effects. The effects $\{\beta_{Rj}\}$ are assigned identical-independent normal priors centered at zero and with variance σ_R^2, i.e., $p(\beta_R|\sigma_R^2) = \prod_{j=1}^{P_R} N(\beta_{Rj}|0, \sigma_R^2)$. This prior, combined with a Gaussian likelihood, yields estimates of marker effects similar to those of a

Ridge Regression [7]; therefore, we denote this model as BRR (Bayesian Ridge Regression). The priors assigned to effects $\{\beta_{Lj}\}$ are identical-independent double-exponential densities, represented, as in the Bayesian LASSO (BL) of Park and Casella [1], as infinite mixtures of scaled normal densities, $p(\beta_L|\lambda,\sigma^2_\varepsilon) = \prod_{j=1}^{P_L} \int N(\beta_{lj}|0,\sigma^2_\varepsilon\tau^2_j)\mathrm{Exp}(\tau^2_j|\lambda)\,\partial\tau^2_j$, where τ^2_j is a marker-specific scale parameter, $\mathrm{Exp}(\tau^2_j|\lambda)$ is an exponential density assigned to τ^2_j, with rate parameter $\lambda^2/2$. This parameter controls the extent of shrinkage of estimates of effects in the same way that the ratio $\sigma^2_\varepsilon\sigma^{-2}_R$ does in the BRR. The vector of effects **u** are assigned a multivariate normal density with mean zero and a covariance matrix proportional to **A**, i.e., $p(\mathbf{u}|\sigma^2_u) = N(0,\mathbf{A}\sigma^2_u)$. For pedigree-regressions, **A** could be a numerator relationship matrix computed from a pedigree (*see* **Note 1** for an example of how to use the package `pedigreemm`, [8, 9] for creating **A** from a pedigree), but in principle **A** can be any symmetric positive definite matrix.

The unknown variance parameters $\{\sigma^2_\varepsilon,\sigma^2_R,\sigma^2_u\}$ are assigned Scaled-inverse Chi-square priors. In the parameterization used in BLR, the prior expectation is $E(\sigma^2|df_\cdot,S_\cdot) = S_\cdot/(df_\cdot - 2)$, *see* **Note 2** for further details about this parameterization. The regularization parameter of the BL (λ) can be either fixed at a user-specified value or treated as random. When λ is treated as unknown one can either assign a gamma density to λ^2 with rate and shape parameters r and s, respectively, $p(\lambda^2|r,s) = G(\lambda^2|r,s)$ or, alternatively, the ratio of λ to a prespecified upper-bound of λ (max) could be assigned a Beta prior (*see* ref. 2, for further discussion about this prior).

Posterior density. Collecting the aforementioned assumptions, we obtain the joint posterior density of the unknowns, given the data:

$$\begin{aligned} p(\mu,\beta_F,\beta_R,\beta_L,\mathbf{u},\sigma^2_\varepsilon,\sigma^2_R,\sigma^2_u,\lambda|\mathbf{y}) &\propto \prod_{i=1}^{n} N\left(y_i|\eta_i,w^2_i\sigma^2_\varepsilon\right) \\ &\times \left\{\prod_{j=1}^{P_R} N\left(\beta_{Rj}\middle|0,\sigma^2_R\right)\right\}\left\{\prod_{j=1}^{P_L} N\left(\beta_{lj}\middle|0,\sigma^2_\varepsilon\tau^2_j\right)\mathrm{Exp}\left(\tau^2_j\middle|\lambda\right)\right\} \\ &N\left(\mathbf{u}|\mathbf{0},\mathbf{A}\sigma^2_u\right)\times\chi^{-2}\left(\sigma^2_\varepsilon\middle|df_\varepsilon,S_\varepsilon\right)\chi^{-2}\left(\sigma^2_R\middle|df_R,S_R\right)\chi^{-2}\left(\sigma^2_u\middle|df_u,S_u\right)p(\lambda) \end{aligned} \tag{3}$$

Algorithms The posterior density of Eq. 3 does not have closed form; however, all the fully conditional densities (except that of λ when $\lambda/\max$ is assigned a Beta prior) have closed form and are easy to sample from. Therefore, in BLR samples are drawn using a Gibbs sampler (*see* **Note 3**) with scalar updates. At each iteration of the Gibbs sampler, the algorithm loops over predictors. In *large-p-regressions*, this can be computationally intensive. Therefore, we have developed a C-function, which produces samples of effects. The sampling of variance parameters can be vectorized; therefore,

these updates can be done efficiently using existing R functions such as `rchisq` of the base package or `rinvGauss` of the SuppDists R-package (http://cran.r-project.org/web/packages/SuppDists/index.html).

Data. A genomic wheat dataset is made available with the package. This dataset is from CIMMYT's (http://www.cimmyt.org) early spring trials and comprises phenotypic, genotypic and pedigree information from 599 pure lines of wheat. The following code illustrates how this data can be loaded into the R session (Box 1).

Box 1: Loading the wheat dataset

```
1  library(BLR)
2  data(wheat)
3  objects()
```

Once the dataset is loaded, the following objects become available in the environment:

- **Y** (599 × 4) a matrix containing an evaluation of the grain yields of 599 wheat lines in four different environments. Phenotypes represent average performance of two years and were centered and standardized to a unit sample variance.
- **A** (599 × 599) a numerator relationship matrix derived from pedigree data (the matrix was computed using the methods described in [10]).
- **X** (599 × 1,279) a matrix containing the genotypes of the 599 wheat lines at 1,279 Diversity Array Technology (DART) markers generated by Triticarte Pty. Ltd. (Canberra, Australia; http://www.triticarte.com.au). Since the wheat data consists of pure lines, the two homozygous were coded as 0 and 1.
- **sets** (599 × 1) a vector assigning the wheat lines into tenfolds, this could be used for cross-validation.

2 Using BLR

In this section we introduce several examples that illustrate the use of the BLR package for regressions using molecular markers and pedigree. The main function of the BLR-package is `BLR()`.

Inputs to BLR include **phenotypes** (denoted as y, numeric vector), **design matrices** for fixed (XL, numeric matrix) and random effects (XR for BRR, XL for BL, both numeric matrices), **prior hyper-parameters** (prior, a list; further description given below), MCMC parameters, including number of iterations (nIter, integer), number of iterations to be discarded as burn-in (burnIn, integer), thinning interval (thin, integer), and a string (saveAt, character) that can be used to provide a path and append

a prefix to the files containing the samples generated by the algorithm. A complete description of the inputs is given in R-help (accessible typing `help(BLR)` in the R-console or at http://cran.r-project.org/web/packages/BLR/index.html). All arguments, except the vector of phenotypes, have default values. By default, BLR runs an intercept model with improper priors; we discourage the use of improper priors for variance parameters and for λ, because when improper priors are used for these unknowns the posterior density is not guaranteed to be a proper density. Further details about inputs and how to choose appropriate values for inputs are given in the various examples included in this section.

Processes. As BLR runs, two processes occur: (a) the function **prints to the screen information on the iterations of the sampler** (iteration number, time per iteration, residual variance, etc.) and (b) **samples** of fixed effects, of variances and of λ **are saved to the hard drive** (*see* **Note 4** for further information about the files that are created by BLR).

Return. As the Gibbs sampler is running, BLR computes posterior means and other statistics. Once the program finishes, BLR returns a list containing estimates of posterior means and posterior deviations, measures of goodness of fit and information on the model that was fitted. Further details are given in the following examples.

2.1 Fitting an Intercept Model

The simplest model that can be fitted using BLR is the intercept model; an example is given in Box 2. The first line of the code removes any object which may have been saved from a previous R-session, the second line loads BLR, the third line sets the seed for the random number generator, and data is simulated in line 4 by drawing samples from a normal density with mean 25 and variance equal to 4. The fifth line fits the intercept model using BLR. The argument y in BLR is used to provide phenotypes; it must be a numeric vector and missing values (NA) are allowed. In the example, in addition to phenotypes, we indicate the number of iterations of the Gibbs sampler (1,200) and the number that we want to discard as burn-in (200 in the example). Note that in the example in Box 2 we did not provide any predictor variable; the intercept is included by default.

Box 2. Fitting an intercept model (improper prior)

```
1  rm(list=ls())
2  library(BLR)
3  set.seed(12345)
4  y<-rnorm(1000,mean=25,sd=2)
5  fm<-BLR(y=y,nIter=1200,burnIn=200,
6          saveAt="box2_")
7  str(fm)
8  fm$mu
9  fm$varE
10 mean(y)
11 var(y)
```

You may have noticed that immediately after the welcome message and before the Gibbs sampler started, the program printed the following warning:

```
==========================================
No prior was provided; BLR is running with
improper priors.
==========================================
```

This occurs because we did not provide prior hyper-parameters; by default BLR runs improper priors.

2.2 Fitting Fixed-Effects Models

Box 3 provides an example of a model that includes an intercept and 5 fixed effects; in this example, we arbitrarily chose to fit the "fixed" effects of markers whose genotypes are given in columns 100, 200, 300, 400, and 500 of the matrix containing the wheat genotypes ($\mathbf{X}$). BLR assigns flat priors (i.e., proportional to a constant) for fixed effects and for the intercept. For the residual variance, the prior is a scaled inverse chi square density. This density has two hyper-parameters: the degree of freedom (df) and the scale (S). If we do not provide them, BLR sets these to zero, which yields an improper prior (*see* example in Box 2).

Choosing the prior for the residual variance. We recommend setting df equal to a small value to reduce the influence of the prior on inferences (e.g., $df = 5$; any value greater than zero is valid, but values less than 4 will yield infinite prior expected value of σ^2_ε), and suggest choosing a scale parameter to match our prior expectations about the residual variance. The prior expectation for this parameter in BLR is: $E(\sigma^2_\varepsilon | df_\varepsilon, S_\varepsilon) = S_\varepsilon/(df_\varepsilon - 2)$ (*see* **Note 2** for details about the parameterizations of the scaled-inverse chi square density used in BLR). Therefore, if V_y represents an estimate of the phenotypic variance and x is the proportion of that variance that we expect to account for with the model, we can set the scale parameter to be $S_\varepsilon = V_y(1 - x)(df_\varepsilon - 2)$. In Box 3, we apply this approach using $x = 0.5$. After the prior is defined (lines 11–13 of Box 3) the model is fitted (lines 16 and 17 of Box 3). In lines 19–23 of Box 3 we show how to extract parameter estimates (estimated posterior means) and in lines 25–30 we show how to read into R and plot samples collected by the Gibbs sampler (*see* **Note 3**).

Box 3. Fitting a fixed effects model

```
rm(list=ls())
library(BLR) ; library(coda)
set.seed(12345)
data(wheat)

# Extract phenotype and markers
 y<-Y[,1]
Z<-X[,c(100,200,300,400,500)]

# Defines the prior
DF<-5
S<-0.5*var(y)*(DF-2)
prior=list( varE=list(df=DF,S=S) )

# Fits the model
fm<-BLR(y=y,XF=Z,nIter=5500,burnIn=500,
    prior=prior,saveAt="box3_")

str(fm)
fm$mu
fm$varE
fm$bF
fm$SD.bF

# Trace plot of  the effect of the (1st marker)
B<-read.table("box3_bF.dat")
plot(B[,1],type="o",col="red")
HPDinterval(as.mcmc(B),prob=.95)
```

For further information about how BLR handles and stores the samples *see* **Note 4**, and for additional information about how to create incidence matrices for fixed or random effects *see* **Note 5**.

2.3 Fitting Marker Effects as Random

In WGR models [4], phenotypes are regressed on genome-wide markers. With current genotyping technologies, the number of markers can vastly exceed the number of phenotypes. When effects are fitted as fixed, the sampling variance of estimates, and consequently the mean-square error (MSE), increases fast as the number of markers does. To overcome this problem, WGR are usually implemented using shrinkage estimation procedures such as penalized or Bayesian regressions. These procedures introduce bias but reduce the variance of estimates and, when the number of predictors is large relative to the number of data-points, the use of shrinkage yields smaller MSE than that of standard estimation procedures such as ordinary least squares, or maximum likelihood. For regressions on markers, BLR implements two Bayesian shrinkage procedures: one uses Gaussian priors (BRR) and the other one uses double-exponential prior densities (BL). Each of these methods is described next.

2.3.1 Bayesian Ridge Regression

Box 4 provides an example where phenotypes from the wheat dataset are regressed on all available markers using a BRR. In this model, we need to provide hyper parameters for the prior of residual variance and for that of the variance of marker effects, σ^2_R. The scale and *df* parameters for the residual variance are defined using the same principles discussed in the example in Box 3.

We use a similar principle for the variance of marker effects. Noting that the variance of the regression on markers is $\text{Var}(\sum_{j=1}^{p_R} x_{Rij}\beta_{Rj}) = \sigma^2_R \sum_{j=1}^{p_R} x^2_{Rij}$ and summing over individuals and dividing by n, we get $n^{-1}\sum_{i=1}^{n}\text{Var}\left(\sum_{j=1}^{p_R} x_{Rij}\beta_{Rj}\right) = \sigma^2_R \times MS_x$, where $MS_x = n^{-1}\sum_{i=1}^{n}\sum_{j=1}^{p_R} x^2_{Rij}$ is the average sum of squares of the genotypes. We can then equate $\sigma^2_R \times MS_x$ to the product of our prior expectation about trait heritability times (an estimate of) the phenotypic variance, to get $\sigma^2_R = h^2 V_y / MS_x$. Equating this to the prior expectation of the variance parameter and solving for the scale yields

$$S_R = \frac{h^2 V_y (df_R - 2)}{MS_x} \tag{4}$$

In Box 4, we use this formula and $df_R = 5$ to define the prior for σ^2_R. The model includes an intercept and usually we measure variance using squared-deviations from the center of the sample; therefore, we can compute MS_x as

$$MS_x = n^{-1}\sum_{i=1}^{n}\sum_{j=1}^{P_R}\left(x_{Rij} - \overline{x}_{Rj}\right)^2, \tag{5}$$

where $\overline{x}_{Rj}$ is mean of the j^{th} marker in the sample.

After fitting the model, we provide plots for squared estimates of marker effects (line 22) and predicted genetic values (lines 24–26). The term `fm$yHat` is the estimated posterior mean of the linear predictor η_i, which in the case of the example in Box 4 is $\eta_i = \mu + \sum_{j=1}^{p_R} x_{Rij}\beta_{Rj}$. Note that relative to observed phenotypes, predictions are shrunk towards zero; in this example, the regression of predictions on phenotypes is roughly 0.45 (see line 27, Box 4).

2.3.2 Bayesian LASSO

The code needed to fit the Bayesian LASSO (BL) is very similar to that in Box 4. The most important modifications are (a) in line 19 of Box 4 we replace XR with XL (this indicates to the program that BL instead of BRR needs to be fitted), and, (b) we need to provide hyper-parameters for the BL.

Setting hyper-parameters in the BL. The hyper-parameters of the BL include the scale and *df* parameters assigned to the residual variance and the regularization parameter λ. Hyper-parameters corresponding to the residual variance can be set using the same principles

Box 4. Fitting a Bayesian Ridge Regression

```
rm(list=ls())
library(BLR)
set.seed(12345)
data(wheat)

# Extract phenotype
 y<-Y[,1]

# Defines the prior
DF<-5
Vy<-var(y)
h2<-0.5
Se<-Vy*(1-h2)*(DF-2)
MSx<-sum(apply(FUN=var,MARGIN=2,X=X))
Sr<-Vy*h2*(DF-2)/MSx
prior=list( varE=list(df=DF,S=Se) , varBR=list(df=DF,S=Sr ) )

# Fits the model
fm<-BLR(y=y,XR=X,
nIter=12000,burnIn=2000,prior=prior,saveAt="box4_")

# Plot of squared-estimated effects
plot(fm$bR^2,col=2,ylab=expression(paste(beta[j]^2)),main="BRR")
# Plot of estimated genetic values versus phenotypes
plot( fm$yHat~y,col=2,ylab="Genomic Value",
       xlab="Phenotype", ylim=c(-3,3),xlim=c(-3,3))
abline(b=1,a=0,lty=2) ; abline(h=0,lty=2,v=0)
reg<-lm(fm$yHat~y)$coef ; abline(a=reg[1],b=reg[2],col=4);
print(reg)
```

outlined in the previous example (*see* Box 4). For λ BLR has three options: (a) assign a **gamma prior** to λ^2 (*see* ref. 1 and information provider below for further details), (b) assign a **Beta prior** to the ratio of $\tilde{\lambda} = \lambda/\max$ where max is a user prespecified upper-bound to λ (*see* ref. 2 for further details about this approach), or (c) fit the model with λ **fixed** at a value specified by the user (see below).

The list that specifies the prior for λ has the following arguments: `$value` (numeric), which is the initial value that will be used by the sampler, and `$type` (character), which informs the algorithm whether λ should remain fixed at the initial value (`$type="fixed"`) or updated from its fully conditional density (`$type="random"`), `$rate` and `$shape`, in case λ^2 is assigned a gamma prior, or `$max`, `$shape1` and `$shape2`, in case a beta prior is assigned to $\tilde{\lambda} = \lambda/\max$. Here we focus on the case where λ^2 is assigned a gamma prior. In this case, we need to specify the rate (r) and shape (s) parameters of the gamma density assigned to λ^2. To do this, one possibility is to first find a "focal point" by computing a "guess" about λ^2 as a function of trait heritability and marker information. For instance, using $h^2 V_y = \mathrm{Var}(\beta_{jL}|\lambda^2) \times MS_x$ and noting that in the BL $\mathrm{Var}(\beta_{jL}|\lambda^2) = 2(\sigma_\varepsilon^2/\lambda^2)$, we obtain $h^2 V_y = 2(\sigma_\varepsilon^2/\lambda^2)MS_x$, and solving for λ^2, we get:

$$\lambda^2 = 2\frac{(1-h^2)}{h^2}MS_x \tag{6}$$

Box 5. Displaying the Density of λ^2 in the BL

```
rm(list=ls())
library(BLR)
data(wheat)

# computes MSx and noise-signal ratios
MSx<-0
for(i in 1:ncol(X)){ MSx<-MSx+mean((X[,i]-mean(X[,i]))^2) }
h2<-c(0.3,0.5,0.7)
noiseToSignal<-(1-h2)/h2
lambda<-1:50

# hyper-parameters
shape<-1.01
rate<-(shape-1)/(2*noiseToSignal[2]*MSx)

 # Density Plot
 plot( y=dgamma(lambda^2,rate=rate,shape=shape),
       x=(lambda^2),col=2,type="l",
       ylab="Density",xlab=expression(paste(lambda^2)))
 abline(v= (2*(1-h2)/h2*MSx),lty=2)
```

We can then use Eq. 6 to compute a prior "guess" about λ^2. Subsequently we choose the rate (r) and shape (s) parameters of the gamma density so that this density has a mode and is relatively uninformative in the neighborhood of the value computed using Eq. 6. If $\lambda^2 \sim G(\lambda^2|r,s)$, the prior mode is $r^{-1}(s-1)$ (for $s>1$) and the prior coefficient of variation is $s^{-1/2}$. Our recommendation is to choose s so it is slightly greater than 1 (e.g., $s=1.01$); this guarantees the existence of the prior mode and gives a large coefficient of variation (i.e., a relatively uninformative prior). Then, one can solve for the rate parameter so that the prior mode matches the value computed using Eq. 6, that is $r=(s-1)(h^2/2(1-h^2)MS_x)$. An example of this is given in Box 5.

We are now ready to fit the BL. The code is given in Box 6.

2.4 Fitting Models for Molecular Markers and Pedigree

We now extend the marker-based regression in Box 6 by adding a regression on the pedigree. BLR has an argument GF (for "grouping factor") that allows fitting a random effect (see u_i in Eq. 1). This argument is specified using a two-components list. The first component of the list, `GF$ID`, is an integer vector of dimension $n \times 1$, and each entry gives the corresponding level (if there are q-levels, `GF$ID` must take values from 1 to q) of the random effect. The second entry of the list, `GF$A`, is a numeric (symmetric, positive definite) matrix of dimensions $q \times q$ providing a (co)variance structure between levels of the random effect (*see* **Note 6** for examples of how to specify GF with and without repeated measures). An example of a model including markers and pedigree is given in Box 7.

2.4.1 Evaluation of Goodness of Fit, Model Complexity, and Deviance Information Criterion

The posterior means of the residual variances of the models fitted in Boxes 6 and 7 are obtained by typing `fmM$varE` and `fmMP$varE`; these are 0.55 and 0.46, respectively, indicating that adding the pedigree to the model increased the goodness-of-fit of the training dataset. Adding the pedigree can also affect the estimates of marker effects; in this case, when we add the pedigree, the posterior mean of λ increases only slightly (from 19.7, `fmM$lambda` to 20.7, `fmMP $lambda`), suggesting a slight increase in shrinkage of estimates of marker effects. Moreover, the correlations between estimates of marker effects and predictions derived from the model with and without the pedigree, `cor(fmM$bL,fmMP$bL)` and `cor(fmM $yHat,fmMP$yHat)`, were both very high, on the order of 0.97.

Further information on model goodness-of-fit and complexity, Deviance Information Criterion (DIC) and effective number of parameters (pD) [11], is given in the `$fit` entry of the fitted models: adding the pedigree to the model (`fmMP` vs. `fmM`) increased the estimated number of effective parameters by about 55 units (`fmMP$fit$pD-fmM$fit$pD`) and decreased DIC by roughly 50 units. Therefore, DIC ("smaller is better") favors the model that includes marker and pedigree information relative to the one using markers only.

2.5 Evaluation of Predictive Performance

Evaluating models based on predictive performance has become a common practice in genomic selection. Next we describe alternative ways of estimating measures of prediction accuracy.

2.5.1 Training–Testing Evaluation

The training–testing (TRN–TST) validation scheme is the simplest approach we can use; here we partition the data into two disjoint sets (TRN and TST), fit the model to the TRN dataset and evaluate predictive performance in the TST dataset. Therefore, when fitting the model, we need to "mask" all the data in the TST dataset to BLR. We can do this in two different ways: one possibility is to subset the data and give BLR only the data in the TRN dataset. The second possibility is to replace the observed phenotypes in the TST dataset with missing values. Both ways are illustrated in the code of Box 8 for a marker-regression model using the BL.

2.5.2 Replicated TRN–TST Evaluations

The code in Box 9 illustrates how we can replicate the TRN–TST evaluation in Box 8 by randomly generating multiple partitions. The code is implemented using a loop; however, as we discuss later on in this chapter, these partitions could be run in parallel. The number of replicates is defined in line 11 of Box 9. Using `nReplicates<-100` we obtained an average correlation of 0.5 with a 90 % CI given by [0.43, 0.58]. The variability in measures of prediction accuracy observed across replicates reflects uncertainty about estimates of prediction accuracy due to sampling of TRN and TST datasets.

Box 6. Fitting the Bayesian LASSO to genome-wide markers

```
rm(list=ls()); library(BLR); set.seed(12345); data(wheat)
# Extract phenotype
 y<-Y[,1]

# Defines the prior
DF<-5
Vy<-var(y)
h2<-0.5
Se<-Vy*(1-h2)*(DF-2)
MSx<-sum(apply(FUN=var,MARGIN=2,X=X))
lambda<-sqrt(2*(1-h2)/h2*MSx)
shape<-1.01
rate<-(shape-1)/(lambda^2)

prior=list(  varE=list(df=DF,S=Se) ,
             lambda=list( type="random",value=lambda,
                          shape=shape,rate=rate
                        )
          )

# Fits the model
fmM<-BLR(y=y,XL=X,
   nIter=12000,burnIn=2000,prior=prior,saveAt="box6_")
str(fmM)
# (continues in Box 7)
```

Box 7. Fitting the Bayesian LASSO to genome-wide markers and pedigree

```
#(continued from Box 6)

# Defines the prior
DF<-5
Vy<-var(y)
h2<-0.5
Se<-Vy*(1-h2)*(DF-2)
Su<-Vy*h2/3
MSx<-sum(apply(FUN=var,MARGIN=2,X=X))
lambda<-sqrt(2*(1-(h2*2/3))/(h2*2/3)*MSx)
shape<-1.01
rate<-(shape-1)/(lambda^2)

prior=list(  varE=list(df=DF,S=Se) ,
             lambda=list( type="random",value=lambda,
                          shape=shape,rate=rate
                        ),
             varU=list(df=DF,S=Su)
          )

# Fits the model
fmMP<-BLR(y=y,XL=X,GF=list(ID=1:599,A=A),
        nIter=12000,burnIn=2000,prior=prior,saveAt="box7_")
str(fmMP)
```

Box 8. Evaluating predictive performance in training-testing partitions

```
rm(list=ls()); library(BLR); set.seed(12345); data(wheat)

# Extracts phenotypes
 y<-Y[,1]

#  Prior  (see Boxes 5 and 6 for details)

prior=list(  varE=list(df=5,S=1.5) ,
             lambda=list(type="random",value=20,
                         rate=2e-4,shape=2)
          )

# Random assignment of data into training and testing
tst<-sample(x=1:599,size=150,replace=FALSE)

## Method 1: Introduce NAs in TST
yNA<-y ; yNA[tst]<-NA
fm1<-BLR(y=yNA,XL=X,
  nIter=12000,burnIn=2000,prior=prior,saveAt="box8_1_")


## Method 2: Subsets the data
XTrn<-X[-tst,] ; yTrn<-y[-tst]
XTst<-X[tst,] ; yTst<-y[tst]
fm2<-BLR( y=yTrn,XL=XTrn,nIter=12000,
          burnIn=2000,prior=prior,saveAt="box8_2_")
yHatTst<-fm2$mu+XTst%*%fm2$bL
plot(yHatTst~fm1$yHat[tst],col=2,
     main="Predictions in TST set by method",
     xlab="Introducting Missing Values",
     ylab="Subseting the data")
```

Box 9. Replicated training-testing partitions

```
rm(list=ls()); library(BLR); set.seed(12345);data(wheat)
# Extracts phenotypes
 y<-Y[,1]

#  Prior (see Boxes 5 and 6 for details).

prior=list(  varE=list(df=5,S=1.5) ,
             lambda=list(type="random",value=20,
                    rate=2e-4,shape=2)
          )

trnTstCor<-numeric()
nReplicates<-10

for(i in 1:nReplicates){
    tst<-sample(x=1:599,size=150,replace=FALSE)
    yNA<-y ; yNA[tst]<-NA
    fm<-BLR(y=yNA,XL=X,nIter=6000,burnIn=1000,prior=prior,
            saveAt= paste("box10_rep_",i,"_",sep=""))
    trnTstCor[i]<-cor(y[tst],fm$yHat[tst])
}
summary(trnTstCor)
```

Box 10. Example of a 5-fold cross-validation

```
rm(list=ls()); library(BLR); set.seed(12345); data(wheat)

# Extracts phenotypes
 y<-Y[,1]

#  Prior  (see Boxes 5 and 6 for details)

prior=list(  varE=list(df=5,S=1.5) ,
             lambda=list(type="random",value=20,
                         rate=2e-4,shape=2)
          )

cvCor<-numeric()
folds<-rep(1:5,120)[order(runif(599))]

for(i in 1:5){
    tst<-which(folds==i)
    yNA<-y ; yNA[tst]<-NA
    fm<-BLR(y=yNA,XL=X,nIter=6000,burnIn=1000,
            prior=prior,saveAt= paste("box10_",i,"_",sep=""))
    cvCor[i]<-cor(y[tst],fm$yHat[tst])
}
cvCor
```

2.5.3 Cross-Validation

Cross-validation (CV) is another commonly used method for estimating prediction accuracy. In a CV we assign data points into disjoint sets (e.g., 5 for a fivefold cross-validation, CV). Subsequently, for each of these folds we conduct a TRN–TST evaluation where all individuals assigned to the fold are used for TST and the rest are used for TRN. An example of how to implement this sequentially is given in Box 10.

2.6 Some Useful Commands for Executing R in a Linux Environment

Some of the tasks we have discussed in the previous section can be performed in parallel. For instance, if we have access to several processors or nodes, we could compute each of the TRN–TST evaluations in Box 9 or the CV in Box 10 in parallel. In high-performance clusters, we may do this by creating arrays of jobs which are simultaneously sent to several nodes. In performing these tasks, it is usually convenient to provide some arguments to R using the command line. To do this, we first need to include the script we want to execute (e.g., the code in Box 10) in a text file (let's assume we have named this file `exBox10.R`); then we can execute these instructions from the command line by typing:

```
R CMD BATCH exBox10.R  out &
```

In the above command:

1. `R CMD BATCH` tells the system that we want to run an R command in batch mode.

2. `exBox10.R` is the name of the file with the R-script printed in Box 10 included.
3. `out` is a log file where the system will print all the outputs that are printed to the R-console when the script is executed.
4. `&` sends the job to the background and frees the terminal. In this way, you can continue to work in the command line while the job is executed.

For parallel computing, it is sometimes useful to set values to R objects from the command line. For instance, in a tenfold CV, it would be useful to execute in parallel the analyses required for each fold. This could be done if one of the variables in the R environment (e.g., fold number) is defined at the command line. In order to be able to set variables using arguments provided from the command line, first, we need to include, at the beginning of the R-script, the following instructions (see `help(commandArgs)` in the R-console for further information about the `commandArgs` function)

```
args=(commandArgs(TRUE))
for(i in 1:length(args)){
        eval(parse(text=args[[i]]))
}
```

These instructions allow reading arguments from the command line and set those arguments as variables in the R-session. Subsequently, we execute the script and pass arguments to R using the following command:

```
R CMD BATCH "--args fold=1" exBox11.R out &
```

When we execute the above command, the variable `fold` is set equal to 1 within the R-session. Box 11 gives a modified version of the R-code of Box 10, which could be run in parallel by executing R from the command line. Lines 2–3 read arguments from the command line. Lines 4–14 are exactly as in Box 10. In the remaining lines, instead of running a loop over folds, we just fit a model for onefold (the one specified in the command line). The code in line 16 creates the TST set for that particular fold. Since we are running this in BATCH mode, before quitting R we save the model. In line

Box 11. R-script for running 1 fold of a CV (fold specified in the command line)

```
rm(list=ls())
args=(commandArgs(TRUE))
for(i in 1:length(args)){  eval(parse(text=args[[i]])) }
 library(BLR); set.seed(12345);data(wheat)
y<-Y[,1]

if(is.null(fold)){ stop("Fold was not provided") }

#  Prior  (see Boxes 5 and 6 for details)

prior=list(  varE=list(df=5,S=1.5) ,
             lambda=list(type="random",value=20,
                         rate=2e-4,shape=2)
          )

cvCor<-numeric()
folds<-rep(1:5,120)[order(runif(599))]

tst<-which(folds==fold)
yNA<-y ; yNA[tst]<-NA
fm<-BLR(y=yNA,XL=X,nIter=6000,burnIn=1000,
        prior=prior,saveAt=paste("fold_",fold,"_",sep="")
       )
save(fm,file= paste("fm_fold_",fold, ".rda",sep="") )
```

24 we give instructions to R to save the fitted model in a compressed file format (`*.rda`). All outputs are saved with the fold number indicated in the filenames.

In the previous example we show how to fit models to the first fold. We can ran all folds in parallel using the following bash-code

```
R CMD BATCH "--args fold=1" exBox11.R out1 &
R CMD BATCH "--args fold=2" exBox11.R out2 &
R CMD BATCH "--args fold=3" exBox11.R out3 &
R CMD BATCH "--args fold=4" exBox11.R out4 &
R CMD BATCH "--args fold=5" exBox11.R out5 &
```

Or, even simpler, using a do-loop in bash

```
#!/bin/bash
for (( fold=1; fold<=5; fold++ ))
 do
    R CMD BATCH "--args fold=$fold " exBox11.R out$fold &
done
```

3 Discussion

We developed the first version of the Gibbs sampler in BLR in 2009 [2] and in 2010 we offered the program as an R-package [3]. In the R-package, a few steps of the Gibbs sampler were programmed in C, and this greatly increased the computational speed. We have used BLR with up to 100 thousand SNPs and close to 10 thousand records. In principle there are no limits to the number of markers or records beyond those imposed by the availability of RAM memory and processing time.

Memory requirements and run-time complexity. The incidence matrices for fixed and random effects (XF, XR, and XL) are all stored as numeric. This allows fast computations and the possibility of using the program with discrete (e.g., SNP codes) or continuous predictors. This generality comes at the price of higher memory requirements, and the user should be aware that memory requirements increase proportionally to the number of subjects and number of markers. For high density markers (e.g., hundreds of thousands of SNPs), memory requirements can be high.

In the Gibbs sampler, updates are done one effect at a time; therefore, computational time usually increases proportionally to the number of markers (p). Figure 1 displays the time in seconds that it takes for BLR to run (in an Intel Core i7 with a 2.4Gz processor) 1,000 iterations for various values of p and n. With

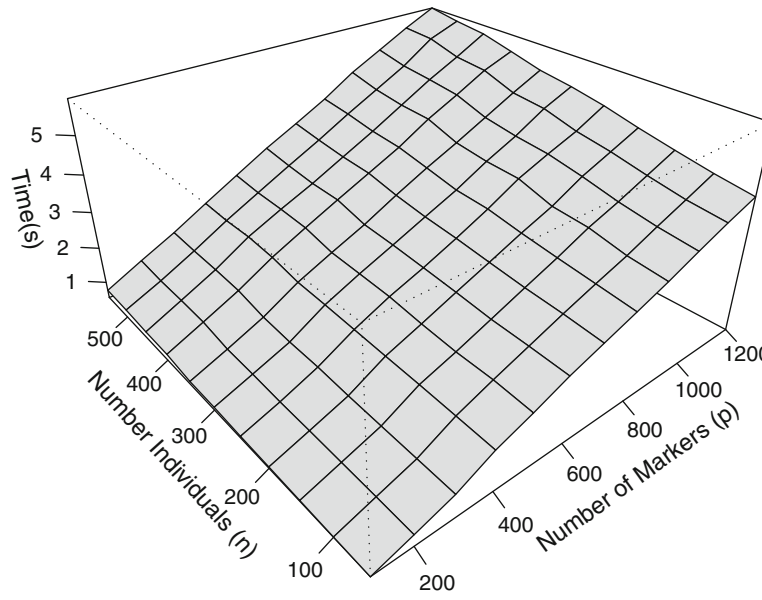

Fig. 1 Running time by sample size and number of markers (1,000 iterations run on an Intel Core i7 with a 2.4 Gz processor)

1,279 markers and 599 individuals it took approximately 6 seconds to run 1,000 iterations of the Gibbs sampler.

Influence of hyper-parameters on inferences. When $p \gg n$ marker effects cannot be uniquely estimated from the likelihood and under these conditions the influences of the prior on inferences are nontrivial. This applies to any Bayesian regression model where $p \gg n$, including the BRR and the BL. We characterized the influences of the prior on inferences about marker effects in the BL, and the results were published in de los Campos et al. [2]. In general, predictions of genetic values are quite robust within reasonable values of λ; however, estimates of marker effects are more sensitive to this parameter. Our recommendation is to first compute the value of λ that we expect using Eq. 6 and then choose values of the rate and shape parameters so that the prior has a mode that it is reasonably flat in the neighborhood of the value derived from Eq. 6.

In some instances, when p is very large and n is too small, the mixing of the MCMC algorithm may be too poor. In these cases, the only alternative is to fix λ, using `prior$lambda$type="fixed"`, and then choose the value of λ either using Eq. 6 or some type of cross-validation. Again, our experience dictates that predictions are relatively robust with respect to λ, within reasonable ranges in the neighborhood of the value given by Eq. 6. Importantly, although Bayesian methods provide a way of dealing with the "*large-p with small-n*" problem, n cannot be too small; otherwise we are at the mercy of the prior. Establishing a minimum number of records is difficult, because this will depend on factors such as number of markers, heritability and effective population size. However, the consequence of a too-small n should be evident from inspection of trace plots of the residual variance or regularization parameters.

Future developments. Over the past two years we have identified a series of limitations in the program, which we plan to address in future releases. These include:

(a) The available R-version can only handle continuous outcomes; we have already developed a beta version of BLR that handles binary and censored outcomes. This version is available upon request and we plan to incorporate this possibility in future releases.

(b) The design of the user interface and the internal structure of the program impose important restrictions. The fact than only one `XL`, `XR`, or `GF` is allowed, implies that we cannot estimate different regularization or variance parameters for different sets of predictors. We have developed a proposal to modify the program using an interface that will allow including varying numbers of design matrices for effects, each of which will be

assigned different priors. This will give the user the possibility of generating a diverse array of models. We plan to incorporate this in future releases.

(c) Another line of development that we plan to pursue is the incorporation of alternatives for priors that induce variable selection such as those of models BayesB [4] or BayesC [12].

Releases from this project will be made available first at the R-forge site http://bglr.r-forge.r-project.org/ and, after a testing period, at the R repository.

4 Notes

1. *Use of the R-package pedigreemm for building a numerator relationship matrix and related objects.* We illustrate how to compute additive relationship matrices and functions of it using the pedigreemm package of R ([9] http://cran.r-project.org/web/packages/pedigreemm/index.html). The example uses a simple pedigree from [13].

```
##  Example 1
############################################################
 rm(list=ls())
 library(pedigreemm)

 # defines the pedigree
   id<-1:6
   sire<-c(NA,NA,1, 1,4,5)
   dam<-c(NA,NA,2,NA,3,2)

 # Creating a pedigree object
   ped <- pedigree(label=id,sire=sire,dam=dam)

 # Computes inbreeding coefficients
   F<-inbreeding(ped)

 # Computes the additive relationship matrix (A)
   A<- getA(ped)

 # The cholesky factor of A (upper-triangular)
   U<-relfactor(ped)

 # T-inverse and D-inverse from A=TDT'
    TInv <- as(ped, "sparseMatrix")
    DInv <- diag(1/Dmat(ped))

 # Computes the inverse of the additive relationship matrix
   AInv<-getAInv(ped)
```

Note the pedigree must be sorted so that ancestors appear before offspring and complete (i.e., any animal that appear as sire or dam must also appear in the id vector. The function `editPed()` can be used to complete and sort a pedigree before the pedigree object is created (see example below).

```
##  Example 2
#########################################################
 rm(list=ls()); library(pedigreemm)
# Pedigree complete and sorted (From Example 1)
   id<-1:6;
   sire<-c(NA,NA,1,1,4,5);
   dam<-c(NA,NA,2,NA,3,2)
# Now we remove sire 1 and dam 2
# and shuffle the remaining pedigree
  PED<-cbind(id,sire,dam)
  PED<-PED[-c(1,2),]      # removes 1 sire and 1 dam
  PED<-PED[4:1,]          # shuffles pedigree
# Try to build the pedigree object yields an error
  ped<- pedigree(label=PED[,1],sire=PED[,2],dam=PED[,3])

# Now we use the editPed() function to fix the problems
  PED<-editPed(label= PED[,1] ,sire=PED[,2], dam=PED[,3])
  ped<- pedigree(label=PED[,1],sire=PED[,2],dam=PED[,3])
  image(getA(ped))
```

2. *Parameterization of scaled-inverse chi-square densities.* In BLR scaled inverse chi-square densities assigned to variance parameters $(\sigma^2_\cdot)$ are parameterized as follows $p(\sigma^2_\cdot|df_\cdot, S_\cdot) \propto \left[\sigma^2_\cdot\right]^{-(1+(df_\cdot/2))} \times \text{Exp}\{-S_\cdot/2\sigma^2_\cdot\}$, with this parameterization the prior expectation and mode of $\sigma^2_\cdot$ are $S_\cdot/(df_\cdot - 2)$ (for $df_\cdot > 2$) and $S_\cdot/(df_\cdot + 2)$, respectively.
3. *Gibbs Sampler.* In a Gibbs sampler [14, 15] draws from a joint density are obtained by sampling from fully conditional densities. To illustrate, let $p(\theta_1, \theta_2|\mathbf{y})$ be the joint posterior density of $\{\theta_1, \theta_2\}$, and let $p(\theta_1|\mathbf{y}, \theta_2)$ and $p(\theta_2|\mathbf{y}, \theta_1)$ denote the fully conditional densities of θ_1 and of θ_2, respectively. Using these densities one could form a Gibbs sampler by sequentially repeating the following steps:

   – Draw one sample of θ_1 from $p(\theta_1|\mathbf{y}, \theta_2)$, denote it $\theta_1^{[t]}$.
   – Set $\theta_1 = \theta_1^{[t]}$.
   – Draw one sample of θ_2 from $p(\theta_2|\mathbf{y}, \theta_1)$, denote it $\theta_2^{[t]}$.
   – Set $\theta_2 = \theta_2^{[t]}$.

   Repeat the above steps B times. Pairs $\{\theta_1^{[t]}, \theta_2^{[t]}\}$ drawn after convergence to the posterior density can be regarded as draws from the joint posterior density.
4. *Storage of samples collected by BLR.* As BLR runs, samples of fixed effects, variance and regularization parameters are stored in text files. By default, samples are saved to the working directory. However, one could use the `saveAt` argument to provide a path and prefix to be appended to filenames. For example, if one sets `saveAt="c:/myExamples/example1_"`, the samples will be saved at `c:/myExamples` and the prefix `example1_` will be added to every file that is created

by BLR. Importantly, if several runs are performed in the same folder and with the same `saveAt` value, samples from different runs will be stacked in the same output files. To avoid this, the user can change the path/prefix provided at saveAt (see line 6, Box 2, for an example) or remove files from previous analysis.

5. *Incidence matrices for effects.* In the example in Box 2, we use the argument XF to provide the incidence matrix for fixed effects. These should be numeric matrices, objects of other classes (e.g., data.frame) will induce errors. BLR does not do a comprehensive check of all inputs; therefore, we always recommend checking that the object type is as required (see help (BLR) for a complete description of input-types). For class-variables (e.g., sex, year-effects), the user needs to create the appropriate contrasts (dummy variables) in advance. The `model.matrix()` function of R can be used to create incidence matrices for class and continuous variables. If the incidence matrices have column-names, these are appended to the estimated posterior means (type `names(fm$bF)` in the example in Box 2 to check this).
6. We explain how to set `ID` and `A` in the `GF` argument for cases with and without repeated measures. Suppose we have a random effect with three levels, and assume that the A matrix has the following form:

	ID1	ID2	ID3
ID1	1.1	0.5	0.0
ID2	0.5	1.1	0.0
ID3	0.0	0.0	1.0

If we have only one record by level of the random effect (i.e., in absence of repeated measures), we sort phenotypes according to the ID variable and use the following code to set the GF argument:

```
A<-rbind(c(1.1,0.5,0),c(0.5,1.1,0),c(0,0,1))
GF<-list(  ID=1:3, A=A )
```

Now, suppose that we have repeated measures. Assume, for instance that there are five phenotypic records and that the phenotypic vector (y) is sorted in a way that the first and third entry of it are associated to the first level of the random effect (`ID[1]=ID[3]=1`), the second entry of y is associated to the third level of the random effect (`ID[2]=3`) and the fourth and

fifth entries are associated to the second level of the random effect (`ID[4]=ID[5]=2`). In this case, we set the GF argument as follows:

```
A<-rbind(c(1.1,0.5,0),c(0.5,1.1,0),c(0,0,1))
GF<-list(  ID=c(1,3,1,2,2), A=A )
```

Acknowledgments

de los Campos, Pérez, and Crossa acknowledge financial support from the International Maize and Wheat Improvement Center (CIMMYT). Pérez and de los Campos were also supported by NIH Grants R01GM101219-01 and R01GM099992-01A1.

References

1. Park T, Casella G (2008) The Bayesian lasso. J Am Stat Assoc 103(482):681–686
2. de los Campos G, Naya H, Gianola D, Crossa J, Legarra A, Manfredi E, Weigel K, Cotes JM (2009) Predicting quantitative traits with regression models for dense molecular markers and pedigree. Genetics 182(1):375–385
3. Pérez P, de los Campos G, Crossa J, Gianola D (2010) Genomic-enabled prediction based on molecular markers and pedigree using the Bayesian linear regression package in R. Plant Genome J 3(2):106–116
4. Meuwissen TH, Hayes BJ, Goddard ME (2001) Prediction of total genetic value using genome-wide dense marker maps. Genetics 157(4):1819–1829
5. de los Campos G, Hickey JM, Detwyler HD, Pong-Wong R, Calus MPL (2013) Whole genome regression and prediction methods applied to plant and animal breeding. Genetics 193:327–345
6. R Development Core Team (2010) R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria [Internet]. R Foundation for Statistical Computing. http://www.R-project.org
7. Hoerl AE, Kennard RW (1970) Ridge regression: biased estimation for nonorthogonal problems. Technometrics 12(1):55–67
8. Vazquez AI, Bates DM, Rosa GJM, Gianola D, Weigel KA (2010) Technical note: an R package for fitting generalized linear mixed models in animal breeding1. J Anim Sci 88 (2):497–504
9. Bates D, Vazquez AI (2009) Pedigreemm: pedigree-based mixed-effects models V 0.2-4 [Internet]. http://cran.r-project.org/web/packages/pedigreemm/index.html
10. McLaren CG, Bruskiewich RM, Portugal AM, Cosico AB (2005) The international rice information system. A platform for meta-analysis of rice crop data. Plant Physiol 139(2):637–642
11. Spiegelhalter DJ, Best NG, Carlin BP, van der Linde A (2002) Bayesian measures of model complexity and fit. J R Stat Soc Series B (Stat Methodol) 64(4):583–639
12. Habier D, Fernando R, Kizilkaya K, Garrick D (2011) Extension of the Bayesian alphabet for genomic selection. BMC Bioinforma 12(1):186
13. Mrode RA, Thompson R (2005) Linear models for the prediction of animal breeding values [Internet]. Cabi. [cited 15 Aug 2012] http://books.google.com/books?hl=en&lr=&id=bnewaF4Uq2wC&oi=fnd&pg=PR5&dq=mrode+animal+breeding+genetics&ots=05EITYRwMh&sig=aDDnpg69HSI4acAmi38yTAVMjqg
14. Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans Pattern Anal Mach Intell 6:721–741
15. Casella G, George EI (1992) Explaining the Gibbs sampler. Am Stat 46:167–174