Bayesian model selection and averaging with mombf

David Rossell

The `mombf` package implements Bayesian model selection (BMS) and model averaging (BMA) for linear, asymmetric linear, median and quantile regression. This is the main package implementing the family of *non-local prior* (NLP) distributions (briefly reviewed here, see Johnson and Rossell (2010, 2012) for a more detailed treatment), although other priors (mainly Zellner's) are also implemented. The main features are:

- Density, cumulative density, quantiles and random numbers for NLPs
- BMS in linear regression (Section 1, Johnson and Rossell (2010, 2012)).
- BMA in linear regression (Section 4, Rossell and Telesca (2016)).
- Exact BMS and BMA under orthogonal and block-diagonal regression (Section 5, Papaspiliopoulos and Rossell (2016)).
- BMS and BMA for certain generalized linear models (Section 6, Johnson and Rossell (2012); Rossell et al. (2013))
- BMS in linear regression with non-normal residuals (Rossell and Rubio, 2016). Particular cases are Bayesian versions of asymmetric least squares, median and quantile regression.

This manual introduces some basic notions underlying NLPs and illustrates the use of R functions implementing the main operations required for model selection and averaging. Most of these are internally implemented in C++ so, while they are not optimal in any sense they are designed to be minimally scalable to high dimensions (large p).

1 Basics on non-local priors

The basic motivation for NLPs is what one may denominate the *model separation principle*. The idea is quite simple, suppose we are considering a (possibly infinite) set of probability models $M_1, M_2, \ldots$ for an observed dataset y, if these models overlap then it becomes hard to tell which of them generated y. The notion is important because the vast majority of applications consider nested models: if say M_1 is nested within M_2 then these two models are not well-separated. Intuitively, if y are truly generated from M_1 then M_1 will receive high integrated likelihood however that for M_2 will also be relatively large given that M_1 is contained in M_2. We remark that the notion remains valid when none of the posed models are true, in that case M_1 is the model of smallest dimension minimizing Kullback-Leibler divergence to the data-generating distribution of y. A common mantra is that performing Bayesian model selection via posterior model probabilities (equivalently, Bayes factors) automatically incorporate Occam's razor, e.g. M_1 will eventually be favoured over M_2 as the sample size $n \to \infty$. This statement is correct but can be misleading: there is no guarantee that the extent to which parsimony is enforced is adequate, indeed it turns out to be insufficient in many practical situations even for small p. This issue is exacerbated for large p to the extent that one may even loose consistency of posterior model probabilities (Johnson and Rossell, 2012) unless sufficiently strong sparsity penalities are introduced into the model space prior.

Intuitively, NLPs induce a probabilistic separation between the considered models which, aside from being philosophically appealing (to us), one can show mathematically leads to stronger parsimony. When we compare two nested models and the smaller one is true the resulting BFs converge faster to 0 than when using conventional priors and, when the larger model is true, they present the usual exponential convergence rates in standard procedures. That is, the extra parsimony induced by NLPs is data-dependent, as opposed to inducing sparsity by formulating sparse model prior probabilities or increasingly vague prior distributions on model-specific parameters.

To fix ideas we first give the general definition of NLPs and then proceed to show some examples. Let $y \in \mathcal{Y}$ be the observed data with density $p(y \mid \theta)$ where $\theta \in \Theta$ is the (possibly infinite-dimensional) parameter. Suppose we are interested in comparing a series of models $M_1, M_2, \ldots$ with corresponding parameter spaces $\Theta_k \subseteq \Theta$ such that $\Theta_j \cap \Theta_k$ have zero Lebesgue measure for $j \neq k$ and, for precision, there exists an l such that $\Theta_l = \Theta_j \cap \Theta_k$ so that the whole parameter space is covered.

Definition 1 *A prior density $p(\theta \mid M_k)$ is a non-local prior under M_k iff*

$\lim p(\theta \mid M_k) = 0$ *as* $\theta \to \theta_0$ *for any* $\theta_0 \in \Theta_j \subset \Theta_k$.

In words, $p(\theta \mid M_k)$ vanishes as θ approaches any value that would be consistent with a submodel M_j. Any prior not satisfying Definition 1 is a *local prior* (LP). As a canonical example, suppose that $y = (y_1, \ldots, y_n)$ with independent $y_i \sim N(\theta, \phi)$ and known ϕ, and that we entertain the two following models:

$$M_1 : \theta = 0$$
$$M_2 : \theta \neq 0$$

Under M_1 all parameter values are fully specified, the question is thus reduced to setting $p(\theta \mid M_2)$. Ideally this prior should reflect one's knowledge or beliefs about likely values of θ, conditionally on the fact that $\theta \neq 0$. The left panel in Figure 1 shows two LPs, specifically the unit information prior $\theta \sim N(0, 1)$ and a heavy-tailed alternative $\theta \sim \text{Cachy}(0, 1)$ as recommended by Jeffreys. These assign $\theta = 0$ as their most likely value a priori, even though $\theta = 0$ is not even a possible value under M_2, which we view as philosophically unappealing. The right panel shows three NLPs (called MOM, eMOM and iMOM, introduced below). Their common defining feature is their vanishing as $\theta \to 0$, thus probabilistically separating M_2 from M_1 or, to borrow terminology from the stochastic processes literature, inducing a repulsive force between M_1 and M_2. As illustrated in the figure beyond this defining feature the user is free to choose any other desired property, e.g. the speed at which $p(\theta \mid M_2)$ vanishes at the origin, prior dispersion, tail thickness or in multivariate cases the prior dependence structure.

Once the NLP has been specified inference proceeds as usual, e.g. posterior model probabilities are

$$p(M_k \mid y) = \frac{p(y \mid M_k) p(M_k)}{\sum_j p(y \mid M_j) p(M_j)} \tag{1}$$

where $p(y \mid M_k) = \int p(y \mid \theta) dP(\theta \mid M_k)$ is the integrated likelihood under M_k and $p(M_k)$ the prior model probability. Similarly, inference on parameters can be carried out conditional on any chosen model via $p(\theta \mid M_k, y) \propto p(y \mid \theta) p(\theta \mid M_k)$ or via Bayesian model averaging $p(\theta \mid y) = \sum_k p(\theta \mid M_k, y) p(M_k \mid y)$. A useful construction (Rossell and Telesca, 2016) is that any NLP can be expressed as

$$p(\theta \mid M_k) = \frac{p(\theta \mid M_k)}{p^L(\theta \mid M_k)} p^L(\theta \mid M_k) = d_k(\theta) p^L(\theta \mid M_k), \tag{2}$$

where $p^L(\theta \mid M_k)$ is a LP and $d_k(\theta) = p(\theta \mid M_k)/p^L(\theta \mid M_k)$ a penalty term. Simple algebra shows that

$$p(y \mid M_k) = p^L(y \mid M_k) E^L(d_k(\theta) \mid M_k, y), \tag{3}$$

where $E^L(d_k(\theta) \mid M_k, y) = \int d_k(\theta) dP^L(\theta \mid M_k, y)$ is the posterior mean of the penalty term under the underlying LP. That is, the integrated likelihood under a NLP is equal to that under a LP times the posterior expected penalty under that LP. The construction allows to use NLPs in any situation where LPs are already implemented, all one needs is $p^L(y \mid M_k)$ or an estimate thereof and posterior samples under $p^L(\theta \mid y, M_k)$. We remark that most functions in mombf do not rely on construction (2) but instead work directly with NLPs, as this is typically more efficient computationally. For instance, there are closed-form expressions and Laplace approximations to $p(y \mid M_k)$ (Johnson and Rossell, 2012), and one may sample from $p(\theta \mid M_k, y)$ via simple latent truncation representations (Rossell and Telesca, 2016).

Up to this point we kept the discussion as generic as possible, in the next section we proceed to illustrate the use of NLPs for variable selection. For extensions to other settings see for instance Consonni and La Rocca (2010) for directed acyclic graphs under an objective Bayes framework, Chekouo et al. (2015) for gene regulatory networks, Collazo et al. (2016) for chain event graphs, or Fúquene et al. (2016) for finite mixture models. We also remark that this manual focuses mainly on practical aspects. Readers interested in theoretical NLP properties should see Johnson and Rossell (2010) and Rossell and Telesca (2016) for an asymptotic characterization under asymptotically Normal models with fixed $\dim(\Theta)$, essentially showing that $E^L(d_k(\theta) \mid M_k, y)$ leads to stronger parsimony, Fúquene et al. (2016) for similar results in mixture models, and Rossell and Rubio (work in progress) for robust linear regression with non-normal residuals where data may be generated by a model other than those under consideration (M-complete). Regarding high-dimensional results Johnson and Rossell (2012) prove that under certain linear regression models $p(M_t \mid y) \xrightarrow{P} 1$ as $n \rightarrow \infty$ where M_t is the data-generating truth when using NLPs and $p = O(n^\alpha)$ with $\alpha < 1$. The authors also proved the conceptually stronger result that $p(M_t \mid y) \xrightarrow{P} 0$ under LPs, which implies that NLPs are a necessary condition for strong consistency in high dimensions (unless extra parsimony is induced via $p(M_k)$ or increasingly diffuse $p(\theta \mid M_k)$ as n grows, but this may come at a loss of signal detection power). Shin et al. (2015) extend the consistency result to ultra-high linear regression with $p = O(e^{n^\alpha})$ with $\alpha < 1$ under certain specific NLPs.

2 Some default non-local priors

Definition 1 in principle allows one to define NLPs in any manner that is convenient, as long as $p(\theta \mid M_k)$ vanishes for any value θ_0 that would be consistent with a submodel of M_k. mombf implements some simple priors that lead to convenient implementation and interpretation while offering a reasonable modelling flexibility, but naturally we encourage everyone to come up with more sophisticated alternatives as warranted by their specific problem at hand. It is important to distinguish between two main strategies to define NLPs, namely imposing additive vs. product penalties. Additive NLPs were historically the first to be introduced (Johnson and Rossell, 2010) and primarily aimed to compare only two models, whereas product NLPs were introduced later on (Johnson and Rossell, 2012) for the more general setting where considers multiple models. Throughout let $\theta = (\theta_1, \ldots, \theta_p) \in \mathbb{R}^p$ be a vector of regression coefficients and ϕ a dispersion parameter such as the residual variance in linear regression.

Suppose first that we wish to test $M_1 : \theta = (0, \ldots, 0)$ versus $M_2 : \theta \neq (0, \ldots, 0)$. An additive NLP takes the form $p(\theta \mid M_k) = d(q(\theta))p^L(\theta \mid M_k)$, where $q(\theta) = \theta' V \theta$ for some positive-definite $p \times p$ matrix V, the penalty $d(q(\theta)) = 0$ if and only if $q(\theta) = 0$ and $p^L(\theta \mid M_k)$ is an arbitrary LP with the only restriction that $p(\theta \mid M_k)$ is proper. For instance,

$$\begin{aligned}
p_M(\theta \mid \phi, M_k) &= \frac{\theta' V \theta}{p \tau \phi} N(\theta; 0, \tau \phi V^{-1}) \\
p_E(\theta \mid \phi, M_k) &= c_E e^{-\frac{\tau \phi}{\theta' V \theta}} N(\theta; 0, \tau \phi V^{-1}) \\
p_I(\theta \mid \phi, M_k) &= \frac{\Gamma(p/2)}{|V|^{\frac{1}{2}} (\tau \phi)^{\frac{p}{2}} \Gamma(\nu/2) \pi^{p/2}} (\theta' V \theta)^{-\frac{\nu + p}{2}} e^{-\frac{\tau \phi}{\theta' V \theta}} \qquad (4)
\end{aligned}$$

are the so-called moment (MOM), exponential moment (eMOM) and inverse moment (iMOM) priors, respectively, and c_E is the moment generating function of an inverse chi-square random variable evaluated at -1. By default $V = I$, but naturally other choices are possible.

Suppose now that we wish to consider all 2^p models arising from setting individual elements in θ to 0. Product NLPs are akin to (4) but now the penalty term $d(\theta)$ is a product of univariate penalties.

$$p_M(\theta \mid \phi, M_k) = \prod_{i \in M_k} \frac{\theta_i^2}{\tau\phi_k} N(\theta_i; 0, \tau\phi_k)$$

$$p_E(\theta \mid \phi, M_k) = \prod_{i \in M_k} \exp\left\{\sqrt{2} - \frac{\tau\phi_k}{\theta_i^2}\right\} N(\theta_i; 0, \tau\phi_k),$$

$$p_I(\theta \mid \phi, M_k) = \prod_{i \in M_k} \frac{(\tau\phi_k)^{\frac{1}{2}}}{\sqrt{\pi}\theta_i^2} \exp\left\{-\frac{\tau\phi_k}{\theta_i^2}\right\}. \quad (5)$$

This implies that $d(\theta) \to 0$ whenever any individual $\theta_i \to 0$, in contrast with (4) which requires the whole vector $\theta = 0$. More generally, one can envision settings requiring a combination of additive and product penalties. For instance in regression models for continuous predictors product penalties are generally appropriate, but for categorical predictors one would like to either include or exclude all the corresponding coefficients simultaneously, in this sense additive NLPs resemble group-lasso type penalties and have the nice property of being invariant to the chosen reference category. At this moment mombf primarily implements product NLPs and in some cases additive NLPs, we plan to incorporate combined product and addivite penalties in the future.

Figure 1 displays the prior densities in the univariate case, where (4) and (5) are equivalent. p_M induces a quadratic penalty as $\theta \to 0$, and has the computational advantage that for posterior inference the penalty can often be integrated in closed-form, as it simply requires second order moments. p_E and p_I vanish exponentially fast as $\theta \to 0$, inducing stronger parsimony in the Bayes factors than p_M. This exponential term converges to 1 as $q(\theta)$ increases, thus the eMOM has Normal tails and the iMOM can be easily checked to have tails proportional to those of a Cauchy. Thick tails can be interesting to address the so-called *information paradox*, namely that the posterior probability of the alternative model converges to 1 as the residual sum of squares from regressing y on a series of covariates converges to 0 (Liang et al., 2008; Johnson and Rossell, 2010), although in our experience this is often not an issue unless n is extremely small. The priors above can be extended to include nuisance regression parameters that are common to all models, and also to consider higher powers $q(\theta)^r$ for some $r > 1$, but for simplicity we restrict attention to (4).

We now briefly discuss how to set the prior dispersion τ and propose default values, though we encourage everyone to think which values are more appropriate for the current problem at hand. By default we set τ so that $P(|\theta/\sqrt{\phi}| > 0.2) = 0.99$, that is the signal-to-noise ratio $|\theta_j|/\sqrt{\phi}$ is a priori

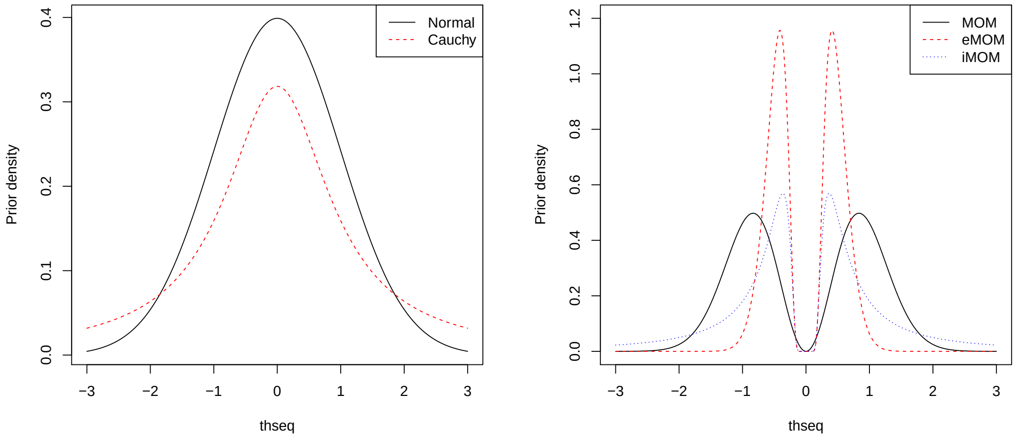

Figure 1: Priors for θ under a model $M_2 : \theta \neq 0$. Left: local priors. Right: non-local priors

expected to be > 0.2. The reason for choosing the 0.2 threshold is that in many applications smaller signals are not practically relevant (e.g. the implied contribution to the R^2 coefficient is small). The function `priorp2g` facilitates finding τ for any given threshold. Other useful functions are `pmom`, `pemom` and `pimom` for distribution functions, and `qmom`, `qemom` and `qimom` for quantiles. Prior densities can be evaluated and plotted as shown below.

```
> library(mombf)
> thseq <- seq(-3,3,length=1000)
> plot(thseq,dnorm(thseq),type='l',ylab='Prior density')
> lines(thseq,dt(thseq,df=1),lty=2,col=2)
> legend('topright',c('Normal','Cauchy'),lty=1:2,col=1:2)
```

```
> thseq <- seq(-3,3,length=1000)
> plot(thseq,dmom(thseq,tau=.348),type='l',ylab='Prior density',ylim=c(0,1.2))
> lines(thseq,demom(thseq,tau=.119),lty=2,col=2)
> lines(thseq,dimom(thseq,tau=.133),lty=3,col=4)
> legend('topright',c('MOM','eMOM','iMOM'),lty=1:3,col=c(1,2,4))
```

Another way to elicit τ is to mimic the Unit Information Prior and set the prior variance to 1, for the MOM prior this leads to the same τ as the earlier rule $P(|\theta/\sqrt{\phi}| > 0.2) = 0.99$.

Another helpful function for prior elicitation of τ is `mode2g`, which returns the τ value associated to a given prior mode $m = \arg\max_{|\theta|} p(|\theta|)$, for instance $m = 0.4$ in our example below. The intuition is that NLPs place most of the prior mass on $|\theta| > m$, which implicitly means we have little interest in detecting $|\theta| < m$.

```
> prior.mode <- .4^2
> taumom <- mode2g(prior.mode,prior='normalMom')
> tauimom <- mode2g(prior.mode,prior='iMom')
> taumom

[1] 0.08

> tauimom

[1] 0.16
```

3 Variable selection for linear models

The main function for model selection is `modelSelection`, which returns model posterior probabilities under linear regression models allowing for Normal, asymmetric Normal, Laplace and asymmetric Laplace residuals. A second interesting function is `nlpMarginal`, which computes the integrated likelihood for a given model under the same settings. We illustrate their use with a simple simulated dataset. Let us generate 100 observations for the response variable and 3 covariates, where the true regression coefficient for the third covariate is 0.

```
> set.seed(2011*01*18)
> x <- matrix(rnorm(100*3),nrow=100,ncol=3)
> theta <- matrix(c(1,1,0),ncol=1)
> y <- x %*% theta + rnorm(100)
```

To start with we assume Normal residuals (the default). We need to specify the prior distribution for the regression coefficients, the model space and the residual variance. We specify a product iMOM prior on the coefficients with prior dispersion `tau=.131`, which targets the detection of standardized effect sizes above 0.2. Regarding the model space we use a Beta-binomial(1,1) prior (Scott and Berger, 2010). Finally, for the residual variance we set a minimally informative inverse gamma prior. For defining other prior distributions see the help for `msPriorSpec` (*e.g.* `momprior`, `emomprior` and `zellnerprior` can be used to define MOM, eMOM and Zellner priors, respectively).

```
> priorCoef <- imomprior(tau=.131)
> priorDelta <- modelbbprior(alpha.p=1,beta.p=1)
> priorVar <- igprior(.01,.01)
```

`modelSelection` enumerates all models when its argument `enumerate` is set to TRUE, otherwise it uses a Gibbs sampling scheme to explore the model space (saved in the slot postSample). It returns the visited model with highest posterior probability and the marginal posterior inclusion probabilities for each covariate (when using Gibbs sampling these are estimated via Rao-Blackwellization to improve accuracy).

```
> fit1 <- modelSelection(y=y, x=x, center=FALSE, scale=FALSE,
+ priorCoef=priorCoef, priorDelta=priorDelta, priorVar=priorVar)

Enumerating models...
Computing posterior probabilities........ Done.

> fit1$postMode

x1 x2 x3
 1  1  0

> fit1$margpp

        x1         x2         x3
1.00000000 1.00000000 0.04138239

> postProb(fit1)

  modelid family           pp
4     1,2 normal 9.586176e-01
8   1,2,3 normal 4.138239e-02
2       1 normal 2.765878e-13
3       2 normal 9.613353e-14
6     1,3 normal 1.221807e-15
7     2,3 normal 6.836394e-16
1         normal 8.690008e-20
5       3 normal 2.871131e-22

> fit2 <- modelSelection(y=y, x=x, center=FALSE, scale=FALSE,
+ priorCoef=priorCoef, priorDelta=priorDelta, priorVar=priorVar,
+ enumerate=FALSE, niter=1000)

Greedy searching posterior mode... Done.
Running Gibbs sampler........... Done.

> fit2$postMode

x1 x2 x3
 1  1  0

> fit2$margpp

        x1         x2         x3
1.00000000 1.00000000 0.04138239

> postProb(fit2,method='norm')
```

```
  modelid family         pp
1     1,2 normal 0.95861761
2   1,2,3 normal 0.04138239
```

```
> postProb(fit2,method='exact')
```

```
  modelid family   pp
1     1,2 normal 0.97
2   1,2,3 normal 0.03
```

The highest posterior probability model is the simulation truth, indicating that covariates 1 and 2 should be included and covariate 3 should be excluded. `fit1` was obtained by enumerating the $2^3 = 8$ possible models, whereas `fit2` ran 1,000 Gibbs iterations, delivering very similar results. `postProb` estimates posterior probabilities by renormalizing the probability of each model conditional to the set of visited models when `method=’norm’` (the default), otherwise it uses the proportion of Gibbs iterations spent on each model.

Below we run modelSelection again but now using Zellner's prior, with prior dispersion set to obtain the so-called Unit Information Prior. The posterior mode is still the data-generating truth, albeit its posterior probability has decreased substantially. This illustrates the core issue with NLPs: they tend to concentrate more posterior probability around the true model (or that closest in the Kullback-Leibler sense). This difference in behaviour relative to LPs becomes amplified as the number of considered models becomes larger, which may result in the latter giving a posterior probability that converges to 0 for the true model (Johnson and Rossell, 2012).

```
> priorCoef <- zellnerprior(tau=nrow(x))
> fit3 <- modelSelection(y=y, x=x, center=FALSE, scale=FALSE, niter=10^2,
+ priorCoef=priorCoef, priorDelta=priorDelta, priorVar=priorVar,
+ method='Laplace')
```

```
Enumerating models...
Computing posterior probabilities........ Done.
```

```
> postProb(fit3)
```

```
  modelid family           pp
4     1,2 normal 7.214937e-01
8   1,2,3 normal 2.785063e-01
2       1 normal 1.079508e-13
3       2 normal 3.565310e-14
6     1,3 normal 1.096444e-14
7     2,3 normal 3.827255e-15
1         normal 3.640151e-20
5       3 normal 1.394484e-21
```

Finally, we illustrate how to relax the assumption that residuals are Normally distributed. We may set the argument `family` to `'twopiecenormal'`, `'laplace'` or `'twopiecelaplace'` to allow for asymmetry (for two-piece Normal and two-piece Laplace) or thicker-than-normal tails (for Laplace and asymmetric Laplace). For instance, the maximum likelihood estimator under Laplace residuals is equivalent to median regression and under asymmetric Laplace residuals to quantile regression, thus these options can be interpreted as robust alternatives to Normal residuals. A nice feature is that regression coefficients can still be interpreted in the usual manner. These families add flexibility while maintaining analytical and computational tractability, e.g. they lead to convex optimization and efficient approximations to marginal likelihoods, and additionally to robustness we have found they can also lead to increased sensitivity to detect non-zero coefficients. Alas, computations under Normal residuals are inevitably faster, hence whenever this extra flexibility is not needed it is nice to be able to fall back onto the Normal family, particularly when p is large. `modelSelection` and `nlpMarginal` incorporate this option by setting `family=='auto'`, which indicates that the residual distribution should be inferred from the data. When p is small a full model enumeration is conducted, but when p is large the Gibbs scheme spends most time on models with high posterior probability, thus automatically focusing on the Normal family when it provides a good enough approximation and resorting to one of the alternatives when warranted by the data.

For instance, in the example below there's roughly 0.95 posterior probability that residuals are Normal, hence the Gibbs algorithm would spend most time on the (faster) Normal model. The two-piece Normal and two-piece Laplace (also known as asymmetric Laplace) incorporate an asymmetry parameter $\alpha \in [-1, 1]$, where $\alpha = 0$ corresponds to the symmetric case (i.e. Normal and Laplace residuals). We set a NLP on $\text{atanh}(\alpha) \in (-\infty, infty)$ so that under the asymmetric model we push prior mass away from $\alpha = 0$, which intuitively means we are interested in finding significant departures from asymmetry and otherwise we fall back onto the simpler symmetric case.

```
> priorCoef <- imomprior(tau=.131)
> fit4 <- modelSelection(y=y, x=x, center=FALSE, scale=FALSE,
+ priorCoef=priorCoef, priorDelta=priorDelta, priorVar=priorVar,
+ priorSkew=imomprior(tau=.131),family='auto')
```

```
Enumerating models...
Computing posterior probabilities.......... Done.
```

```
> head(postProb(fit4))
```

```
   modelid          family           pp
4      1,2          normal 9.472921e-01
```

```
8    1,2,3          normal 4.089348e-02
20     1,2         laplace 8.904186e-03
12     1,2  twopiecenormal 2.693917e-03
24   1,2,3         laplace 1.024381e-04
28     1,2 twopiecelaplace 6.088958e-05
```

All examples above use `modelSelection`, which is based on product NLPs (5). `mombf` also provides some (limited) functionality for additive NLPs (4). The code below contains an example based on the Hald data, which has $n = 13$ observations, a continuous response variable and $p = 4$ predictors. We load the data and fit a linear regression model.

```
> data(hald)
> dim(hald)

[1] 13  5

> lm1 <- lm(hald[,1] ~ hald[,2] + hald[,3] + hald[,4] + hald[,5])
> summary(lm1)

Call:
lm(formula = hald[, 1] ~ hald[, 2] + hald[, 3] + hald[, 4] +
    hald[, 5])

Residuals:
    Min      1Q  Median      3Q     Max
-3.1750 -1.6709  0.2508  1.3783  3.9254

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  62.4054    70.0710   0.891   0.3991
hald[, 2]     1.5511     0.7448   2.083   0.0708 .
hald[, 3]     0.5102     0.7238   0.705   0.5009
hald[, 4]     0.1019     0.7547   0.135   0.8959
hald[, 5]    -0.1441     0.7091  -0.203   0.8441
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 2.446 on 8 degrees of freedom
Multiple R-squared:  0.9824,      Adjusted R-squared:  0.9736
F-statistic: 111.5 on 4 and 8 DF,  p-value: 4.756e-07

> V <- summary(lm1)$cov.unscaled
> diag(V)

 (Intercept)    hald[, 2]    hald[, 3]    hald[, 4]    hald[, 5]
820.65457471   0.09271040   0.08756026   0.09520141   0.08403119
```

Bayes factors between a fitted model and a submodel where some of the variables are dropped can be easily computed from the `lm` output using functions `mombf` and `imombf`. As an example here we drop the second coefficient from the model. Parameter g corresponds to the prior dispersion τ in our notation. There are several options to estimate numerically iMOM Bayes factors (for MOM they have closed form), here we compare adaptive quadrature with a Monte Carlo estimate.

```
> mombf(lm1,coef=2,g=taumom)

         [,1]
[1,] 2.475336

> imombf(lm1,coef=2,g=tauimom,method='adapt')

         [,1]
[1,] 1.536302

> imombf(lm1,coef=2,g=tauimom,method='MC',B=10^5)

         [,1]
[1,] 1.534466
```

4 Parameter estimation

A natural question after performing model selection is obtaining estimates for the parameters. Rossell and Telesca (2016) developed a general posterior sampling framework for NLPs based on Gibbs sampling an augmented probability model that expresses NLPs as mixtures of truncated distributions. The methodology is implemented in function `rnlp`. Its basic use is simple, by setting parameter `msfit` to the output of `modelSelection` the function produces posterior samples under each model γ visited by `modelSelection`. The number of samples is proportional to its posterior probability $p(\gamma \mid y)$, thus averaging the output gives Bayesian model averaging estimates $E(\theta \mid y) = \sum_{\gamma} E(\theta \mid \gamma, y) p(\gamma \mid y)$ and likewise for the residual variance $E(\phi \mid y)$.

```
> priorCoef <- momprior(tau=.348)
> priorDelta <- modelbbprior(alpha.p=1,beta.p=1)
> fit1 <- modelSelection(y=y, x=x, center=FALSE, scale=FALSE,
+ priorCoef=priorCoef, priorDelta=priorDelta)

Enumerating models...
Computing posterior probabilities........ Done.

> th <- rnlp(y=as.numeric(y),x=x,msfit=fit1,priorCoef=priorCoef,niter=10000)
> colMeans(th)

       beta1        beta2        beta3          phi
 1.006847248  0.943283241 -0.004003268  1.001267547
```

```
> head(th)

         beta1     beta2 beta3       phi
[1,] 1.0699785 0.8691159     0 0.8907935
[2,] 0.9656519 0.8487383     0 1.2229253
[3,] 0.9649019 0.8758050     0 1.0516350
[4,] 0.9050806 0.8189167     0 0.8862611
[5,] 0.9087373 0.9623837     0 1.0739552
[6,] 1.1247194 0.8844062     0 0.8097169
```

From this output we can obtain model-specific posterior means.

```
> model <- apply(th!=0,1,function(z) paste(which(z),collapse=','))
> table(model)

model
1,2,3,4   1,2,4
    250    9750

> colMeans(th[model=='1,2,4',])

    beta1     beta2     beta3       phi
1.0061922 0.9433184 0.0000000 1.0016814

> colMeans(th[model=='1,2,3,4',])

     beta1      beta2      beta3        phi
 1.0323933  0.9419133 -0.1601307  0.9851272
```

Another interestingn use of `rnlp` is to obtain posterior samples under a generic non-local posterior

$$p(\theta \mid y) \propto d(\theta)N(\theta; m, V),$$

where $d(\theta)$ is a non-local prior penalty and $N(\theta; m, V)$ is the normal posterior that one would obtain under the underlying local prior. For instance, suppose our prior is proportional to Zellner's prior times a product MOM penalty $p(\theta) \propto \prod_j \theta_j^2 N(\theta; 0, n\tau\phi(X'X)^{-1})$ where ϕ is the residual variance, then the posterior is proportional to $p(\theta \mid y) \propto \prod_j \theta_j^2 N(\theta; m, V)$ where $m = s_\tau(X'X)^{-1}X'y$, $V = \phi(X'X)^{-1}s_\tau^2$ where $s_\tau = n\tau/(1 + n\tau)$ is the usual ridge regression shrinkage factor. We may obtain posterior samples as follows. Note that the posterior mean is close to that obtained above.

```
> tau= 0.348
> shrinkage= nrow(x)*tau/(1+nrow(x)*tau)
> V= shrinkage * solve(t(x) %*% x)
> m= as.vector(shrinkage * V %*% t(x) %*% y)
> phi= mean((y - x%*%m)^2)
> th= rnlp(m=m,V=phi * V,priorCoef=momprior(tau=tau))
> colMeans(th)

     beta1      beta2      beta3
 1.0038342  0.9289234 -0.1617318
```

5 Exact inference for block-diagonal regression

Papaspiliopoulos and Rossell (2016) proposed a fast computational framework to compute exact posterior model probabilities, variable inclusion probabilities and parameter estimates for Normal linear regression when the $X'X$ matrix is block-diagonal. Naturally this includes the important particular case of orthogonal regression where $X'X$ is diagonal. The framework performs a fast model search that finds the best model of each size (i.e. with $1, 2, \ldots, p$ variables) and a fast deterministic integration to account for the fact that the residual variance is uknown (the residual variance acts as a "cooling" parameter that affects how many variables are included, hence the associated uncertainty must be dealt with appropriately). The function `postModeOrtho` tackles the diagonal $X'X$ case and `postModeBlockDiag` the block-diagonal case.

The example below simulates $n = 210$ observations with $p = 200$ variables where all regression coefficients are 0 except for the last three (0.5, 0.75, 1) and the residual variance is one. We then perform variable selection under Zellner's and the MOM prior.

```
> set.seed(1)
> p <- 200; n <- 210
> x <- scale(matrix(rnorm(n*p),nrow=n,ncol=p),center=TRUE,scale=TRUE)
> S <- cov(x)
> e <- eigen(cov(x))
> x <- t(t(x %*% e$vectors)/sqrt(e$values))
> th <- c(rep(0,p-3),c(.5,.75,1)); phi <- 1
> y <- x %*% matrix(th,ncol=1) + rnorm(n,sd=sqrt(phi))
> priorDelta=modelbinomprior(p=1/p)
> priorVar=igprior(0.01,0.01)
> priorCoef=zellnerprior(tau=n)
> pm.zell <-
+ postModeOrtho(y,x=x,priorCoef=priorCoef,priorDelta=priorDelta,
+ priorVar=priorVar,bma=TRUE)
> head(pm.zell$models)

              modelid          pp
4         200,199,198 0.828818155
5      200,199,198,54 0.037323461
107    200,199,198,11 0.006070065
108   200,199,198,186 0.004133534
110    200,199,198,36 0.003679338
6   200,199,198,54,11 0.000316667

> priorCoef=momprior(tau=0.348)
> pm.mom <- postModeOrtho(y,x=x,priorCoef=priorCoef,priorDelta=priorDelta,
+ priorVar=priorVar,bma=TRUE)
> head(pm.mom$models)
```

```
              modelid           pp
4         200,199,198 9.779392e-01
5      200,199,198,54 1.144910e-02
107    200,199,198,11 1.209685e-03
108   200,199,198,186 7.314291e-04
110    200,199,198,36 6.262828e-04
6   200,199,198,54,11 1.717659e-05
```

postModelBlockDiag returns a list with the best model of each size and its corresponding (exact) posterior probability, displayed in Figure 2 (left panel). It also returns marginal inclusion probabilities and BMA estimates, shown in the right panel. The code required to produce these figures is below.

```
> par(mar=c(5,5,1,1))
> nvars <- sapply(strsplit(as.character(pm.zell$models$modelid),split=','),length)
> plot(nvars,pm.zell$models$pp,ylab=expression(paste("p(",gamma,"|y)")),
+ xlab=expression(paste("|",gamma,"|")),cex.lab=1.5,ylim=0:1,xlim=c(0,50))
> sel <- pm.zell$models$pp>.05
> text(nvars[sel],pm.zell$models$pp[sel],pm.zell$models$modelid[sel],pos=4)
> nvars <- sapply(strsplit(as.character(pm.mom$models$modelid),split=','),length)
> points(nvars,pm.mom$models$pp,col='gray',pch=17)
> sel <- pm.mom$models$pp>.05
> text(nvars[sel],pm.mom$models$pp[sel],pm.mom$models$modelid[sel],pos=4,col='gray')
> legend('topright',c('Zellner','MOM'),pch=c(1,17),col=c('black','gray'),cex=1.5)
```

```
> par(mar=c(5,5,1,1))
> ols <- (t(x) %*% y) / colSums(x^2)
> plot(ols,pm.zell$bma$coef,xlab='Least squares estimate',
+ ylab=expression(paste('E(',beta[j],'|y)')),cex.lab=1.5,cex.axis=1.2,col=1)
> points(ols,pm.mom$bma$coef,pch=3,col='darkgray')
> legend('topleft',c('Zellner','MOM'),pch=c(1,3),col=c('black','darkgray'))
```

We now illustrate similar functionality under block-diagonal $X'X$. To this end we consider a total $p = 100$ variables split into 10 blocks of 10 variables each, generated in such a way that they all have unit variance and within-blocks pairwise correlation of 0.5. The first block has three non-zero coefficients, the second block two and the remaining blocks contain no active variables.

```
> set.seed(1)
> p <- 100; n <- 110
> blocksize <- 10
> blocks <- rep(1:(p/blocksize),each=blocksize)
> x <- scale(matrix(rnorm(n*p),nrow=n,ncol=p),center=TRUE,scale=TRUE)
> S <- cov(x)
> e <- eigen(cov(x))
> x <- t(t(x %*% e$vectors)/sqrt(e$values))
```

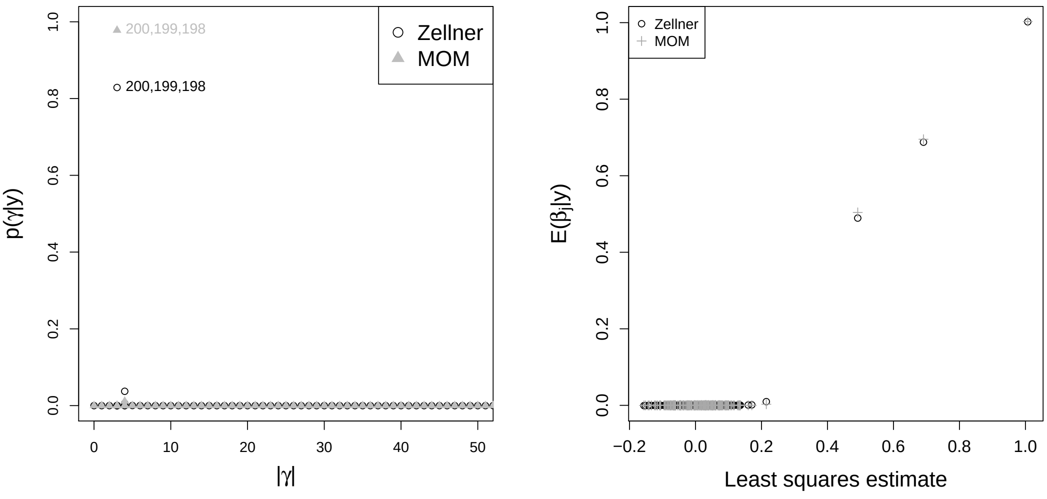

Figure 2: Posterior probability under simulated orthogonal data

```
> Sblock <- diag(blocksize)
> Sblock[upper.tri(Sblock)] <- Sblock[lower.tri(Sblock)] <- 0.5
> vv <- eigen(Sblock)$vectors
> sqSblock <- vv %*% diag(sqrt(eigen(Sblock)$values)) %*% t(vv)
> for (i in 1:(p/blocksize)) x[,blocks==i] <- x[,blocks==i] %*% sqSblock
> th <- rep(0,ncol(x))
> th[blocks==1] <- c(rep(0,blocksize-3),c(.5,.75,1))
> th[blocks==2] <- c(rep(0,blocksize-2),c(.75,-1))
> phi <- 1
> y <- x %*% matrix(th,ncol=1) + rnorm(n,sd=sqrt(phi))
```

postModeBlockDiag performs the model search using an algorithm nicknamed ”Coolblock” (as it is motivated by treating the residual variance as a cooling parameter). Briefly, Coolblock visits a models of sizes ranging from 1 to p and returns the best model for that given size, thus also helping identify the best model overall.

```
> priorCoef=zellnerprior(tau=n)
> priorDelta=modelbinomprior(p=1/p)
> priorVar=igprior(0.01,0.01)
> pm <- postModeBlockDiag(y=y,x=x,blocks=blocks,priorCoef=priorCoef,
+ priorDelta=priorDelta,priorVar=priorVar,bma=TRUE)
> head(pm$models)
```

```
        modelid nvars           pp     pp.upper
1                   0 1.764684e-24 1.764684e-24
2            10     1 7.864656e-12 7.864656e-12
3          9,10     2 7.232276e-10 7.232276e-10
```

```
4      9,10,20     3 5.857901e-07 5.857901e-07
5   9,10,19,20     4 4.814026e-03 4.814026e-03
6 8,9,10,19,20     5 8.430706e-01 8.430706e-01
```

```
> head(pm$postmean.model)
```

```
       modelid X1 X2 X3 X4 X5 X6 X7       X8        X9      X10 X11 X12 X13 X14
1               0  0  0  0  0  0  0 0.000000 0.0000000 0.000000   0   0   0   0
2           10  0  0  0  0  0  0  0 0.000000 0.0000000 1.688461   0   0   0   0
3         9,10  0  0  0  0  0  0  0 0.000000 0.8786666 1.249127   0   0   0   0
4      9,10,20  0  0  0  0  0  0  0 0.000000 0.8786666 1.249127   0   0   0   0
5   9,10,19,20  0  0  0  0  0  0  0 0.000000 0.8786666 1.249127   0   0   0   0
6 8,9,10,19,20  0  0  0  0  0  0  0 0.657439 0.6595203 1.029981   0   0   0   0
  X15 X16 X17 X18       X19       X20 X21 X22 X23 X24 X25 X26 X27 X28 X29 X30
1   0   0   0   0 0.0000000  0.000000   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0 0.0000000  0.000000   0   0   0   0   0   0   0   0   0   0
3   0   0   0   0 0.0000000  0.000000   0   0   0   0   0   0   0   0   0   0
4   0   0   0   0 0.0000000 -0.740712   0   0   0   0   0   0   0   0   0   0
5   0   0   0   0 0.8089637 -1.145194   0   0   0   0   0   0   0   0   0   0
6   0   0   0   0 0.8089637 -1.145194   0   0   0   0   0   0   0   0   0   0
  X31 X32 X33 X34 X35 X36 X37 X38 X39 X40 X41 X42 X43 X44 X45 X46 X47 X48 X49
1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  X50 X51 X52 X53 X54 X55 X56 X57 X58 X59 X60 X61 X62 X63 X64 X65 X66 X67 X68
1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  X69 X70 X71 X72 X73 X74 X75 X76 X77 X78 X79 X80 X81 X82 X83 X84 X85 X86 X87
1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  X88 X89 X90 X91 X92 X93 X94 X95 X96 X97 X98 X99 X100
1   0   0   0   0   0   0   0   0   0   0   0   0    0
2   0   0   0   0   0   0   0   0   0   0   0   0    0
3   0   0   0   0   0   0   0   0   0   0   0   0    0
4   0   0   0   0   0   0   0   0   0   0   0   0    0
5   0   0   0   0   0   0   0   0   0   0   0   0    0
6   0   0   0   0   0   0   0   0   0   0   0   0    0
```

Figure 3 shows a LASSO-type plot with the posterior means under the

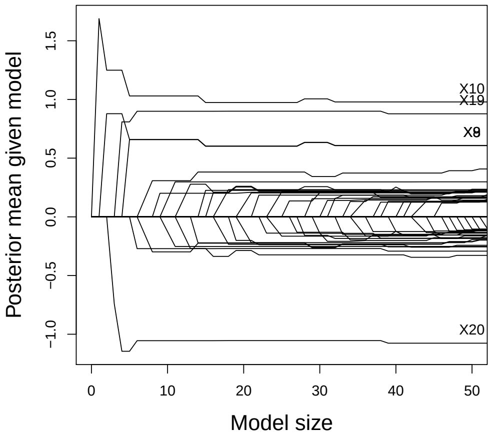

Figure 3: Coolblock algorithm: posterior mean of regression coefficients under best model of each size

best model of each size visited by Coolblock. We appreciate how the truly active variables 8, 9, 10, 19 and 20 are picked up first.

```
> maxvars=50
> ylim=range(pm$postmean.model[,-1])
> plot(NA,NA,xlab='Model size',
+   ylab='Posterior mean given model',
+   xlim=c(0,maxvars),ylim=ylim,cex.lab=1.5)
> visited <- which(!is.na(pm$models$pp))
> for (i in 2:ncol(pm$postmean.model)) {
+   lines(pm$models$nvars[visited],pm$postmean.model[visited,i])
+ }
> text(maxvars, pm$postmean.model[maxvars,which(th!=0)+1],
+ paste('X',which(th!=0),sep=''), pos=3)
```

6 Bayes factors for generalized linear models

At the moment `mombf` provides some limited functionality for generalized linear models. `pmomLM` implements probit models following the framework in Rossell et al. (2013), see also Nikooienejad et al. (2016) for an alternative framework that bypasses the need to sample θ to improve the mixing of the Markov Chain.

As an illustration we simulate $n = 50$ observations from a probit regression model with $p = 2$ correlated predictors and $\theta = (\log(2), 0)$. The predictors are stored in the matrix x, the success probabilities in the vector p and the observed responses in the vector y. For reproducibility purposes we set the random number generator seed.

```
> set.seed(4*2*2008)
> n <- 50; theta <- c(log(2),0)
> x <- matrix(NA,nrow=n,ncol=2)
> x[,1] <- rnorm(n,0,1); x[,2] <- rnorm(n,.5*x[,1],1)
> p <- pnorm(x %*% matrix(theta,ncol=1))
> y <- rbinom(n,1,p)
```

```
> th <- pmomLM(y=y,x=x,xadj=rep(1,n),niter=10000)
```

```
Running MCMC...........Done.
```

```
> head(th$postModel)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    1    0
[3,]    1    0
[4,]    1    0
[5,]    1    1
[6,]    1    0
```

```
> table(apply(th$postModel,1,paste,collapse=','))
```

```
 0,0  0,1  1,0  1,1 
 237   19 7482 1262 
```

An alternative to pmomLM, which considers all possible models under product NLPs, is to compute Bayes factors between pairs of models under additive NLPs using functions momknown and imomknown.

Before computing Bayes factors, we fit a probit regression model with the function glm. The maximum likelihood estimates are stored in thetahat and the asymptotic covariance matrix in V. These functions take as primary arguments a vector of regression coefficients and their covariance matrix, and hence they can be used in any setting where one has a statistic that is asymptotically sufficient and normally distributed. The resulting Bayes factors are approximate. The functions also allow for the presence of a dispersion parameter sigma, *i.e.* the covariance of the regression coefficients is sigma*V, but they assume that sigma is known. The probit regression model that we simulated has no over-dispersion and hence it corresponds to sigma=1. We first compare the full model with the model resulting from excluding the second covariate (the first term corresponds to the intercept), setting $g = 0.5$ for illustration.

```
> glm1 <- glm(y~x[,1]+x[,2],family=binomial(link = "probit"))
> thetahat <- coef(glm1)
> V <- summary(glm1)$cov.scaled
> g <- .5
> bfmom.1 <- momknown(thetahat[2],V[2,2],n=n,g=g,sigma=1)
> bfimom.1 <- imomknown(thetahat[2],V[2,2],n=n,nuisance.theta=2,g=g,sigma=1)
> bfmom.1
```

```
         [,1]
[1,] 4.262401
```

```
> bfimom.1
```

```
         [,1]
[1,] 3.336888
```

Both priors result in evidence for including the first covariate. We now check whether the second covariate can be dropped.

```
> bfmom.2 <- momknown(thetahat[3],V[3,3],n=n,g=g,sigma=1)
> bfimom.2 <- imomknown(thetahat[3],V[3,3],n=n,nuisance.theta=2,g=g,sigma=1)
> bfmom.2
```

```
           [,1]
[1,] 0.02784354
```

```
> bfimom.2
```

```
            [,1]
[1,] 0.008250121
```

Both Mom and iMom BF provide strong evidence in favor of the simpler model, *i.e.* excluding `x[,2]`. To compare the full model with the model that has no covariates (*i.e.* only the constant term remains) we use the same routines, passing a vector as the first argument and a matrix as the second argument.

```
> bfmom.0 <- momknown(thetahat[2:3],V[2:3,2:3],n=n,g=g,sigma=1)
> bfimom.0 <- imomknown(thetahat[2:3],V[2:3,2:3],n=n,nuisance.theta=2,g=g,sigma=1)
> bfmom.0
```

```
          [,1]
[1,] 0.5272556
```

```
> bfimom.0
```

```
         [,1]
[1,] 0.953978
```

Based on the resulting BF being close to 1, it is not clear whether the full model is preferable to the model with no covariates.

The BF can be used to easily compute posterior probabilities for each of the four considered models: no covariates, only `x[,1]`, only `x[,2]` and both `x[,1]` and `x[,2]`. We assume equal probabilities *a priori*.

```
> prior.prob <- rep(1/4,4)
> bf <- c(bfmom.0,bfmom.1,bfmom.2,1)
> pos.prob <- prior.prob*bf/sum(prior.prob*bf)
> pos.prob
```

```
[1] 0.090632677 0.732686026 0.004786169 0.171895128
```

The model with the highest posterior probability is the one including only `x[,1]`, *i.e.* the correct model, and the model with the lowest posterior probability is that including only `x[,2]`.

References

T. Chekouo, F.C. Stingo, J.D. Doecke, and K.-A. Do. mirna–target gene regulatory networks: A bayesian integrative approach to biomarker selection with application to kidney cancer. *Biometrics*, 71(2):428–438, 2015.

Rodrigo A Collazo, Jim Q Smith, et al. A new family of non-local priors for chain event graph model selection. *Bayesian Analysis*, (in press), 2016.

G. Consonni and L. La Rocca. On moment priors for Bayesian model choice with applications to directed acyclic graphs. In J.M. Bernardo, M.J. Bayarri, J.O. Berger, A.P. Dawid, D. Heckerman, A.F.M Smith, and M. West, editors, *Bayesian Statistics 9 - Proceedings of the ninth Valencia international meeting*, pages 119–144. Oxford University Press, 2010.

J. Fúquene, M.F.J. Steel, and D. Rossell. On choosing mixture components via non-local priors. *arXiv 1604.00314v1*, pages 1–43, 2016.

V.E. Johnson and D. Rossell. Prior densities for default bayesian hypothesis tests. *Journal of the Royal Statistical Society B*, 72:143–170, 2010.

V.E. Johnson and D. Rossell. Bayesian model selection in high-dimensional settings. *Journal of the American Statistical Association*, 24(498):649–660, 2012.

F. Liang, R. Paulo, G. Molina, M.A. Clyde, and J.O. Berger. Mixtures of g-priors for Bayesian variable selection. *Journal of the American Statistical Association*, 103:410–423, 2008.

Amir Nikooienejad, Wenyi Wang, and Valen E Johnson. Bayesian variable selection for binary outcomes in high dimensional genomic studies using non-local priors. *Bioinformatics*, btv764:1–8, 2016.

O. Papaspiliopoulos and D. Rossell. Scalable variable selection and model averaging under block-orthogonal design. *arXiv*, pages 1–19, 2016.

D. Rossell and F.J. Rubio. Tractable bayesian variable selection: beyond normality. *arXiV*, 1609.01708:1–59, 2016.

D. Rossell and D. Telesca. Non-local priors for high-dimensional estimation. *Journal of the American Statistical Association*, (in press):1–33, 2016.

D. Rossell, D. Telesca, and V.E. Johnson. High-dimensional Bayesian classifiers using non-local priors. In *Statistical Models for Data Analysis XV*, pages 305–314. Springer, 2013.

J.G. Scott and J.O Berger. Bayes and empirical Bayes multiplicity adjustment in the variable selection problem. *The Annals of Statistics*, 38(5): 2587–2619, 2010.

M. Shin, A. Bhattacharya, and V.E. Johnson. Scalable Bayesian variable selection using nonlocal prior densities in ultrahigh-dimensional settings. *Texas A&M University (technical report)*, pages 1–33, 2015.