WORKING PAPER SERIES

WORKING PAPER NO 4, 2007

ESI

Computational Efficiency in Bayesian Model and Variable Selection

By

Jana Eklund
Bank of England

Sune Karlsson
Örebro University

Jana.Eklund@bankofengland.co.uk

Sune.Karlsson@esi.oru.se

http://www.oru.se/esi/wps

SE-701 82 Örebro

SWEDEN

ISSN 1403-0586

Computational Efficiency in Bayesian Model and Variable Selection *

Jana Eklund
Bank of England†

Sune Karlsson
Örebro University‡

September 10, 2007

Abstract

Large scale Bayesian model averaging and variable selection exercises present, despite the great increase in desktop computing power, considerable computational challenges. Due to the large scale it is impossible to evaluate all possible models and estimates of posterior probabilities are instead obtained from stochastic (MCMC) schemes designed to converge on the posterior distribution over the model space. While this frees us from the requirement of evaluating all possible models the computational effort is still substantial and efficient implementation is vital. Efficient implementation is concerned with two issues: the efficiency of the MCMC algorithm itself and efficient computation of the quantities needed to obtain a draw from the MCMC algorithm. We evaluate several different MCMC algorithms and find that relatively simple algorithms with local moves perform competitively except possibly when the data is highly collinear. For the second aspect, efficient computation within the sampler, we focus on the important case of linear models where the computations essentially reduce to least squares calculations. Least squares solvers that update a previous model estimate are appealing when the MCMC algorithm makes local moves and we find that the Cholesky update is both fast and accurate.
Keywords: Bayesian Model Averaging, Sweep operator, Cholesky decomposition, QR decomposition, Swendsen-Wang algorithm
JEL-codes: C110, C150, C520, C630

*The views expressed in this paper are those of the authors, and not necessarily those of the Bank of England. The work on this paper was carried out while the first author was affiliated with the Stockholm School of Economics.

†Jana.Eklund@bankofengland.co.uk
‡Sune.Karlsson@esi.oru.se

1 Introduction

Bayesian model averaging (BMA) was introduced by Leamer (1978) some 30 years ago. It has grown in popularity in the last decade as new theoretical developments have become available and computer power easily accessible. Hoeting, Madigan, Raftery, and Volinsky (1999) provide a comprehensive historical overview and summary of the literature on this topic. BMA has been applied successfully to many statistical model classes, including linear regression, Raftery, Madigan, and Hoeting (1997), Fernández, Ley, and Steel (2001); discrete graphical models, Madigan and Raftery (1994); survival analysis, Raftery, Madigan, and Volinsky (1995); factor-based models, Koop and Potter (2004); and large macroeconomic panels, Jacobson and Karlsson (2004), in all cases improving predictive performance in the presence of model uncertainty.

At first glance, Bayesian model averaging is straightforward to implement: one needs the marginal distribution of the data, the prior probabilities of the models and the posterior distribution of the quantity of interest conditional on each model. In linear regression, for nicely behaved prior distributions, all these components are available in closed form. Even if one has the closed form, another problem is that in many applications, the model space is too large to allow enumeration of all models, and beyond 20-25 variables, see George and McCulloch (1997), estimation of posterior model probabilities and BMA must be based on a sample of models.

For a variable selection problem in a linear regression setting with 50 potential explanatory variables, implying $2^{50} \approx 10^{15}$ different models, the CPU time for a brute force attack would be close to 5 millennia with our fastest algorithm on a standard desktop PC. While this can obviously be speeded up by throwing more hardware at the problem and the computations are trivial to parallelize, the computational burden is simply too large. Assuming a 100-fold increase in processor speed it would still take over 17 days to evaluate all the models on a 1024 node cluster, which clearly puts a brute force approach beyond everyday use. Instead, Markov chain Monte Carlo (MCMC) methods provide a stochastic method of obtaining a sample from the posterior distribution over the model space. These methods have to be run sufficiently long in order to achieve satisfactory level of convergence to the posterior distribution. The number of steps required depend crucially on the design of the sampler and how well it moves through the model space. Efficient computation within the sampler is of equal importance, e.g. computing he marginal likelihood or generating draws from the posterior distribution.

This paper investigates these issues in the context of linear regression models. Algorithms for solving least squares problems and various MCMC algorithms for exploring the model space are evaluated both in terms of speed and accuracy.

The paper is organized as follows, the next section 2 introduces Bayesian model averaging and sets the stage for the remainder of the paper. Section 3 reviews methods for solving linear normal equations and Section 4 evaluates the speed and accuracy of the algorithms. Section 5 describes several Markov chain samplers and Section 6 evaluates their performance. Finally, Section 7 concludes.

2 Bayesian model averaging and linear regression

Bayesian model averaging and model selection depends on the ability to calculate posterior model probabilities

$$p\left(\mathcal{M}_i|\,\mathbf{y}\right) = \frac{m\left(\mathbf{y}|\,\mathcal{M}_i\right)p\left(\mathcal{M}_i\right)}{\sum_{j=1}^{M} m\left(\mathbf{y}|\,\mathcal{M}_j\right)p\left(\mathcal{M}_j\right)}, \tag{1}$$

and in particular the marginal likelihoods

$$m\left(\mathbf{y}|\,\mathcal{M}_i\right) = \int L\left(\mathbf{y}|\,\boldsymbol{\theta}_i, \mathcal{M}_i\right)p\left(\boldsymbol{\theta}_i|\,\mathcal{M}_i\right)\mathrm{d}\boldsymbol{\theta}_i, \tag{2}$$

for the considered models $\mathcal{M}_i$, $i = 1, \ldots, M$. If M is not too large and the models are tractable in the sense that the marginal likelihoods are available in closed form, this can be solved by brute force calculation of all the posterior model probabilities. In the large scale problems motivating the current paper, brute force calculations are not feasible even when the marginal likelihood is available in closed form.

Instead, Markov chain Monte Carlo methods are used to estimate the posterior model probabilities. These are typically variations on the reversible jump MCMC (RJMCMC) scheme developed by Green (1995). A typical RJMCMC algorithm, and the basis for all the algorithms considered in this paper, is given as Algorithm 1. The Markov chain moves between models and converges to the posterior model probabilities under general conditions, and the output can be used to estimate the posterior model probabilities after a suitable burn-in. With the very large model sets considered here, it is clear that the chain can not visit all, or even a majority, of the models in a reasonable amount of time. It is thus important that the chain moves well through the model space and finds the models with high posterior probabilities.[1] The output of the chain can then be taken as an estimate of the (normalized) posterior probabilities for the visited models. When the marginal likelihoods are available in closed form, the exact posterior probabilities, conditional on the set of visited models, can be calculated. Besides being more accurate than the relative frequencies from the chain, this can also be used as a check of the convergence properties of the chain.

The way the chain moves through the model space is largely determined by the type of moves allowed and their probabilities, or in terms of Algorithm 1, the jump distribution $j(\mathcal{M}'|\mathcal{M})$. Let $\boldsymbol{\gamma}$ be a binary vector with $\gamma_j = 1$ indicating inclusion and $\gamma_j = 0$ exclusion of variable j in a variable selection problem. A basic implementation of the RJMCMC algorithm for variable selection would use simple, local, model changing moves:

- **Add/Drop** Draw an integer j from a uniform distribution on $1, \ldots, N$ and flip the value of γ_j. This adds or drops variable j from the model. This gives $j(\mathcal{M}'|\mathcal{M}) = j(\mathcal{M}|\mathcal{M}') = 1/N$.

- **Swap** Select an index i at random from $\{i : \gamma_i = 1\}$, an index j from $\{j : \gamma_j = 0\}$ and set $\gamma_i = 0, \gamma_j = 1$. This replaces variable i with variable j. The jump probabilities are $j(\mathcal{M}'|\mathcal{M}) = j(\mathcal{M}|\mathcal{M}') = 1/k(N-k)$, where k is the number of variables in the model. This replaces variable i with variable j.

[1] The capture-recapture technique proposed by George and McCulloch (1997) can be used to estimate the fraction of the total posterior probability accounted for by the chain.

Algorithm 1 Reversible jump Markov chain Monte Carlo

Suppose that the Markov chain is at model $\mathcal{M}$, having parameters $\boldsymbol{\theta}_{\mathcal{M}}$, where $\boldsymbol{\theta}_{\mathcal{M}}$ has dimension $\dim(\boldsymbol{\theta}_{\mathcal{M}})$.

A: Marginal likelihood is not available in closed form

1. Propose a jump from model $\mathcal{M}$ to a new model $\mathcal{M}'$ with probability $j(\mathcal{M}'|\mathcal{M})$.
2. Generate a vector $\mathbf{u}$ (which can have a different dimension than $\boldsymbol{\theta}_{\mathcal{M}'}$) from a specified proposal density $q(\mathbf{u}|\boldsymbol{\theta}_{\mathcal{M}}, \mathcal{M}, \mathcal{M}')$.
3. Set $(\boldsymbol{\theta}_{\mathcal{M}'}, \mathbf{u}') = g_{\mathcal{M},\mathcal{M}'}(\boldsymbol{\theta}_{\mathcal{M}}, \mathbf{u})$, where $g_{\mathcal{M},\mathcal{M}'}$ is a specified invertible function. Hence $\dim(\boldsymbol{\theta}_{\mathcal{M}}) + \dim(\mathbf{u}) = \dim(\boldsymbol{\theta}_{\mathcal{M}'}) + \dim(\mathbf{u}')$. Note that $g_{\mathcal{M},\mathcal{M}'} = g^{-1}_{\mathcal{M}',\mathcal{M}}$.
4. Accept the proposed move with probability

$$\alpha = \min\left\{1, \frac{L(\mathbf{y}|\boldsymbol{\theta}_{\mathcal{M}',\mathcal{M}'})\, p(\boldsymbol{\theta}_{\mathcal{M}'}|\mathcal{M}')\, p(\mathcal{M}')\, j(\mathcal{M}|\mathcal{M}')}{L(\mathbf{y}|\boldsymbol{\theta}_{\mathcal{M}}, \mathcal{M})\, p(\boldsymbol{\theta}_{\mathcal{M}}|\mathcal{M})\, p(\mathcal{M})\, j(\mathcal{M}'|\mathcal{M})} \times \frac{q(\mathbf{u}'|\boldsymbol{\theta}_{\mathcal{M}'}, \mathcal{M}', \mathcal{M})}{q(\mathbf{u}|\boldsymbol{\theta}_{\mathcal{M}}, \mathcal{M}, \mathcal{M}')} \left|\frac{\partial g_{\mathcal{M},\mathcal{M}'}(\boldsymbol{\theta}_{\mathcal{M}}, \mathbf{u})}{\partial(\boldsymbol{\theta}_{\mathcal{M}}, \mathbf{u})}\right|\right\}. \quad (3)$$

5. Set $\mathcal{M} = \mathcal{M}'$ if the move is accepted and stay at $\mathcal{M}$ otherwise.

B: Marginal likelihood is available in closed form

When the marginal likelihood is available in closed form the acceptance probability (3) can be simplified substantially by employing the fiction that the proposal distribution for the parameters is the posterior distribution, $q(\mathbf{u}|\boldsymbol{\theta}_{\mathcal{M}}, \mathcal{M}, \mathcal{M}')$ is the posterior $p(\boldsymbol{\theta}_{\mathcal{M}'}|\mathbf{y}, \mathcal{M}')$. The Jacobian is then unity and the acceptance probability simplifies to

$$\alpha = \min\left\{1, \frac{m(\mathbf{y}|\mathcal{M}')\, p(\mathcal{M}')\, j(\mathcal{M}|\mathcal{M}')}{m(\mathbf{y}|\mathcal{M})\, p(\mathcal{M})\, j(\mathcal{M}'|\mathcal{M})}\right\}. \quad (4)$$

Clearly steps 2 and 3 above are then unnecessary and the algorithm simplifies to

1. Propose a jump from model $\mathcal{M}$ to a new model $\mathcal{M}'$ with probability $j(\mathcal{M}'|\mathcal{M})$.
2. Accept the move with probability (4) otherwise stay at the current model.

Using only the Add/Drop move yields the Markov chain Monte Carlo model composition $(MC)^3$ algorithm of Madigan and York (1995). The Swap move is used in conjunction with the Add/Drop by, among others; Denison, Mallick, and Smith (1998), and Jacobson and Karlsson (2004).

Moves like Add/Drop and Swap are attractive since they are very easy to implement, but their simple nature might cause the chain to mix badly and converge slowly. Consequently a number of alternative schemes has been proposed with the aim of speeding up convergence, in particular for the common situation, where there is a high degree of multicollinearity between potential explanatory variables. However, little is known about the relative merits of these sampling schemes, and one aim of this paper is to evaluate them against a common set of benchmarks. There are two important aspects which we focus on:

1. How quickly will the algorithm account for all but a negligible portion of the total posterior probability?

2. How quickly will the algorithm converge to the posterior distribution over the (visited) models?

In general, the first item will be easier to satisfy, and is needed with models where the marginal likelihood is available in closed form and exact posterior model probabilities can be calculated for the visited models. The second might require many more steps of the Markov chain and is needed in order to estimate the posterior model probabilities from the output of the chain.

Linear regression is an important special case where the marginal likelihood is available in closed form for suitable prior distributions. The calculation of the marginal likelihood essentially reduces to solving a least squares problem. For example, with the default prior proposed by Fernández, Ley, and Steel (2001) the marginal likelihood is given by

$$m\left(\mathbf{y}\middle|\,\mathcal{M}\right) \propto (c+1)^{-(k+1)/2}\left(\frac{c\cdot \mathrm{RSS}+\mathrm{TSS}}{c+1}\right)^{-T/2}, \tag{5}$$

where RSS is the residual sum of squares from a least squares fit and TSS the (corrected) total sum of squares and c is a tuning constant in the prior.

The simplified version B of Algorithm 1 applies in this case, which eases the computational burden considerably, but computational efficiency is still important. In many cases it might be needed to run the chain for 5 million steps or more. Scaling up our benchmark this can take as much as 4 hours when using a standard OLS routine to solve the OLS problem. But this is, as many authors have noted, obviously wasteful since the proposed model is a minor variation on the current model. Algorithms that update the current model rather than redoing all the calculations can reduce the time needed by as much as 85% compared to otherwise optimized algorithms. While promising increased speed, there is a potential cost in the form of loss of numerical accuracy due to accumulated round-off errors. We evaluate the speed and accuracy of a number of algorithms for updating OLS estimates when variables are added to or dropped from a model.

It is obvious that the RJMCMC scheme involves small changes between models, with the move types discussed above. This is also to a large extent the case with the more complex updating schemes evaluated in Section 5 and we expect similar savings in computational time. There are clearly similar benefits when calculating the marginal likelihoods by brute force since the models can be enumerated by the binary indicator vector $\boldsymbol{\gamma}$.

3 Solving least squares problems

Consider the linear model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{6}$$

where $\mathbf{y}$ is a $(T \times 1)$ vector, $\mathbf{X}$ is a $(T \times k)$ matrix of explanatory variables, with $T \geq k$, and $\boldsymbol{\beta}$ is a $(k \times 1)$ vector of unknown parameters. A basic computational problem in regression analysis is to obtain the solution to the normal equations

$$\mathbf{X}'\mathbf{X}\boldsymbol{\beta} = \mathbf{X}'\mathbf{y}. \tag{7}$$

The representation of the solution as $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ is mostly a notational or theoretical convenience. In practice this is almost never solved by first computing the inverse and then the product. Instead least squares solvers are based on a factorization $\mathbf{A} = \mathbf{BC}$, where $\mathbf{B}$ and $\mathbf{C}$ are non-singular and easily invertible, for example triangular or orthogonal matrices. The most commonly used factorization is probably the QR factorization, which is considered to have good numerical properties. The Cholesky decomposition is also commonly used, but is in general considered to be more susceptible to round-off errors. Other algorithms that can be used include singular value decomposition and the LU factorization. The latter two are not considered here, because they are difficult to update or are relatively inefficient. We will instead include the Sweep operator in our comparison since it is relatively efficient and eminently suited to calculating successive least squares estimates for models that differ by a few explanatory variables. To our knowledge this is the first time the Sweep operator is considered in this context.

Table 1 gives a general impression of the efficiency of these algorithms as least squares solvers. The table gives the leading terms in the number of floating point operations (flop) needed for the different steps in order to solve for $\hat{\boldsymbol{\beta}}$ and calculate the residual sum of squares. It is clear that there is little difference for moderate size problems and that the advantage of the Cholesky decomposition and the Sweep operator increases with T. The increased saving in computations is due to these algorithms operating on $\mathbf{X}'\mathbf{X}$ rather than $\mathbf{X}$. This is also a great advantage when repeatedly solving related least squares problems since $\mathbf{X}'\mathbf{X}$, $\mathbf{X}'\mathbf{y}$ and $\mathbf{y}'\mathbf{y}$ can be precomputed, and the Cholesky decomposition and the Sweep operator become $O(k^3)$ algorithms, whereas the QR decomposition is $O(Tk^2)$.[2]

In what follows, we review the algorithms and show how they can be used to update least squares solutions when variables are dropped from or added to a model.[3] In addition to theoretical predictions about their numerical efficiency based on flop counts, we also evaluate the efficiency and accuracy of the algorithms in scenarios designed to mimic the sequence of computations in MCMC-based variable selection and BMA exercises. It should be noted that other implementations of the algorithms might be more numerically accurate. In particular, we do not use pivoting with the Cholesky and QR decompositions since this makes updating more complicated.

[2]The QR decomposition can be applied to $\mathbf{X}'\mathbf{X}$, but this requires about 4 times as much calculations as a Cholesky decomposition.

[3]The exposition draws on Golub and van Loan (1996) and the reader is referred to this book for additional background.

Table 1 Order of floating point operations for OLS

Operation	Cholesky	Sweep	QR (Householder)
$\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y}, \mathbf{y}'\mathbf{y}$	$Tk^2/2$	$Tk^2/2$	-
Factor	$k^3/3$	-	$2Tk^2 - 2k^3/3$
Calculate $\mathbf{Q}'\mathbf{y}$	-	-	$4Tk - 2k^2$
Sweep	-	$3k^3/2$	-
Solve for $\hat{\boldsymbol{\beta}}$	k^2	-	k^2
Calculate RSS	$4k + k^2$	-	$2(T - k)$
Overall order	$Tk^2/2 + k^3/3$	$Tk^2/2 + 3k^3/2$	$2Tk^2 - 2k^3/3$

3.1 QR decomposition

The QR decomposition operates directly on the $(T \times k)$ matrix

$$\mathbf{X} = \mathbf{QR} = \mathbf{Q}_1\mathbf{R}_1, \tag{8}$$

decomposing it into a $(T \times T)$ orthogonal matrix $\mathbf{Q} = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix}$ and $\mathbf{R} = \begin{bmatrix}\mathbf{R}_1' & \mathbf{O}'\end{bmatrix}'$, where $\mathbf{R}_1$ is a $(k \times k)$ upper triangular matrix and $\mathbf{O}$ a $((T-k) \times k)$ null matrix. The decomposition $\mathbf{X} = \mathbf{Q}_1\mathbf{R}_1$ is a so-called thin factorization. The normal equations reduce to

$$\begin{aligned}\mathbf{R}_1'\mathbf{R}_1\hat{\boldsymbol{\beta}} &= \mathbf{R}_1'\mathbf{Q}_1'\mathbf{y} \\ \mathbf{R}_1\hat{\boldsymbol{\beta}} &= \mathbf{Q}_1'\mathbf{y},\end{aligned} \tag{9}$$

which is trivial to solve due to the triangularity of $\mathbf{R}_1$. The residual sum of squares can be obtained by summing the squares of the vector $\mathbf{Q}_2'\mathbf{y}$.

The QR decomposition can alternatively be applied on an augmented matrix

$$\mathbf{A} = \begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix} = \mathbf{Q}^*\mathbf{R}^*. \tag{10}$$

$\mathbf{R}_1$ is then the leading $(k \times k)$ submatrix of $\mathbf{R}^*$ and $\mathbf{Q}_1'\mathbf{y}$ can be found in the first k rows of the last $(k+1)$ column of $\mathbf{R}^*$. The square of the last diagonal element of $\mathbf{R}^*$ is the residual sum of squares. In this way the multiplication $\mathbf{Q}'\mathbf{y}$ is not necessary and the calculation time is reduced.

Methods for calculating the QR decomposition include Householder reflections, Givens rotations, and Gram-Schmidt orthogonalization.

3.1.1 QR decomposition by Householder reflections

A Householder reflection is a $(m \times m)$ symmetric, and orthogonal matrix of the form

$$\mathbf{H} = \mathbf{I} - \frac{2}{\mathbf{u}'\mathbf{u}}\mathbf{u}\mathbf{u}', \tag{11}$$

where $\mathbf{u}$ is a $(m \times 1)$ Householder vector defined as

$$\mathbf{u} = \mathbf{x} + \operatorname{sgn}(x_1)\|\mathbf{x}\|_2\,\mathbf{e}_1, \tag{12}$$

where $\mathbf{x}$ is a non-zero $(m \times 1)$ vector, $\mathbf{e}_1$ is the first vector of an identity matrix, $\|\cdot\|_2$ denotes the Euclidean norm, and $\operatorname{sgn}(x_1)$ is the sign of the first element in $\mathbf{x}$.

Using the Householder reflections it is easy to transform a non zero vector

$$\mathbf{x} = \begin{pmatrix} x_1, & x_2, & x_3, & \ldots, & x_m \end{pmatrix}' \tag{13}$$

into a vector

$$\tilde{\mathbf{x}}_{\mathbf{H}} = \begin{pmatrix} \tilde{x}_{H1}, & 0, & 0, & \ldots, & 0 \end{pmatrix}', \tag{14}$$

thus introducing zeros on a grand scale.

Let $\mathbf{H}_1$ be the $(T \times T)$ Householder matrix that sets the elements below the diagonal of the first column of $\mathbf{X}$ to zero. Denote the resulting matrix as $\mathbf{X}_1 = \mathbf{H}_1\mathbf{X}$. Next consider the second column of $\mathbf{X}_1$. The elements below the diagonal can be set to zero by applying a $((T-1) \times (T-1))$ Householder matrix $\widetilde{\mathbf{H}}_2$ to the last $T-1$ rows or, equivalently, premultiplying $\mathbf{X}_1$ by $\mathbf{H}_2 = \operatorname{diag}\left(1, \widetilde{\mathbf{H}}_2\right)$. Continuing in this fashion and noting that $\mathbf{H}_j$ only operates on the lower right $T-j+1$ submatrix from the previous step, we obtain $\mathbf{R}$ as

$$\mathbf{R} = \mathbf{H}_k\mathbf{H}_{k-1}\ldots\mathbf{H}_2\mathbf{H}_1\mathbf{X}, \tag{15}$$

and the orthogonal matrix $\mathbf{Q}$ is given by

$$\mathbf{Q} = \mathbf{H}_1\mathbf{H}_2\ldots\mathbf{H}_{k-1}\mathbf{H}_k. \tag{16}$$

In practice $\mathbf{Q}$ is rarely needed, and $\mathbf{Q}'\mathbf{y}$ can be calculated alongside $\mathbf{R}$ as $\mathbf{Q}'\mathbf{y} = \mathbf{H}_k\mathbf{H}_{k-1}\ldots\mathbf{H}_2\mathbf{H}_1\mathbf{y}$, which is an $O(Tk)$ operation, compared to accumulating $\mathbf{Q}$ and explicitly calculating $\mathbf{Q}'\mathbf{y}$, which are $O(T^2k)$ and $O(T^2)$, respectively. Our algorithm, denoted as House, uses the former approach. It also factors $\mathbf{X}$ rather than $\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix}$, since the latter would complicate the updating procedure discussed next. The Householder routines are based on the LINPACK subroutines DQRDC and DQRSL. See Dongarra, Bunch, Moler, and Stewart (1979) for details.

Updating QR decomposition using Householder reflections

The vectors $\mathbf{u}$ needed to form $\mathbf{H}_j$ can be stored efficiently[4] and the Householder reflections are easily recreated. This, together with the fact that a Householder reflection requires much less work than a full matrix product, makes a simple algorithm for adding or deleting columns from $\mathbf{X}$ available.

A column $\mathbf{z}$ is simply added to $\mathbf{X}$ by applying the Householder reflections $\mathbf{H}_1, \ldots, \mathbf{H}_k$ in turn, which forms $\mathbf{Q}'\mathbf{z}$. Next, elements $k+2$ to T of $\mathbf{Q}'\mathbf{z}$ are zeroed with $\mathbf{H}_{k+1}$, and the first $k+1$ elements of the result are appended to $\mathbf{R}_1$ as column $k+1$ in the upper triangle of $\mathbf{X}$. The vector $\mathbf{u}$ defining $\mathbf{H}_{k+1}$ is stored below the diagonal and the previously stored $\mathbf{Q}'\mathbf{y}$ is premultiplied with $\mathbf{H}_{k+1}$.

To remove column j, $1 \le j \le k$, from $\mathbf{X}$, the reflections for columns $i = k, k-1, \ldots, j$ are first undone by premultiplying the previously saved $\mathbf{Q}'\mathbf{y}$ and columns $i+1$ to k of $\mathbf{R}_1$ with $\mathbf{H}_i$ and restoring the elements on or below the diagonal of column i of $\mathbf{X}$ from the saved $\mathbf{u}$ if $i > j$. Columns $j+1$ to k are then shifted left and new Householder matrices $\mathbf{H}_j, \ldots, \mathbf{H}_{k-1}$ are used to zero the elements below the diagonal and update $\mathbf{Q}'\mathbf{y}$.

A swap of variables is implemented by first removing the column in question and then adding the new variable as the last column k.

[4] In practice, $\mathbf{R}_1$ overwrites the upper triangle of $\mathbf{X}$, the last $T-j$ elements of $\mathbf{u}$ are stored below the diagonal in column j and the first element in an auxiliary vector.

Table 2 Order of floating point operations for OLS updates

	Add	Drop (average)	Swap
HouseUp	$4Tk - k^2$	$4Tk^2/3 - 2k^3/3$	$4Tk^2/3 - 2k^3/3$
GGSUp	$4Tk$	$3Tk + k^2$	$11Tk + k^2$
Cholesky	$k^3/3 + 3k^2/2$	$k^3/3 - k^2/3$	$k^3/3 + k^2$
Sweep	$3N^2/2$	$3N^2/2$	$3N^2$

The leading terms of the flop count for this algorithm, denoted as HouseUp, is given in Table 4.2. While this algorithm provides an efficient way of adding a variable to the model it is quite expensive to remove a variable. A much more efficient way of dropping variables is available using Givens rotations, which can be used to selectively zero elements in a matrix.

3.1.2 QR decomposition by Givens rotations

The Givens rotation method, also known as Jacobi rotations, is defined by the $(m \times m)$ matrix $\mathbf{G}$

$$\mathbf{G}(i,k,\phi) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \\ \end{matrix}, \tag{17}$$
$$\begin{matrix} & & i & & k & & \end{matrix}$$

where $c = \cos(\phi)$, and $s = \sin(\phi)$ for some angle ϕ. Premultiplication by $\mathbf{G}(i,k,\phi)'$ amounts to a counterclockwise rotation of ϕ radians in the (i,k) coordinate plane. By choosing ϕ to satisfy

$$\cos(\phi) = \frac{x_i}{\|\mathbf{x}\|_2} = \frac{x_i}{\sqrt{x_i^2 + x_k^2}}, \tag{18}$$

$$\sin(\phi) = \frac{x_k}{\|\mathbf{x}\|_2} = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}, \tag{19}$$

for a $(m \times 1)$ vector $\mathbf{x}$, the elements of $\tilde{\mathbf{x}} = \mathbf{G}(i,k,\phi)'\mathbf{x}$ can be obtained as

$$\tilde{x}_j = \begin{cases} cx_i - sx_k & j = i \\ 0 & j = k \\ x_j & j \neq i,k. \end{cases} \tag{20}$$

That is, element k of $\mathbf{x}$ is zeroed by applying the Givens rotation matrix. In practice the rotations are typically applied to rows i and $i+1$, and we refer to a Givens rotation matrix that zeroes element $i+1$ of column j in a matrix as $\mathbf{G}_{i,j}$. Note that the product $\mathbf{G}'_{i,j}\mathbf{X}$ only operates on rows i and $i+1$ of $\mathbf{X}$.

The QR factorization of $\mathbf{X}$ can be calculated by applying a series of Givens rotations. First, the $T-1$ elements below the diagonal of column 1 are zeroed with $\mathbf{G}_{T-1,1}$ through $\mathbf{G}_{1,1}$ yielding

$$\mathbf{X}_1 = \mathbf{G}'_{1,1} \dots \mathbf{G}'_{T-1,1}\mathbf{X}. \tag{21}$$

The $T-2$ elements below the diagonal in the second column of the matrix $\mathbf{X}_1$ are zeroed by applying $\mathbf{G}_{T-1,2}$ through $\mathbf{G}_{2,2}$ and so on for the remaining columns. This yields

$$\mathbf{R} = \mathbf{G}'_{k,k} \dots \mathbf{G}'_{T-1,k}\mathbf{G}'_{k-1,k-1} \dots \mathbf{G}'_{T-1,k-1}\mathbf{G}'_{k-2,k-2} \dots \mathbf{G}'_{T-1,1}\mathbf{X} \tag{22}$$

and

$$\mathbf{Q} = \mathbf{G}_{T-1,1}\mathbf{G}_{T-2,1} \dots \mathbf{G}_{1,1}\mathbf{G}_{T-1,2} \dots \mathbf{G}_{2,2} \dots \mathbf{G}_{k,k}. \tag{23}$$

This factorization algorithm is $O\left(k^2\left(3T-k\right)\right)$ and less efficient than using Householder matrices. A variation on Givens rotations known as Fast Givens or Modified Givens is of the same order as using Householder matrices, but the maintenance of a vector of scaling constants as well as occasional rescaling is required. Hanson and Hopkins (2004) evaluate the relative performance of Givens and Modified Givens rotations, and find that it is impossible to predict which will perform better on a given hardware/compiler combination.

Deleting a column using Givens rotations

Assume that we have the thin QR factorization

$$\mathbf{X} = \mathbf{Q}_1\mathbf{R}_1, \tag{24}$$

and delete column i in the matrix $\mathbf{X}$. Denote this new matrix as $\widetilde{\mathbf{X}}$. Deleting the i-th column of $\mathbf{R}_1$ gives

$$\widetilde{\mathbf{R}}_1 = \mathbf{Q}'_1\widetilde{\mathbf{X}} = \begin{bmatrix} \mathbf{R}_{1,1} & \mathbf{T}_1 \\ \mathbf{0} & \mathbf{T}_2 \end{bmatrix}, \tag{25}$$

where $\mathbf{R}_{1,1}$ is a $((i-1)\times(i-1))$ upper triangular matrix, $\mathbf{T}_1$ is a rectangular $((i-1)\times(k-i))$ matrix and $\mathbf{T}_2$ is a $(k-i+1)\times(k-i)$ upper Hessenberg matrix[5]. The unwanted subdiagonal elements $\tilde{r}_{i+1,i}, \dots, \tilde{r}_{k,k-1}$, present in $\mathbf{T}_2$, can be zeroed by applying a sequence of Givens rotations to $\widetilde{\mathbf{R}}_1$ and form the new

$$\mathbf{R}_1 = \mathbf{G}'_{k-1,k-1}\mathbf{G}'_{k-2,k-2} \cdots \mathbf{G}'_{i,i}\widetilde{\mathbf{R}}_1.$$

$\mathbf{Q}'\mathbf{y}$ is updated similarly by performing the same sequence of Givens rotations and $\mathbf{Q}$ can, if needed, be accumulated by postmultiplying with the Givens matrices.

These are $O\left(3Tk+k^2\right)$ operations and considerably more efficient than the Householder-based algorithm for deleting a column. This algorithm for deleting columns is, however, difficult to combine with the quite efficient approach of adding columns using Householder reflections.

[5] In an upper Hessenberg matrix all elements below the first subdiagonal are equal to zero.

Adding a column using Givens rotations

The standard textbook treatment of algorithms for adding a column to $\mathbf{X}$ assumes that the full decomposition $\mathbf{X} = \mathbf{QR}$ is available. Inserting a column $\mathbf{z}$ at position j of $\mathbf{X}$ gives

$$\mathbf{X}^* = \begin{pmatrix} \mathbf{x}_1, & \ldots, & \mathbf{x}_{j-1}, & \mathbf{z}, & \mathbf{x}_j, & \ldots, & \mathbf{x}_k \end{pmatrix}. \tag{26}$$

Premultiplying by $\mathbf{Q}'$ produces

$$\mathbf{Q}'\mathbf{X}^* = \begin{pmatrix} \mathbf{Q}'\mathbf{x}_1, & \ldots, & \mathbf{Q}'\mathbf{x}_{j-1}, & \mathbf{Q}'\mathbf{z}, & \mathbf{Q}'\mathbf{x}_j, & \ldots, & \mathbf{Q}'\mathbf{x}_N \end{pmatrix} = \mathbf{R}^*, \tag{27}$$

which is $\mathbf{R}$ with $\mathbf{Q}'\mathbf{z}$ inserted at position j. $\mathbf{R}^*$ is upper triangular except for the j-th column. Applying Givens rotations to zero the elements below the diagonal in column j reduces $\mathbf{R}^*$ to upper Hessenberg form. The unwanted elements $r^*_{j+2,j+1}, \ldots, r^*_{k+2,k+1}$ are then zeroed with a series of Givens rotations. $\mathbf{Q}'\mathbf{y}$ (and $\mathbf{Q}$) can be updated in parallel using the same sequence of Givens rotations. This algorithm is efficient if it is cheap to form $\mathbf{Q}'\mathbf{z}$.

There is a substantial penalty for using a Householder reflection to zero the elements below the diagonal in column j since this turns *all* the elements below the diagonal in columns $j+1, \ldots, k+1$ into non-zero elements, which in turn will have to be zeroed. There is, of course, no penalty if $\mathbf{z}$ is appended to $\mathbf{X}$ as column $k+1$ rather than inserting it as an interior column $j \leq k$. The Householder updating algorithm takes advantage of this and forms $\mathbf{Q}'\mathbf{z}$ directly by applying a small number of Householder reflections, rather than maintaining the $(T \times T)$ matrix $\mathbf{Q}$ and calculating $\mathbf{Q}'\mathbf{z}$ as a matrix product. A similar level of efficiency can be achieved with Givens rotations if a limited number of rotations are needed. However, the use of these algorithms in the settings outlined in the previous section causes the number of rotations to grow without bound with the number of updates of the least squares solution. We are thus reduced to maintaining a full $\mathbf{Q}$ matrix and calculating the matrix product $\mathbf{Q}'\mathbf{z}$, which turns the step of adding a column into an $O(T^2 k)$ operation.

There is thus a trade off. The Householder updating algorithm limits the number of reflections by undoing them in the delete step, which makes an effective add step possible. The Givens-based delete step is much more efficient than the Householder delete step, but the number of rotations grows without bounds, which leads to an inefficient add step.

3.1.3 Gram-Schmidt orthogonalization

Gram-Schmidt orthogonalization is essentially an algorithm for appending columns, which has several advantages in addition to being possible to combine with the efficient Givens-based delete step. It is sufficient to maintain the $(T \times k)$ matrix $\mathbf{Q}_1$ instead of a full $\mathbf{Q}$ since only $\mathbf{Q}_1'\mathbf{z}$ and not $\mathbf{Q}'\mathbf{z}$ is needed for the update. The use of Gram-Schmidt orthogonalization as a tool for adding columns is due to Daniel, Gragg, Kaufman, and Stewart (1976), and our implementation is based on the Fortran code in Reichel and Gragg (1990).

Suppose we have the thin QR factorization $\mathbf{X} = \mathbf{Q}_1\mathbf{R}_1$ and add a column $\mathbf{z}$ to $\mathbf{X}$ to form (26). We then have

$$\mathbf{X}^* = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{z} \end{bmatrix} \begin{bmatrix} \mathbf{R}_{1,1} & \mathbf{0} & \mathbf{R}_{1,2} \\ \mathbf{0}' & 1 & \mathbf{0}' \end{bmatrix}. \tag{28}$$

Next, use the Gram-Schmidt orthogonalization procedure to find a $(T \times 1)$ vector $\mathbf{q}$, a $(k \times 1)$ vector $\mathbf{r}$, and a scalar ρ that satisfy

$$\begin{bmatrix} \mathbf{Q}_1 & \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{q} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{r} \\ \mathbf{0}' & \rho \end{bmatrix}, \tag{29}$$

under the conditions that $\mathbf{Q}_1'\mathbf{q} = \mathbf{0}$ and $\mathbf{q}'\mathbf{q} = 1$. The last column on the right hand side of (29) equals $\mathbf{z} = \mathbf{Q}_1\mathbf{r} + \rho\mathbf{q}$, which premultiplied by $\mathbf{Q}_1'$, gives the vector

$$\mathbf{r} = \mathbf{Q}_1'\mathbf{z}. \tag{30}$$

Setting $\mathbf{z}^* = \mathbf{z} - \mathbf{Q}_1\mathbf{r} = \rho\mathbf{q}$ and using $\mathbf{q}'\mathbf{q} = 1$ we have

$$\rho = \mathbf{z}^{*\prime}\mathbf{z}^*, \tag{31}$$

$$\mathbf{q} = \frac{1}{\rho}\mathbf{z}^*. \tag{32}$$

Combining (28) and (29) yields

$$\mathbf{X}^* = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{q} \end{bmatrix} \begin{bmatrix} \mathbf{R}_{1,1} & \mathbf{r} & \mathbf{R}_{1,2} \\ \mathbf{0}' & \rho & \mathbf{0}' \end{bmatrix} = \mathbf{Q}_1^*\mathbf{R}_1^* \tag{33}$$

and $\mathbf{R}_1^*$ can be reduced to upper triangular form by a series of Givens rotations. We always append $\mathbf{z}$ as the last, $k+1$, column of $\mathbf{X}$, thus no rotations are needed and the algorithm is $O(4Tk)$ in this case.

It is possible that the Gram-Schmidt procedure (30) yields a solution $\mathbf{q}$ that is not orthogonal to $\mathbf{Q}$. In this case reorthogonalization can be used, by simply setting $\mathbf{z} = \mathbf{z}^*$ and applying (30) - (32) again. Daniel, Gragg, Kaufman, and Stewart (1976) provide an error analysis indicating that the Gram-Schmidt procedure with reorthogonalization has good numerical properties.

Our algorithm, abbreviated as GGSUp, combines the Givens rotation-based procedure for deleting columns and the Gram-Schmidt procedure for adding columns. A swap of columns is achieved by first removing the column in question and then adding the new column as the last column k. Since only $\mathbf{Q}_1$ is maintained the residuals must be calculated explicitly after solving the least squares problem in order to obtain the residual sum of squares. In addition, $\mathbf{Q}_1'\mathbf{y}$ is calculated directly as a matrix product rather than continuously updating the product.

3.2 Cholesky decomposition

The Cholesky factorizations decomposes a $(k \times k)$ symmetric, positive definite matrix $\mathbf{A} = \mathbf{X}'\mathbf{X}$ into a product of a lower triangular matrix $\mathbf{L}$ and its transpose $\mathbf{L}'$,

$$\mathbf{A} = \mathbf{X}'\mathbf{X} = \mathbf{L}\mathbf{L}'. \tag{34}$$

The solution to the normal equations is then obtained by solving two triangular equations systems

$$\mathbf{L}\boldsymbol{\xi} = \mathbf{X}'\mathbf{y}, \tag{35}$$

$$\mathbf{L}'\boldsymbol{\beta} = \boldsymbol{\xi}. \tag{36}$$

The residual sum of squares is calculated using the identity $\text{RSS} = \mathbf{y}'\mathbf{y} - \mathbf{y}'\mathbf{X}\left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{y}$, where $\mathbf{y}'\mathbf{X}\left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{y}$ is obtained as a by-product of solving the equations system (36).

As already noted, a great benefit of the Cholesky decomposition is that $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}'\mathbf{y}$ can be precomputed for the full $(T \times N)$ data matrix and the algorithm is then independent of T, when repeatedly solving least squares problems.

The Cholesky routines are based on the Press, Teukolsky, Vetterling, and Flannery (1992) subroutines CHOLDC and CHOLSL.

3.2.1 Updating the Cholesky decomposition

Assume that the Cholesky decomposition (34) has been obtained. When a variable is added to a model, the modified matrix and its decomposition are given as

$$\mathbf{A}^* = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{a}_j & \mathbf{A}_{12} \\ \mathbf{a}_j' & a_{jj} & \boldsymbol{\alpha}_j' \\ \mathbf{A}_{21} & \boldsymbol{\alpha}_j & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{11} & 0 & 0 \\ \mathbf{l}_j' & l_{jj} & 0 \\ \mathbf{L}_{21} & \boldsymbol{\lambda}_j & \mathbf{L}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{11}' & \mathbf{l}_j & \mathbf{L}_{21}' \\ 0 & l_{jj} & \boldsymbol{\lambda}_j' \\ 0 & 0 & \mathbf{L}_{22}' \end{bmatrix}. \tag{37}$$

The updated decomposition is found first by solving

$$\mathbf{l}_j = \mathbf{L}_{11}^{-1}\mathbf{a}_j, \tag{38}$$

$$l_{jj} = \sqrt{a_{jj} - \mathbf{l}_j'\mathbf{l}_j}, \tag{39}$$

$$\boldsymbol{\lambda}_j = \frac{1}{l_{jj}}\left(\boldsymbol{\alpha}_j - \mathbf{L}_{21}l_{jj}\right), \tag{40}$$

and then decomposing the lower right corner $\mathbf{A}_{22}$ as usual, i.e. by using expressions (39)-(40). This is the inner-product version of the factorization. See Golub and van Loan (1996) for the outer-product alternative. Deleting a column j from a model is equivalent to decomposing the lower right corner $\mathbf{A}_{22}$. When swapping two variables, the variable added to the model simply takes the position of the variable being removed and the update as described above is carried out. When only the Add step is performed, the corresponding products are always appended as the last row and column, respectively.

3.3 The reversible sweep operator

The sweep operator was designed by Beaton (1964) as a tool for inverting symmetric matrices. As Goodnight (1979) points out:

> "The importance of the sweep operator in statistical computing is not so much that it is an inversion technique, but rather that it is a conceptual tool to understanding the least squares process."

Goodnight also provides a review of its use, including: ordinary least squares, two-stage and three-stage least squares, nonlinear least squares, multivariate analysis of variance, regressions by leaps and bounds, stepwise regression, and partial correlation.

The details of the Sweep operator are given as Algorithm 2. Applying the sweep operator to the first N columns of the data matrix

$$\mathbf{A} = \begin{bmatrix} \mathbf{X}'\mathbf{X} & \mathbf{X}'\mathbf{y} \\ \mathbf{y}'\mathbf{X} & \mathbf{y}'\mathbf{y} \end{bmatrix} \tag{41}$$

Algorithm 2 The reversible upper triangular sweep operator

Define an auxiliary vector $\mathbf{v}$ with initial values $\mathbf{v} = (v_1, \ldots, v_N) = (1, 1, \ldots, 1)$ indicating that all variables are unswept. If variable i is swept then we set $v_i = -v_i$. Choose a tolerance $\delta > 0$ to guard against numerical problems caused by near non-singularity. The following steps sweep variable k.

1. Return if $a_{kk} \leq \delta$ and $v_k = 1$.
2. Let $\eta = a_{kk}$.
3. Set
$$\zeta = \begin{cases} a_{kj}\,\eta^{-1}, & \text{for each } j = 1, 2, \ldots, k-1, \\ v_j\,v_k\,a_{jk}\,\eta^{-1}, & \text{for each } j = k+1, \ldots, N. \end{cases}$$
4. Set
$$\xi = \begin{cases} a_{ik}, & \text{for each } i = j, j+1 \ldots, k-1, \\ v_i\,v_k\,a_{ki}, & \text{for each } i = k+1, \ldots, N. \end{cases}$$
5. Calculate
$$a_{ij} = a_{ij} - \zeta\xi.$$
6. Set
$$\begin{aligned} a_{kj} &= -a_{kj}\,\eta^{-1}, && \text{for each } j = 1, \ldots, k, \\ a_{ik} &= a_{ik}\,\eta^{-1}, && \text{for each } i = k, \ldots, N. \end{aligned}$$
7. Finally, set
$$\begin{aligned} a_{kk} &= \eta^{-1}, \\ v_k &= -v_k. \end{aligned}$$

---

results in a matrix

$$\begin{aligned} \mathbf{A}^S &= \begin{bmatrix} (\mathbf{X}'\mathbf{X})^{-1} & (\mathbf{X}'\mathbf{X})^{-1}\,\mathbf{X}'\mathbf{y} \\ -\mathbf{y}'\mathbf{X}\,(\mathbf{X}'\mathbf{X})^{-1} & \mathbf{y}'\mathbf{y} - \mathbf{y}'\mathbf{X}\,(\mathbf{X}'\mathbf{X})^{-1}\,\mathbf{X}'\mathbf{y} \end{bmatrix} \\ &= \begin{bmatrix} (\mathbf{X}'\mathbf{X})^{-1} & \hat{\boldsymbol{\beta}} \\ -\hat{\boldsymbol{\beta}}' & \text{RSS} \end{bmatrix}, \end{aligned} \tag{42}$$

since $\text{RSS} = \mathbf{y}'\mathbf{y} - \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y}$.

The estimates of $\boldsymbol{\beta}$, the residual sum of squares and the inverse of $\mathbf{X}'\mathbf{X}$ can therefore be obtained by using simple computational operations. Note that $\mathbf{y}$ and the variables in $\mathbf{X}$ used to form the matrix $\mathbf{A}$ are assumed to be centred. The matrix $\mathbf{X}'\mathbf{X}$ is thus the sums of squares and cross product matrix (SSCP). The $((N+1) \times (N+1))$ matrix $\mathbf{A}$ is also a SSCP matrix, with element $a_{N+1,N+1}$ being the corrected total sum of squares.

The time taken to perform the sweep operations may be reduced by taking into account the symmetry of the matrix (42). The elements below the main diagonal may be

constructed as

$$a_{ij} = \begin{cases} -a_{ji} & \text{if variable } i \text{ or variable } j \text{ has been swept,} \\ a_{ji} & \text{otherwise.} \end{cases} \tag{43}$$

Two properties of the sweep operator are extremely useful when repeatedly solving closely related least squares problems:

1. It is not necessary to perform the sweep of the pivots in any particular order to obtain the inverse,

2. Sweep operations are reversible, that is, the status of a matrix that existed prior to a sweep on a particular column can be reestablished by repeating a sweep operation on the same column. The reverse sweeps do not need to be performed in the same order in which the forward steps were originally performed.

In this context the matrix (41) is formed using the full $(T \times N)$ data matrix and the algorithm is $O(N^2)$ and independent of both T and k.

To guarantee the reversibility of the Algorithm 2 in the case when dependencies between explanatory variables exist, Goodnight (1979) suggests to sweep the diagonal element a_{kk} only if its value is greater than some tolerance, δ, defined as

$$\delta = \begin{cases} \delta_0 \cdot \text{TSS}, & \text{if TSS} > 0 \\ \delta_0, & \text{otherwise,} \end{cases} \tag{44}$$

with $\delta_0 \in [10^{-8}, 10^{-12}]$. In this way the critical information that defines variable's dependency is preserved.

4 Numerical accuracy and computational efficiency

This section examines the numerical accuracy and efficiency of the algorithms in a setting designed to closely resemble the sequence of computations encountered in Bayesian variable selection and model averaging exercises. To summarize, the algorithms[6] included in the evaluation are:

OLS	the IMSL subroutine DRLSE
House	QR decomposition using Householder reflections
HouseUp	Updating the QR decomposition using Householder reflections
GGSUp	Updating the QR decomposition using Givens rotations and Gram-Schmidt orthogonalization
Chol	Cholesky decomposition
CholUp	Updating the Cholesky decomposition
Sweep	Updating the least squares solution using the Sweep operator

While Tables 1 and 2 give a good indication of the speed one can expect from the different algorithms a rough flop count is only part of the story. Other issues such as memory access patterns are at least as important and much more difficult to evaluate

[6] All algorithms are programmed in Fortran 95 and compiled with Compaq Visual Fortran 6.6C using default compiler settings. The code is available on request from the authors.

using theoretical arguments. An empirical evaluation of the relative performance is thus needed.

Numerical accuracy is, of course, of utmost importance, it is of no use to have a fast algorithm if it produces misleading results. When evaluating the numerical accuracy we use the IMSL (Visual Numerics, Inc. (1994)) subroutine DRLSE for OLS as the benchmark. This routine performs an orthogonal reduction of the matrix $\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix}$ to upper triangular form using Fast Givens transformations.

We generate datasets with $N = 25, 50$ and 100 variables for three different sample sizes $T = 100, 250$ and 400.

The design of the experiment is based on Fernández, Ley, and Steel (2001). A matrix of 15 predictors $\mathbf{X}_{(T\times 15)}$ is generated, where the first 10 random variables, $\mathbf{x}_1, \ldots, \mathbf{x}_{10}$, are i.i.d. standard normal and the additional five variables are constructed according to

$$(\mathbf{x}_{11}, \ldots, \mathbf{x}_{15}) = (\mathbf{x}_1, \ldots, \mathbf{x}_5) \begin{pmatrix} 0.3 & 0.5 & 0.7 & 0.9 & 1.1 \end{pmatrix}' \boldsymbol{\iota} + \mathbf{E}, \tag{45}$$

where $\boldsymbol{\iota}$ is a (1×5) vector of ones and $\mathbf{E}$ is a $(T \times 5)$ matrix of i.i.d. standard normals. This produces moderate multicollinearity between the first five and the last five predictors. The dependent variable is generated as

$$y_t = 4 + 2x_{1,t} - x_{5,t} + 1.5x_{7,t} + x_{11,t} + 0.5x_{13,t} + \sigma\varepsilon_t, \tag{46}$$

where the disturbances ε_t are i.i.d. standard normal and $\sigma = 2.5$. To obtain the desired dataset size the remaining $N - 15$ variables are generated as i.i.d. standard normal.

In all cases the explanatory variables are centred and standardized prior to running the experiments. For each dataset we attempt to approximate the behaviour of a RJMCMC scheme like Algorithm 1 when the true model size is $k = 5, 10, 15$ or 20 explanatory variables. A specific average model size over the pseudo-MCMC run is achieved by controlling the manner in which the variables are added to or removed from the model. This process starts at a selected model size k, and follows a sequence of steps: swap, add, swap, drop, swap, drop, swap, add, swap, add, swap, drop, etc. For the smallest model size this produces the following sequence of model sizes: 5,6,6,5,5,4,4,5,5,6,6,5, ... In this manner 25% of the MCMC steps add a variable, 25% drop a variable and remaining 50% are swapping a variable. The 'Markov chain' is run for 50 000 steps and all the algorithms considered visit the same set of models in the same order.

4.1 Results

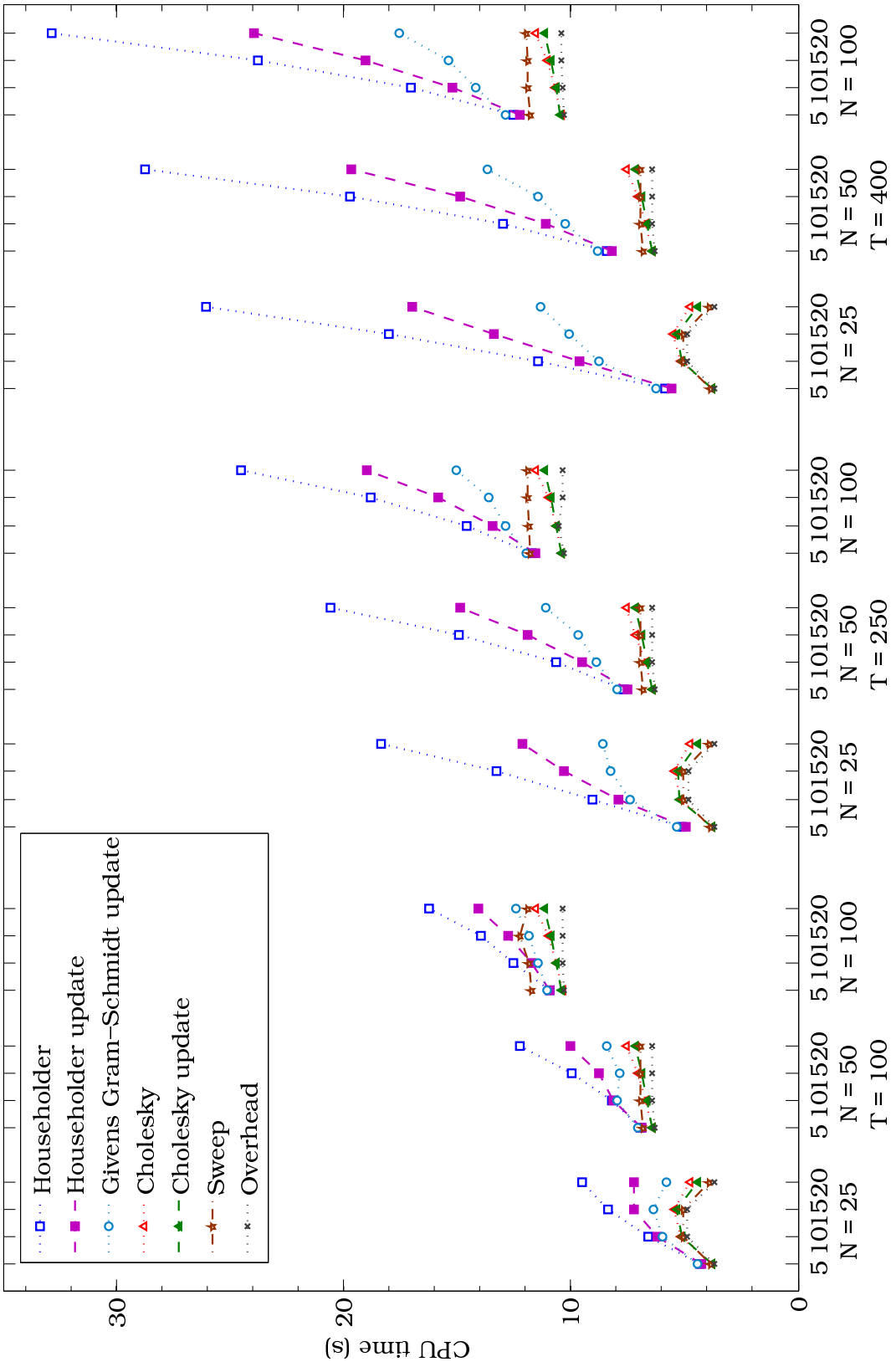
4.1.1 Speed

For each combination of the number of potential explanatory variables, N, number of observations, T, and model size, k, the timing experiments were run 5 times and the average CPU time calculated.[7]

The timings include overhead for generating the proposed models and keeping track of the number of visits to a particular model and are thus representative of the actual time requirements for a RJMCMC algorithm. In order to isolate the contribution of the least squares algorithms we also report on the CPU time for the overhead. It should be noted that the time needed to solve the least squares problem is only a small fraction of the total time for the fastest algorithms.

[7]The experiments were run on an IBM desktop PC running Windows XP SP2 with a Pentium 4 530 processor (3 GHz), 800 MHz memory buss, 1 MB L2 cache and Intel 915G chipset.

Figure 1 CPU time (approximation in seconds) for 50 000 steps of a Markov chain for different combinations of k, N and T.



Tables B.1 - B.3 report the times for all the algorithms. Figure 1 shows the CPU time in seconds for all algorithms except OLS, which by far is the slowest algorithm for all combinations of k, N and T. It is not surprising that OLS is the slowest algorithm, since it is designed for general use with checks of the data, treatment of missing values etc. that is absent from the other algorithms. In general, increasing the dataset size N has the same effect on all algorithms. Except for the Sweep operator the increase in CPU time can be traced to the increased overhead that is due to the larger number of possible models, and that more distinct models are proposed (and visited) with larger N. For the Sweep operator we see a larger increase in the CPU time than for the other algorithms, reflecting the fact that it always operates on a $(N \times N)$ matrix irrespective of the current model size.

Increasing model size does not influence the Sweep operator and only slightly the Cholesky decomposition and Cholesky update. The concave form for the dataset size $N = 25$, showing dependence on k for these 3 methods, is caused by the fact that there are fewer models in the model set for $k = 5$ and $k = 20$ and thus less overhead than for the remaining two model sizes.

The House, the HouseUp, and GGSUp algorithms all depend on k with House the most sensitive and GGSUp the least affected. This is in rough agreement with the predictions from Tables 1 and 2.

The Cholesky decomposition, its update, and the Sweep operator are naturally not affected by the sample size T since they operate on the matrix $\mathbf{X}'\mathbf{X}$. Of the remaining algorithms GGSUp is the least affected followed by HouseUp and House. Again this is as predicted by the flop counts in Tables 1 and 2.

Clearly the Sweep operator, Cholesky, and the Cholesky update algorithms are the most efficient. The Sweep operator is preferred if the number of variables, N, is not too large relative to the average model size, k. The Cholesky update is faster than Cholesky except for the smallest model size, $k = 5$.

4.1.2 Accuracy

For each of the algorithms the values of the estimated beta parameters and the value of the residual sum of squares are recorded at every $s = 100$ steps, giving in total 500 control points. Tables 3 - 4 report the average number of correct significant digits relative to the benchmark algorithm, OLS, for RSS and $\hat{\boldsymbol{\beta}}$, based on the 500 control points. For $\hat{\boldsymbol{\beta}}$ this is an average over both the number of parameters in the model at a particular control point and the control points. That is, the average number of correct digits for $\hat{\boldsymbol{\beta}}$ is given by

$$\zeta_{\hat{\boldsymbol{\beta}}} = \frac{1}{500} \sum_{j=1}^{500} \frac{1}{k_j} \sum_{i=1}^{k_j} \left| \log \left| \frac{\hat{\beta}_i - \hat{\beta}_i^{OLS}}{\hat{\beta}_i^{OLS}} \right| \right|. \tag{47}$$

With the increasing sample size T, the number of correct digits is decreasing for all algorithms. The algorithm that is closest to the benchmark is the Cholesky decomposition and its update. The Givens Gram-Schmidt algorithm, however, has the highest number of correct digits for the model parameters. The worst performing algorithm is the Sweep operator with a difference of around one digit.

Figures A.1 - A.3 provide an example of the relative numerical accuracy for RSS at each of the control points for dataset with $N = 50$ variables and average model size $k = 10$. All the algorithms, except the Sweep operator, are oscillating round zero with a constant

Table 3 Average number of correct significant digits for RSS.

N	k	House	HouseUp	GGSUp	Chol	CholUp	Sweep
T = 100							
	5	15.58	15.59	15.57	15.69	15.69	14.78
25	*10*	15.57	15.57	15.55	15.66	15.66	14.30
	15	15.56	15.58	15.58	15.66	15.66	14.27
	20	15.59	15.55	15.56	15.68	15.67	13.98
	5	15.57	15.56	15.58	15.67	15.67	14.14
50	*10*	15.58	15.57	15.53	15.68	15.68	14.73
	15	15.56	15.55	15.58	15.67	15.68	14.34
	20	15.57	15.54	15.58	15.66	15.66	14.06
	5	15.57	15.56	15.56	15.67	15.67	14.27
100	*10*	15.57	15.56	15.55	15.67	15.67	14.66
	15	15.56	15.56	15.56	15.65	15.65	13.94
	20	15.57	15.56	15.56	15.64	15.63	14.41
T = 250							
	5	15.49	15.49	15.49	15.46	15.46	14.63
25	*10*	15.47	15.47	15.46	15.39	15.39	14.27
	15	15.48	15.44	15.47	15.39	15.39	14.08
	20	15.49	15.46	15.48	15.43	15.43	14.45
	5	15.50	15.45	15.46	15.43	15.42	14.40
50	*10*	15.46	15.45	15.44	15.40	15.40	14.32
	15	15.46	15.51	15.45	15.37	15.37	14.73
	20	15.49	15.49	15.45	15.41	15.41	14.50
	5	15.48	15.51	15.47	15.42	15.42	14.84
100	*10*	15.49	15.49	15.45	15.42	15.42	14.33
	15	15.45	15.47	15.45	15.41	15.41	14.07
	20	15.48	15.47	15.46	15.40	15.41	14.54
T = 400							
	5	15.38	15.35	15.34	15.49	15.49	14.13
25	*10*	15.33	15.35	15.33	15.47	15.47	14.76
	15	15.36	15.36	15.35	15.49	15.49	13.99
	20	15.33	15.33	15.32	15.44	15.44	14.72
	5	15.31	15.33	15.34	15.43	15.43	14.77
50	*10*	15.34	15.34	15.33	15.46	15.46	14.23
	15	15.35	15.33	15.37	15.47	15.47	14.18
	20	15.34	15.37	15.33	15.49	15.50	14.74
	5	15.32	15.32	15.33	15.47	15.47	14.49
100	*10*	15.34	15.32	15.37	15.46	15.46	14.30
	15	15.34	15.31	15.32	15.42	15.43	14.17
	20	15.37	15.35	15.34	15.49	15.48	14.62
Average		15.46	15.46	15.45	15.51	15.51	14.39

Table 4 Average number of correct significant digits for estimated model parameters.

N	k	House	HouseUp	GGSUp	Chol	CholUp	Sweep
T = 100							
	5	15.05	14.99	15.13	15.09	15.11	14.12
25	*10*	14.98	14.85	15.05	15.00	15.00	13.99
	15	14.94	14.73	14.99	14.88	14.88	14.01
	20	14.88	14.64	14.94	14.78	14.77	13.95
	5	15.06	15.02	15.12	15.13	15.13	14.02
50	*10*	14.99	14.91	15.07	15.07	15.08	13.96
	15	14.95	14.79	15.01	15.01	15.01	13.90
	20	14.92	14.71	14.97	14.96	14.96	13.94
	5	15.06	15.00	15.12	15.12	15.13	14.08
100	*10*	15.02	14.93	15.08	15.08	15.09	13.94
	15	14.97	14.81	15.01	15.02	15.02	13.89
	20	14.95	14.73	14.97	14.98	14.98	13.87
T = 250							
	5	14.77	14.73	14.83	14.83	14.83	14.06
25	*10*	14.70	14.60	14.79	14.75	14.74	14.00
	15	14.70	14.53	14.78	14.70	14.69	13.98
	20	14.68	14.43	14.76	14.60	14.59	13.92
	5	14.89	14.83	14.94	14.93	14.93	14.16
50	*10*	14.85	14.77	14.92	14.92	14.92	14.13
	15	14.83	14.72	14.89	14.87	14.87	14.02
	20	14.82	14.66	14.89	14.85	14.85	14.06
	5	14.89	14.87	14.97	14.97	14.97	14.19
100	*10*	14.87	14.81	14.93	14.94	14.94	14.11
	15	14.85	14.75	14.91	14.91	14.91	14.05
	20	14.83	14.70	14.90	14.90	14.90	14.03
T = 400							
	5	14.80	14.78	14.88	14.84	14.84	14.06
25	*10*	14.73	14.64	14.83	14.75	14.75	13.98
	15	14.70	14.57	14.80	14.70	14.69	14.07
	20	14.71	14.49	14.81	14.64	14.62	14.03
	5	14.84	14.81	14.90	14.88	14.88	14.10
50	*10*	14.75	14.69	14.85	14.83	14.83	14.08
	15	14.72	14.62	14.81	14.79	14.79	14.00
	20	14.72	14.58	14.80	14.77	14.77	14.02
	5	14.82	14.82	14.88	14.86	14.86	14.16
100	*10*	14.80	14.74	14.87	14.86	14.85	14.10
	15	14.77	14.69	14.86	14.83	14.84	14.08
	20	14.76	14.65	14.83	14.81	14.81	14.01
Average		14.85	14.74	14.92	14.88	14.88	14.03

variance, which is increasing with the sample size. The Sweep operator oscillates round zero for $T = 100$ with a growing variance, but diverges for the other sample sizes. However, for about the first 70 points, i.e. round 7 000 Markov chain steps, it does not perform much worse than the remaining algorithms. A similar level of accuracy could thus be obtained with the Sweep operator by resetting the crossproduct matrix and reinitializing the calculations after some fixed number of updates.

5 Methods for model space exploration

The posterior distribution of different possible models can be also viewed as a distribution of a set of binary strings or as a binary spatial field. The posterior model probabilities (1) can be then written as

$$p\left(\mathcal{M}_i|\,\mathbf{y}\right) = p\left(\boldsymbol{\gamma}_{\mathcal{M}_i}|\,\mathbf{y}\right) \propto m\left(\mathbf{y}|\,\boldsymbol{\gamma}_{\mathcal{M}_i}\right) p\left(\boldsymbol{\gamma}_{\mathcal{M}_i}\right), \tag{48}$$

where $\boldsymbol{\gamma}_{\mathcal{M}_i} = (\gamma_1, \ldots, \gamma_N)'$ is a binary vector, with $\gamma_k = 1$ indicating inclusion and $\gamma_k = 0$ omission of variable k from model $\mathcal{M}_i$.

When evaluating the different MCMC algorithms we will take the basic RJMCMC algorithm 1 with both Add/Drop and Swap moves as our base case. The jump probabilities for the Add/Drop and Swap moves are as described in Section 2 and we randomize between the Add/Drop and Swap moves, proposing to swap variables with probability 1/2. We refer to this as the RJ ADS algorithm. We also consider a simplified version with only the Add/Drop move, i.e. the probability of proposing a swap is zero. This is the same as the $(\text{MC})^3$ algorithm of Madigan and York (1995) and we refer to this as the RJ AD algorithm.

5.1 Gibbs and Metropolis-Hastings samplers

When the marginal likelihood is available in closed form a Gibbs sampler can be used to simulate the posterior distribution of the binary vector $\boldsymbol{\gamma}$ directly. The Gibbs sampler is obtained as a special case of the basic RJMCMC Algorithm 1B by selecting the jump distribution as the conditional posterior

$$p\left(\gamma_i'|\mathbf{y}, \boldsymbol{\gamma}_{\setminus i}\right) = \frac{m\left(\mathbf{y}|\gamma_i', \boldsymbol{\gamma}_{\setminus i}\right) p\left(\gamma_i'|\boldsymbol{\gamma}_{\setminus i}\right)}{\sum_{j=0}^{1} m\left(\mathbf{y}|\gamma_i = j, \boldsymbol{\gamma}_{\setminus i}\right) p\left(\gamma_i = j|\boldsymbol{\gamma}_{\setminus i}\right)} \tag{49}$$

when updating component i with current value γ_i to a (possibly) different value γ_i'. This proposes to stay with probability $p\left(\gamma_i|\mathbf{y}, \boldsymbol{\gamma}_{\setminus i}\right) = 1 - p\left(\gamma_i'|\mathbf{y}, \boldsymbol{\gamma}_{\setminus i}\right)$, and move with probability $1 - p\left(\gamma_i|\mathbf{y}, \boldsymbol{\gamma}_{\setminus i}\right) = p\left(1 - \gamma_i|\mathbf{y}, \boldsymbol{\gamma}_{\setminus i}\right)$. The acceptance probability (4) clearly simplifies to 1. If in addition the index i to update is drawn with probability $1/N$ it is clear that this is analogous to the basic RJMCMC algorithm with only the Add/Drop move. The probability of moving to a new model is, however, larger for the RJMCMC scheme than for the Gibbs sampler, and it follows from Peskun (1973) that RJMCMC with only Add/Drop moves will mix better and is statistically more efficient.

In practice the Gibbs sampler is usually implemented with a systematic scan updating all the variables, as in Smith and Kohn (1996), rather than drawing a single variable to update at random. In this case the Gibbs sampler corresponds to a thinned RJMCMC sampler where every N^{th} draw is retained.

Algorithm 3 Gibbs sampling scheme for variable selection

Suppose that the Markov chain is at model $\mathcal{M}$ represented by the binary vector $\boldsymbol{\gamma}$.

1. For $i = 1, \ldots, N$, draw γ_i' from the conditional posterior $p\left(\gamma_i' | \, \mathbf{y}, \boldsymbol{\gamma}_{\setminus i}\right)$
2. Set the new model to $\mathcal{M}'$ as given by $\boldsymbol{\gamma}'$.

Algorithm 4 Kohn, Smith and Chan Metropolis-Hastings sampling scheme with prior for $\boldsymbol{\gamma}$ as a proposal probability

Suppose that the Markov chain is at model $\mathcal{M}$ represented by the binary vector $\boldsymbol{\gamma}$.

1. For $i = 1, \ldots, N$,
   i. Draw γ_i' from $p\left(\gamma_i | \boldsymbol{\gamma}_{\setminus i}\right)$
   ii. Accept the proposal with probability

$$\alpha = \frac{m\left(\mathbf{y} | \, \gamma_i', \boldsymbol{\gamma}_{\setminus i}\right)}{m\left(\mathbf{y} | \, \gamma_i, \boldsymbol{\gamma}_{\setminus i}\right)}. \tag{50}$$

   If $\gamma_i' = \gamma_i$ no likelihood evaluation is needed.
2. Set the new model to $\mathcal{M}'$ as given by $\boldsymbol{\gamma}'$.

Kohn, Smith, and Chan (2001) propose several sampling schemes designed to reduce the computational burden by avoiding unnecessary evaluations of the marginal likelihood. The Gibbs sampler and the basic RJMCMC scheme each require that $m\left(\mathbf{y} | \, \boldsymbol{\gamma}'\right)$ is evaluated in every step but in many, perhaps most, cases we will remain at the same model. In particular, it is likely that an attempt to change γ_i from 0 to 1 will fail since the total number of variables, N, is typically large relative to the number of variables in the model. Here, we consider sampling scheme 2 (SS2) of Kohn, Smith, and Chan (2001) outlined in Algorithm 4. It reduces the number of evaluations of the marginal likelihood by using the conditional prior $p\left(\gamma_i | \boldsymbol{\gamma}_{\setminus i}\right)$ as the proposal distribution when considering flipping γ_i in an Add/Drop move. If the prior probability $p\left(\gamma_i = 1 | \boldsymbol{\gamma}_{\setminus i}\right)$ is small, the proposal will often be to stay at $\gamma_i = 0$ and a likelihood evaluation is avoided. If, on the other hand, $\gamma_i = 1$, it is likely that a flip is proposed and the marginal likelihood must be evaluated, but the situation when $\gamma_i = 1$ should occur relatively less frequently. Kohn, Smith, and Chan (2001) show that the SS2 scheme is statistically less efficient than the Gibbs sampler, but required only about 10% as many likelihood evaluations in their application.

The Kohn, Smith, and Chan (2001) idea of reducing the number of likelihood evaluations can also be applied to the RJMCMC sampling scheme by modifying the Add/Drop move proposal probabilities. Instead of selecting a variable index i with probability $1/N$ and proposing to flip γ_i, we select and index at random and propose to *flip* the value of γ_i with probability $p\left(\gamma_i = 1 | \boldsymbol{\gamma}_{\setminus i}\right)$, and propose to *stay* with probability $p\left(\gamma_i = 0 | \boldsymbol{\gamma}_{\setminus i}\right)$ when $\gamma_i = 0$. The proposal probabilities for $\gamma_i = 1$ are analogous. With just an Add/Drop move this is simply a random scan version of the SS2 scheme. For completeness we give

Algorithm 5 RJMCMC with KSC proposal probabilities

Suppose that the Markov chain is at model $\mathcal{M}$ represented by the binary vector $\boldsymbol{\gamma}$ with k variables in the model.

1. With probability δ attempt to add or drop a variable.

   Draw an index $i = 1, \ldots, N$ with probability $1/N$ and γ_i' from $p\left(\gamma_i|\boldsymbol{\gamma}_{\setminus i}\right)$.

   The jump probability is $j(\mathcal{M}'|\mathcal{M}) = \delta p\left(\gamma_i'|\boldsymbol{\gamma}_{\setminus i}\right)$ and the probability of the reverse jump is $j(\mathcal{M}|\mathcal{M}') = \delta p\left(1-\gamma_i'|\boldsymbol{\gamma}_{\setminus i}\right)$

2. Otherwise attempt to swap two variables.

   Select an index i at random from $\{i : \gamma_i = 1\}$, an index j from $\{j : \gamma_j = 0\}$ and set $\gamma_i = 0, \gamma_j = 1$. This replaces variable i with variable j. The jump probabilities are $j(\mathcal{M}'|\mathcal{M}) = j(\mathcal{M}|\mathcal{M}') = (1-\delta)/k(N-k)$.

3. Accept the move with probability

$$\alpha = \min\left\{1, \frac{m\left(\mathbf{y}|\mathcal{M}'\right)p\left(\mathcal{M}'\right)j\left(\mathcal{M}|\mathcal{M}'\right)}{m\left(\mathbf{y}|\mathcal{M}\right)p\left(\mathcal{M}\right)j\left(\mathcal{M}'|\mathcal{M}\right)}\right\} \tag{51}$$

   otherwise stay at the current model.

this as Algorithm 5.

Kohn, Smith, and Chan (2001) also propose versions of SS2 that update blocks of indices γ_i rather than a single index. The block schemes turned out to be less statistically efficient than one at a time updating with little additional computational savings. Consequently, we will not consider the block updating schemes here. Instead we consider a recently proposed scheme based on the Swendsen-Wang algorithm where the blocks are formed dynamically.

5.2 Swendsen-Wang algorithm for Bayesian variable selection

When there are high posterior correlations between components of $\boldsymbol{\gamma}$, the usual Markov chain Monte Carlo methods for exploring the posterior, which update one component of $\boldsymbol{\gamma}$ at a time, can mix slowly. Nott and Green (2004) propose in such cases to use a MCMC algorithm analogous to the Swendsen-Wang algorithm for the Ising model[8]. See Higdon (1998) for a review of the Swendsen-Wang algorithm.

Nott and Leonte (2004) combine the method of Nott and Green (2004) with a RJMCMC to obtain a sampling scheme for Bayesian variable selection in generalized linear models, which is applicable when the regression coefficients can not be integrated out of the posterior distribution analytically.

Let $\mathbf{v} = \{v_{ij} : 1 \leq i < j \leq N\}$ be a collection of auxiliary variables, and $\boldsymbol{\beta}_{\boldsymbol{\gamma}}$ be the subvector of $\boldsymbol{\beta}$ consisting of nonzero elements. The target distribution of the Markov chain is

$$p\left(\boldsymbol{\beta}_{\boldsymbol{\gamma}}, \boldsymbol{\gamma}, \mathbf{v}|\mathbf{y}\right) \propto p\left(\mathbf{y}|\boldsymbol{\beta}_{\boldsymbol{\gamma}}, \boldsymbol{\gamma}\right)p\left(\boldsymbol{\beta}_{\boldsymbol{\gamma}}|\boldsymbol{\gamma}\right)p\left(\boldsymbol{\gamma}\right)p\left(\mathbf{v}|\boldsymbol{\gamma}\right) \tag{52}$$

[8]See for example Kindermann R. and J. Laurie Snell (1980): *Markov Random Fields and Their Applications*. The American Mathematical Society, Rhode Island.

instead of the usual

$$p\left(\boldsymbol{\beta}_{\boldsymbol{\gamma}}, \boldsymbol{\gamma}|\mathbf{y}\right) \propto p\left(\mathbf{y}|\boldsymbol{\beta}_{\boldsymbol{\gamma}}, \boldsymbol{\gamma}\right) p\left(\boldsymbol{\beta}_{\boldsymbol{\gamma}}|\boldsymbol{\gamma}\right) p\left(\boldsymbol{\gamma}\right). \tag{53}$$

Adding auxiliary variables can often simplify calculations and lead to improved mixing. In general, it may be possible to design good sampling algorithms for the distribution $p\left(\boldsymbol{\beta}_{\boldsymbol{\gamma}}, \boldsymbol{\gamma}, \mathbf{v}\right)$ more easily then for the distribution $p\left(\boldsymbol{\beta}_{\boldsymbol{\gamma}}, \boldsymbol{\gamma}\right)$. One can simulate from the joint distribution $p\left(\boldsymbol{\beta}_{\boldsymbol{\gamma}}, \boldsymbol{\gamma}, \mathbf{v}\right)$ and then obtain information about the posterior distribution on the model parameters by ignoring the simulated values for the auxiliary variables $\mathbf{v}$. The auxiliary variables do not need to have an immediate interpretation.

The conditional distribution for the auxiliary variables in (52) is

$$p\left(\mathbf{v}|\boldsymbol{\gamma}\right) = \frac{\prod_{i<j} \mathit{1}\left(0 \leq v_{ij} \leq \exp\left(\psi_{ij} \mathit{1}\left(\gamma_i = \gamma_j\right)\right)\right)}{\exp\left(\sum_{i<j} \psi_{ij} \mathit{1}\left(\gamma_i = \gamma_j\right)\right)}, \tag{54}$$

where $\mathit{1}\left(\mathcal{K}\right)$ is indicator function, that equals 1 if condition $\mathcal{K}$ is satisfied and 0 otherwise. The parameters ψ_{ij} are interaction parameters and the procedure for determining them is described in the next section. The conditional distribution (54) makes v_{ij} given $\boldsymbol{\gamma}$ conditionally independent and uniform on the interval $\left[0, \exp\left(\psi_{ij} \mathit{1}\left(\gamma_i = \gamma_j\right)\right)\right]$.

The auxiliary variables $\mathbf{v}$ impose constraints on the value of $\boldsymbol{\gamma}$. If $\psi_{ij} > 0$ and $1 \leq v_{ij} \leq \exp\left(\psi_{ij}\right)$, then the condition $v_{ij} \leq \exp\left(\psi_{ij} \mathit{1}\left(\gamma_i = \gamma_j\right)\right)$ is satisfied if $\gamma_i = \gamma_j$. For $\psi_{ij} < 0$ and $\exp\left(\psi_{ij}\right) \leq v_{ij} \leq 1$, the condition $v_{ij} \leq \exp\left(\psi_{ij} \mathit{1}\left(\gamma_i = \gamma_j\right)\right)$ is satisfied if $\gamma_i \neq \gamma_j$. There are no constraints for γ_i and γ_j, if $\psi_{ij} > 0$ and $0 \leq v_{ij} < 1$, and if $\psi_{ij} < 0$ and $0 \leq v_{ij} < \exp\left(\psi_{ij}\right)$. The auxiliary variables define clusters among components of $\boldsymbol{\gamma}$ by the constrains $\gamma_i = \gamma_j$ and $\gamma_i \neq \gamma_j$. The constraints always have at least one feasible solution, because they are based on the current value of $\boldsymbol{\gamma}$. Let $\mathcal{C} = \mathcal{C}\left(\mathbf{v}\right)$ be a cluster defined by the auxiliary variables and denote by $\boldsymbol{\gamma}\left(\mathcal{C}\right)$ the subset of $\boldsymbol{\gamma}$ corresponding to the variables in $\mathcal{C}$, and denote by $\boldsymbol{\gamma}\left(\bar{\mathcal{C}}\right)$ the remaining components of $\boldsymbol{\gamma}$. The elements of $\boldsymbol{\gamma}\left(\mathcal{C}\right)$ can be updated by flipping their values from zero to one and vice versa.

Using clusters allows to update components of $\boldsymbol{\gamma}$ in blocks, rather than one or two at a time. This is very beneficial in situations where predictors are far from orthogonality and thus the sampling schemes for exploring the posterior distribution work very poorly. Algorithm 6 describes the RJMCMC algorithm 1 extended by the Swendsen-Wang algorithm for exploring the model space.

5.2.1 Choice of interaction parameters

The choice of interaction parameters is very important, as the auxiliary variables conditionally remove interactions among components of $\boldsymbol{\gamma}$. Nott and Green (2004) suggest to compute the interaction parameters ψ_{ij} via the expression

$$\psi_{ij}^{U} = 0.5 \times \left(\sum_{\substack{\gamma_i = \gamma_j, \\ \gamma_k = \gamma_k^*,\, k \neq i,j}} \log m\left(\mathbf{y}|\boldsymbol{\gamma}\right) - \sum_{\substack{\gamma_i \neq \gamma_j, \\ \gamma_k = \gamma_k^*,\, k \neq i,j}} \log m\left(\mathbf{y}|\boldsymbol{\gamma}\right) \right), \tag{57}$$

Algorithm 6 Swendsen-Wang reversible jump Markov chain Monte Carlo

Suppose that the Markov chain is at model $\mathcal{M}$ and the interaction parameters ψ_{ij} have been calculated. Denote the binary indicator vector for the parameters of the model $\mathcal{M}$ as $\boldsymbol{\gamma}$.

1. Generate auxiliary variables $\mathbf{v}$ from $p(\mathbf{v}|\boldsymbol{\gamma})$.
2. Uniformly select a variable i from the set of possible variables and find the cluster $\mathcal{C}$ containing the variable i .
3. Propose a jump from model $\mathcal{M}$ to a new model $\mathcal{M}'$ by setting $\boldsymbol{\gamma}'(\mathcal{C}) = 1 - \boldsymbol{\gamma}(\mathcal{C})$.
4. Accept the proposed move with probability

$$\alpha = \min\left\{1, \frac{m(\mathbf{y}|\boldsymbol{\gamma}')\, p(\boldsymbol{\gamma}')}{m(\mathbf{y}|\boldsymbol{\gamma})\, p(\boldsymbol{\gamma})} \exp\left(\sum_{i<j} \psi_{ij}\Big(1(\gamma_i = \gamma_j) - 1\left(\gamma_i' = \gamma_j'\right)\Big)\right)\right\}. \quad (55)$$

5. Set $\mathcal{M} = \mathcal{M}'$ if the move is accepted.

Since $\boldsymbol{\gamma}'\left(\bar{\mathcal{C}}\right) = \boldsymbol{\gamma}\left(\bar{\mathcal{C}}\right)$ the acceptance probability is simplified to

$$\alpha = \min\left\{1, \frac{m(\mathbf{y}|\boldsymbol{\gamma}')\, p(\boldsymbol{\gamma}')}{m(\mathbf{y}|\boldsymbol{\gamma})\, p(\boldsymbol{\gamma})} \exp\left(\sum_{(i,j)\partial\mathcal{C}} \psi_{ij}\Big(1(\gamma_i = \gamma_j) - 1\left(\gamma_i' = \gamma_j'\right)\Big)\right)\right\}, \quad (56)$$

where $\partial\mathcal{C} = \{(i,j) : i < j \text{ and either } i \in \mathcal{C}, j \notin \mathcal{C} \text{ or } i \notin \mathcal{C}, j \in \mathcal{C}\}$.
Steps 1 and 2 can be combined, since only the components of $\mathbf{v}$, which determine the cluster containing γ_i in step 2, have to be generated.

where $\boldsymbol{\gamma}^*$ is set to be a vector of ones. The interaction parameters are then formed as

$$\psi_{ij} = c\psi_{ij}^U 1\big(\left|c\psi_{ij}^U\right| \geq t\big), \quad (58)$$

where c is a scaling factor and t is a threshold parameter. The transformation (58) scales down the values ψ_{ij}^U by a factor of c, and truncates these values to zero if the scaled values are less than t in magnitude. With $\boldsymbol{\gamma}^*$ to be vector of ones, c is chosen so that all the algorithm interaction parameters lie in the interval $[-1, 1]$, and t is set to 0.1. From the definition of the auxiliary variables and the constraints conditions it is easily seen that if the current states for γ_i and γ_j are such that a constraint is possible between them at the next iteration, then the probability of such a constraint is $1 - \exp(-|\psi_{ij}|)$. If $|\psi_{ij}|$ is large, this constraint probability will be close to one. If many of the algorithm interaction parameters are large, the cluster $\mathcal{C}$ tends to contain large numbers of variables, and at least one of the γ in a large cluster will have a value fixed by the likelihood. This means that any proposal to flip the values in a large cluster is unlikely to be accepted, which prevents mixing in this sampling scheme. Scaling down the parameters has therefore a beneficial effect on the performance of the algorithm.

In datasets containing many variables computing all interaction parameters ψ_{ij} can be very time consuming. Nott and Green (2004) suggest to reduce the number of pairs (i, j) for which the interactions parameters have to be calculated by using standard mul-

ticollinearity diagnostic, the variance proportions. For linearly independent variables the interaction parameters are set to zero, and only for variables (i, j) involved in severe linear dependence are they allowed to be nonzero.

Write $\mathbf{X}'\mathbf{X} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}'$ for the eigenvalue decomposition, where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues with diagonal entries $\lambda_1, \ldots, \lambda_N$ and $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_N]$ is a orthogonal matrix with columns given by the eigenvectors $\mathbf{w}_1, \ldots, \mathbf{w}_N$ of $\mathbf{X}'\mathbf{X}$. The variance of the estimated parameters, $\hat{\boldsymbol{\beta}}$, can be written as

$$\text{var}\left(\hat{\boldsymbol{\beta}}\right) = \sigma^2 \left(\mathbf{X}'\mathbf{X}\right)^{-1} = \sigma^2 \mathbf{W}\mathbf{\Lambda}^{-1}\mathbf{W}', \tag{59}$$

and the variance of $\hat{\beta}_i$ is

$$\text{var}\left(\hat{\beta}_i\right) = \sigma^2 \sum_{j=1}^{N} \frac{w_{ij}^2}{\lambda_j}. \tag{60}$$

The variance proportion that can be attributed to the eigenvalue λ_k is

$$\xi_{ki} = \frac{w_{ik}^2/\lambda_k}{\sum_{j=1}^{N} w_{ij}^2/\lambda_j}. \tag{61}$$

If for a given small value of λ_k both variance proportions ξ_{ki} and ξ_{kj} are large this suggests that variables i and j are involved in a near linear dependence among the regressors. Nott and Green suggest to calculate the interaction parameters ψ_{ij}^U only for such pairs (i, j), where both ξ_{ki} and ξ_{kj} are bigger than some cut-off value for some eigenvalue λ_k. The interaction parameters are then scaled and truncated as in expression (58). The cut-off value is set to 0.25 in this paper.

6 Performance of MCMC algorithms

In this section we wish to address the issues outlined in Section 2. The two main questions are: how quickly a sampler converges to the posterior distribution and how fast does it account for all but a negligible proportion of the total posterior probabilities. We investigate these in two experiments based on different designs for the matrix of potential explanatory variables. The first experiment is the same as the basic design with 15 variables described in Section 4.

The second experiment is more challenging with severe and complicated multicollinearity among potential regressors. It is based on an example originally in George and McCulloch (1997) and used by, among others, Nott and Green (2004) and Nott and Leonte (2004). We simulate 15 variables $\mathbf{X}_{(T\times 15)}$ as follows. First, 16 i.i.d. standard normal variables, $\mathbf{z}_i$, are generated. Then we construct the regressors as $\mathbf{x}_i = \mathbf{z}_i + 2\mathbf{z}_{16}$ for $i = 1, 3, 5, 8, 9, 10, 12, 13, 14, 15$. Furthermore, $\mathbf{x}_i = \mathbf{x}_{i-1} + 0.15\mathbf{z}_i$, for $i = 2, 4, 6$, $\mathbf{x}_7 = \mathbf{x}_8 + \mathbf{x}_9 - \mathbf{x}_{10} + 0.15\mathbf{z}_7$, and finally $\mathbf{x}_{11} = -\mathbf{x}_{12} - \mathbf{x}_{13} + \mathbf{x}_{14} + \mathbf{x}_{15} + 0.15\mathbf{z}_{11}$. This leads to a correlation about 0.998 between $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ for variables $1, 3$ and 5, and strong linear dependencies among $(\mathbf{x}_7, \ldots, \mathbf{x}_{10})$ and $(\mathbf{x}_{11}, \ldots, \mathbf{x}_{15})$. The true model is

$$y_t = \mathbf{x}_t\boldsymbol{\beta} + \sigma\varepsilon_t, \tag{62}$$

where

$$\boldsymbol{\beta} = \begin{pmatrix} 1.5 & 0 & 1.5 & 0 & 1.5 & 0 & 1.5 & -1.5 & 0 & 0 & 1.5 & 1.5 & 1.5 & 0 & 0 \end{pmatrix}',$$

the disturbances ε_t are i.i.d. standard normal, and $\sigma = 2.5$. In both experiments we set $T = 250$. We generate 100 different datasets for each experiment. We refer to these two experiments as the FLS and NL experiment, respectively.

In order to assess the performance of the different samplers we test how close the posterior distribution of the chain is to the exact posterior distribution. Calculating the exact posterior distribution is trivial for these experiments since the marginal likelihoods are available in closed form. With only $N = 15$ predictors in both exercises, and the constant term always included in the model, exactly 32768 models have to be evaluated. Using our fastest least squares routine this takes very little time, about 0.2 seconds.

To assess the convergence we use the Kolmogorov-Smirnov (KS) test as one metric. The KS test is defined as the maximum value of absolute difference between the empirical cumulative distribution function, $S_n(x)$, and the assumed cumulative distribution function, $P(x)$,

$$D = \max_{-\infty<x<\infty} |S_n(x) - P(x)|, \tag{63}$$

where n denotes the sample size.

Brooks, Giudici, and Philippe (2003) use the KS test for testing homogeneity of subpopulations of different chains under the assumption that replicated chains that have converged generate similar posterior model probability estimates. The use of the KS test is justified by considering the sample-path of a model indicator, $\mathcal{M}_i$. Brooks, Giudici, and Philippe point out that the KS test is invariant under reparametrization of x. The test is however not invariant to model labelling, so the value of D may change with a different model labelling. On the other hand, the interpretation of the diagnostic plots over time, in terms of whether or not the simulations are performing well, is generally invariant across different labelling schemes. The KS test requires continuous variables to derive an exact distribution, but as Brooks, Giudici, and Philippe also validate, with sufficiently large number of models, this assumption is valid in an approximate sense. In the test (63) the realizations of x are replaced by $\mathcal{M}_i$, the model indicator, and $n = M$, the total number of models visited by the Markov chain.

We evaluate the following samplers:

GIBBS	Gibbs sampler (Algorithm 3)
KSC	Kohn, Smith, and Chan's (2001) SS2 algorithm (Algorithm 4)
RJ ADS	RJMCMC (Algorithm 1) with Add/Drop and Swap moves, as defined on page 1 and $\delta = 1/2$
RJ AD	RJMCMC with Add/Drop move only, $\delta = 0$
RJ KSC ADS	RJMCMC with KSC jump proposal, ADS moves, $\delta = 1/2$, (Algorithm 5)
RJ KSC AD	RJMCMC with KSC jump proposal, no swap move, $\delta = 0$
SWANG	Swendsen-Wang algorithm (Algorithm 6)

6.1 Results

For each sampler and each dataset we run the chain in total for $s = 125\,000$, $250\,000$, $500\,000$, $1\,000\,000$ and $2\,000\,000$ steps. We stop the chain after each $r = s/5$ steps, giving 5 control points. At the control points we record the output of the chain, i.e. the KS test statistics are calculated based on the last r steps. After the control point the chain continues moving form the last visited model $\mathcal{M}_i^{(r)}$, but the count of visited models is

discarded. For each s, the smallest r with few significant KS tests gives an indication of the burn-in needed, provided that the 'batch-size', r, is sufficient to provide a good estimate of the posterior model probabilities, or at least allow us to visit all but a negligible set of models. The KS test is carried out both for the exact posterior probabilities based on visited models by the Markov chain and the relative frequencies from the chain. We refer to these as exact probabilities and MCMC probabilities, respectively. Tables 5 - 8 report the number of significant KS tests at significance level $\alpha = 0.05$.

The results for the 'simple' FLS experiment do not show any significant differences among the samplers. All the samplers perform well, and according to the KS test, 25 000 steps of the Markov chain is sufficient for estimating the posterior model probabilities.

Turning to the more complicated NL experiment, we find substantial differences. For the exact probabilities the Swendsen-Wang sampler produces posterior distributions very close to the 'true' distribution already for short chains. It is followed by the RJ KSC ADS and RJ ADS samplers. It appears that these three algorithms will visit all but a negligible set of models in 25 000 steps. The next group of algorithms are the Gibbs sampler and RJ AD where 50 000 draws is sufficient for visiting all the important models. The chains produced by the KSC and RJ KSC AD algorithms perform worst and 100 000 steps appear to be needed. It is interesting to note the role of the Swap move. Algorithms with only the Add/Drop move perform badly, whereas the corresponding algorithm with a Swap move performs quite well. The inferior performance of the algorithms with KSC type proposals is to some extent expected. It remains to be seen if this is made up for by savings in computational time.

Turning to Table 8 and the issue of how well the relative frequencies of the visited models estimate the posterior probabilities, we again find that the Swendsen-Wang sampler performs best. The KS test is unable to reject the null hypothesis that the sampler has converged after only 25 000 steps. The performance of the Gibbs sampler, RJ ADS, RJ AD and RJ KSC ADS is similar to each other, these samplers appear to require about 400 000 steps for acceptable results. Again KSC and RJ KSC AD perform worst with KSC doing somewhat better than RJ KSC AD. For the latter we clearly need more than 400 000 steps of the Markov chain for good estimates of the posterior distribution.

Figures 2(a) and 2(b) plot the average number of steps that each sampler needed to reach a desired coverage. This is complementary to the results in Tables 5 and 7 and gives a somewhat more nuanced picture of the FLS experiment. The inferior performance of the KSC and RJ KSC AD algorithms is now evident for this dataset as well with these algorithms requiring about twice the number of steps for 99% coverage. For the NL experiment we again find that Swendsen-Wang performs best followed by RJ ADS with the Gibbs sampler, RJ AD and RJ KSC ADS in a third group.

This far we have not taken the computational requirements into account. Figures 3(a) and 3(b) show the average CPU time required for a desired level of coverage. In the FLS experiment we find that the Swendsen-Wang sampler requires most CPU time for 99% coverage, and that KSC along with the Gibbs sampler and RJ AD require the least. The RJ ADS, RJ KSC ADS and RJ KSC AD occupy the middle ground with RJ ADS doing best of these three algorithms. The speed of the KSC algorithm clearly compensates for the lack of statistical efficiency in this case. Turning to the NL experiment, we find that the RJ ADS algorithm requires least CPU time followed by KSC and RJ KSC ADS. The Gibbs sampler and the Swendsen-Wang algorithm have similar CPU time requirements, while the RJ AD and RJ KSC AD require most CPU time with RJ KSC AD performing

Table 5 Number of significant Kolmogorov-Smirnov tests for exact probabilities, FLS experiment.

r		*control step*				
s		1	2	3	4	5
	GIBBS	0	0	0	0	0
	KSC	0	0	0	0	0
25 000	RJ ADS	0	0	0	0	0
125 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	1	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	1	0	0	0	0
	GIBBS	0	0	0	0	0
	KSC	0	0	0	0	0
50 000	RJ ADS	0	0	0	0	0
250 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	1	0	0	0	0
	GIBBS	0	0	0	0	0
	KSC	0	0	0	0	0
100 000	RJ ADS	0	0	0	0	0
500 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	1	0	0	0	0
	GIBBS	0	0	0	0	0
	KSC	0	0	0	0	0
200 000	RJ ADS	0	0	0	0	0
1 000 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	1	0	0	0	0
	GIBBS	0	0	0	0	0
	KSC	0	0	0	0	0
400 000	RJ ADS	0	0	0	0	0
2 000 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	0	0	0	0	0

Table 6 Number of significant Kolmogorov Smirnov tests for MCMC probabilities, FLS experiment.

r		*control step*				
s		1	2	3	4	5
	GIBBS	0	0	0	0	0
	KSC	0	2	0	0	0
25 000	RJ ADS	1	0	0	0	0
125 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	6	1	3	5	1
	RJ KSC AD	0	0	0	0	0
	SWANG	1	1	0	0	0
	GIBBS	0	0	0	0	0
	KSC	1	0	0	0	0
50 000	RJ ADS	0	0	0	0	0
250 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	2	0	0	0	2
	RJ KSC AD	0	0	0	0	0
	SWANG	1	0	0	0	0
	GIBBS	0	0	0	0	0
	KSC	0	0	0	0	0
100 000	RJ ADS	0	0	0	0	0
500 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	1	0	0	0	0
	GIBBS	0	0	0	0	0
	KSC	0	0	0	0	0
200 000	RJ ADS	0	0	0	0	0
1 000 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	1	0	0	0	0
	GIBBS	0	0	0	0	0
	KSC	0	0	0	0	0
400 000	RJ ADS	0	0	0	0	0
2 000 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	0	0	0	0	0

Table 7 Number of significant Kolmogorov Smirnov tests for exact probabilities, NL experiment.

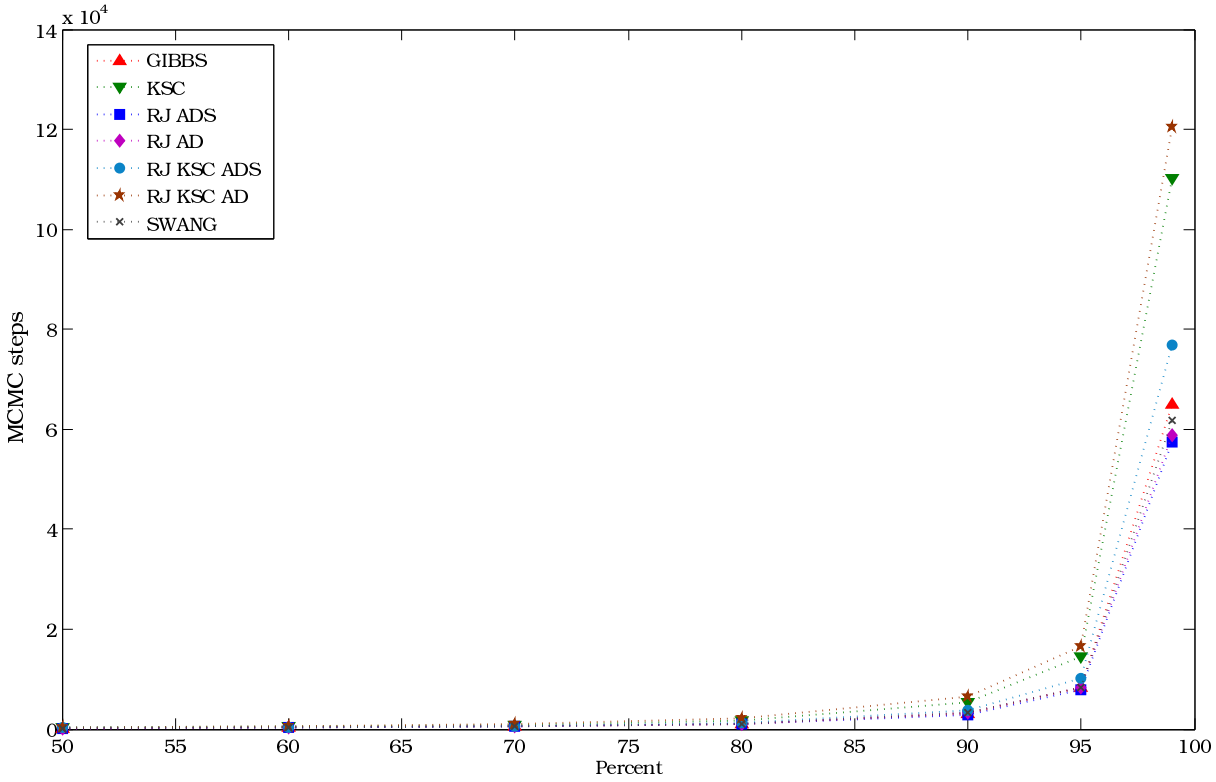
r		*control step*				
s		1	2	3	4	5
	GIBBS	5	13	15	9	6
	KSC	43	24	29	47	46
25 000	RJ ADS	3	0	0	1	0
125 000	RJ AD	14	19	11	2	13
	RJ KSC ADS	2	6	1	4	3
	RJ KSC AD	51	30	49	52	48
	SWANG	0	0	0	0	0
	GIBBS	2	0	1	0	0
	KSC	3	5	13	3	16
50 000	RJ ADS	4	0	0	0	0
250 000	RJ AD	5	0	6	2	4
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	12	16	28	33	33
	SWANG	0	0	0	0	0
	GIBBS	1	0	0	0	0
	KSC	0	0	4	1	0
100 000	RJ ADS	1	0	0	0	0
500 000	RJ AD	0	0	0	2	0
	RJ KSC ADS	1	0	0	0	0
	RJ KSC AD	3	16	1	6	0
	SWANG	0	0	0	0	0
	GIBBS	3	0	0	0	0
	KSC	1	0	0	2	0
200 000	RJ ADS	2	0	0	0	0
1 000 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	0	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	0	0	0	0	0
	GIBBS	0	0	0	0	0
	KSC	1	0	0	0	0
400 000	RJ ADS	2	0	0	0	0
2 000 000	RJ AD	0	0	0	0	0
	RJ KSC ADS	1	0	0	0	0
	RJ KSC AD	0	0	0	0	0
	SWANG	0	0	0	0	0

Table 8 Number of significant Kolmogorov Smirnov tests for MCMC probabilities, NL experiment.

r		*control step*				
s		1	2	3	4	5
	GIBBS	72	55	44	21	56
	KSC	51	51	65	47	46
25 000	RJ ADS	38	45	40	36	31
125 000	RJ AD	29	49	49	72	63
	RJ KSC ADS	38	37	43	49	35
	RJ KSC AD	53	72	74	56	51
	SWANG	0	0	0	1	1
	GIBBS	55	33	40	22	39
	KSC	42	40	22	40	44
50 000	RJ ADS	28	29	20	19	18
250 000	RJ AD	37	59	53	44	52
	RJ KSC ADS	28	40	29	26	22
	RJ KSC AD	46	50	51	45	47
	SWANG	0	0	0	1	0
	GIBBS	24	19	32	26	28
	KSC	22	17	34	37	40
100 000	RJ ADS	24	12	6	11	16
500 000	RJ AD	30	21	31	26	24
	RJ KSC ADS	24	15	22	13	24
	RJ KSC AD	39	32	34	24	50
	SWANG	0	0	0	0	0
	GIBBS	12	20	17	11	12
	KSC	7	19	16	15	22
200 000	RJ ADS	9	4	17	11	7
1 000 000	RJ AD	9	12	16	11	27
	RJ KSC ADS	11	8	19	17	4
	RJ KSC AD	26	21	52	21	24
	SWANG	0	0	0	0	0
	GIBBS	8	4	2	5	3
	KSC	4	6	17	11	16
400 000	RJ ADS	5	9	5	2	2
2 000 000	RJ AD	3	8	2	7	19
	RJ KSC ADS	6	4	0	3	5
	RJ KSC AD	12	35	19	16	12
	SWANG	0	0	0	0	0

Figure 2 Average number of MCMC steps to reach model space coverage.

(a) FLS experiment



(b) NL experiment

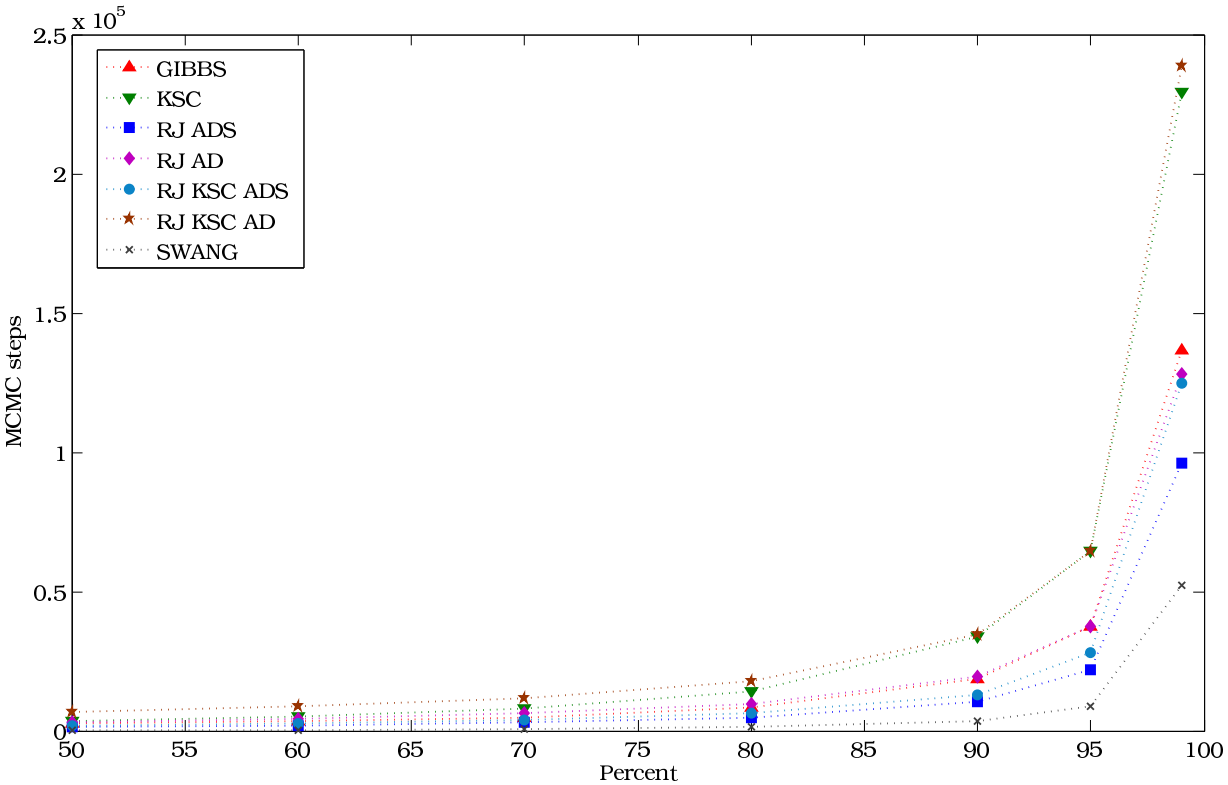
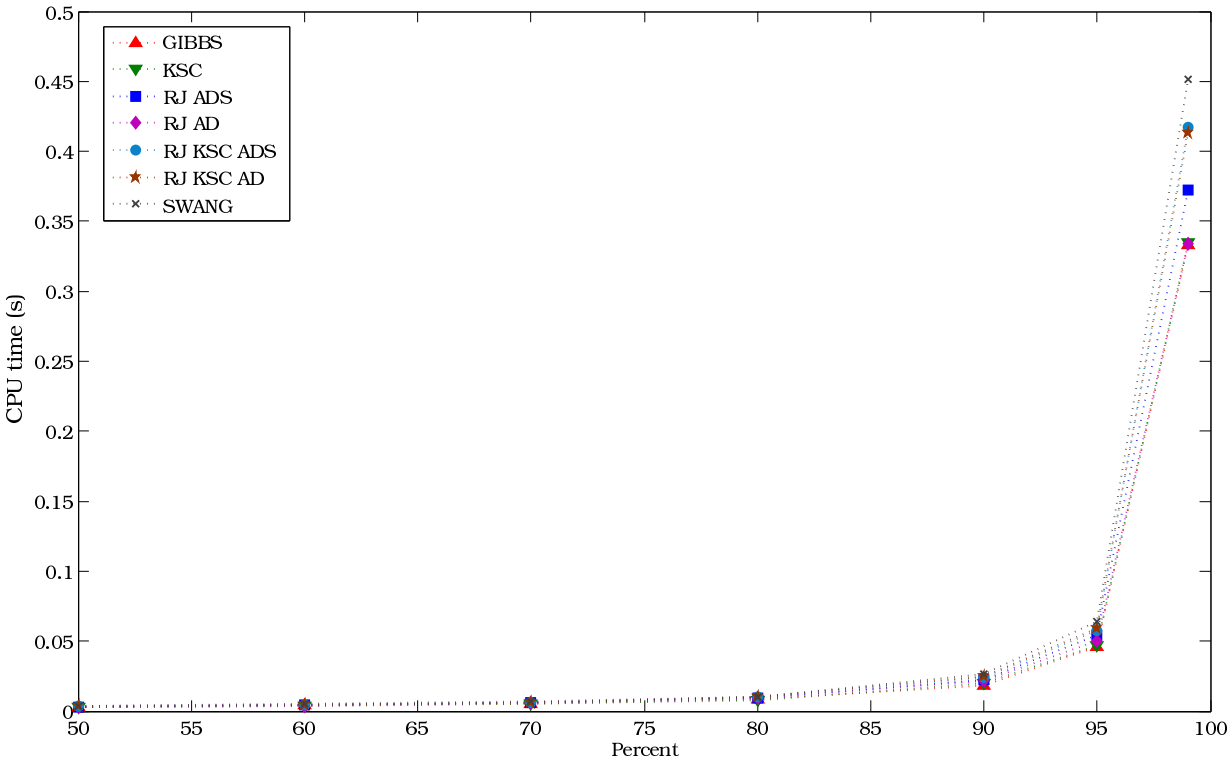
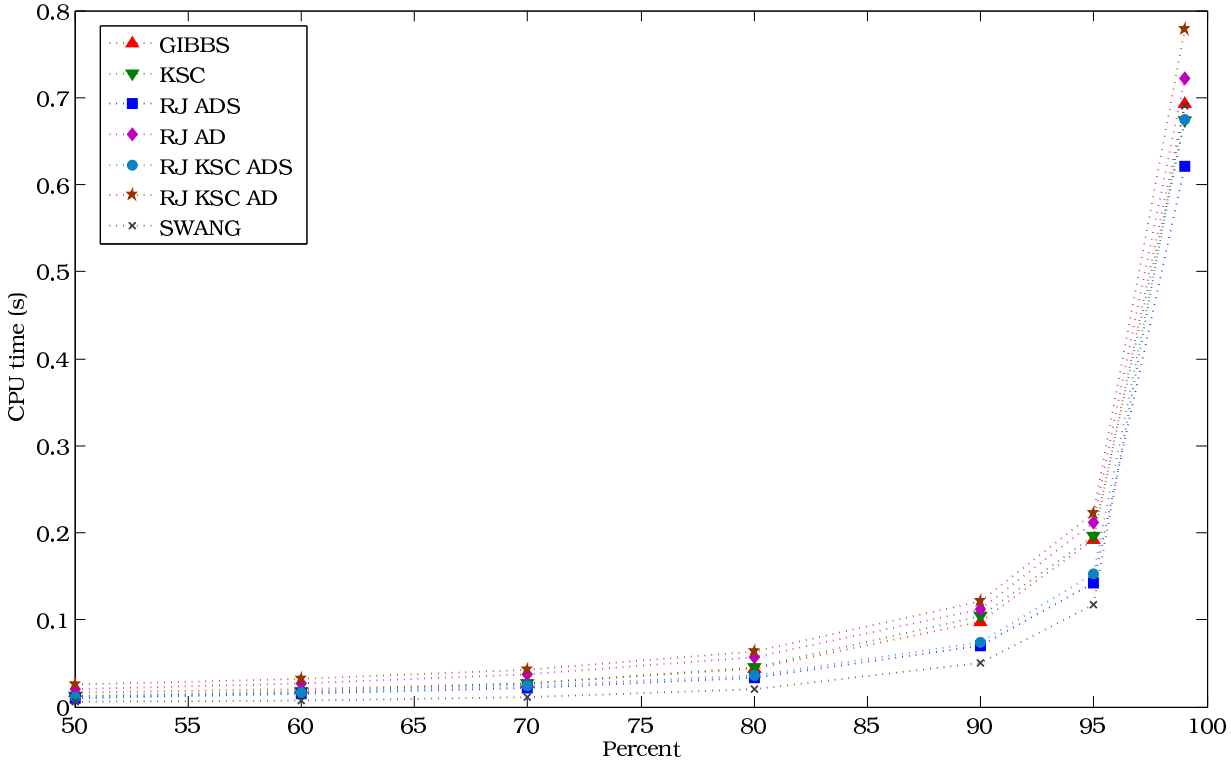


Figure 3 Average CPU time to reach model space coverage.

(a) FLS experiment



(b) NL experiment



quite poorly.

7 Conclusions

Solving least squares problems is an important subset of the calculations required for Bayesian model averaging and variable selection. When analyzing the performance of different algorithms we find obvious gains from solvers that update previous model estimates. Overall Cholesky decomposition, Cholesky updating and the sweep operator stand out as the fastest algorithms with a considerable speed gain compared to popular algorithms based on the QR decomposition. The Cholesky decomposition and its update are also the most accurate algorithms. It should also be kept in mind that, with the fastest alorithms, the time for calculating the least squares solution needed for the marginal likelihood is only a small fraction of the total time.

MCMC bases schemes for estimating posterior model and variable inclusion probabilities are required for large scale problems and good convergence properties are crucial. When analyzing the convergence properties we find that the performance of the samplers vary considerably, both between samplers and depending on the difficulty of the problem. Most of the MCMC samplers we study are local approaches restricting transitions to adjacent subsets of the model space. The most successful of these is the reversible jump Markov chain Monte Carlo algorithm with Add/Drop and Swap moves (RJ ADS). When there is high dependence among the variables and the chain is mixing slowly, the Swendsen-Wang algorithm which allows for more global moves, provides substantial accuracy improvements compared to the local transition samplers.

The Swendsen-Wang algorithm is clearly a superior sampler among the ones considered here in terms of statistical efficiency. It is, however, relatively complicated to implement and requires more CPU time. Taking CPU time into account, other algorithms such as RJ ADS strike a balance between statistical efficiency and computational efficiency and might be preferable. It should, however, be noted that this is in a setting where the marginal likelihood is available in closed form and can be evaluated efficiently. For more complicated models the computational cost of generating a proposal is less important and the Swendsen-Wang algorithm might be more efficient in terms of computational time. At the same time this is a case where strategies like KSC, that reduce the number of likelihood evaluation, can be expected to do well.

References

Beaton, A. E. (1964): "The use of Special Matrix Operators in Statistical Calculus," Research Bulletin 64-51, Educational Testing Service, Princeton, New Jersey.

Brooks, S. P., P. Giudici, and A. Philippe (2003): "Nonparametric Convergence Assessment for MCMC Model Selection," *Journal of Computational & Graphical Statistics*, 12(1), 1–22.

Daniel, J. W., W. B. Gragg, L. Kaufman, and G. W. Stewart (1976): "Reorthogonalization and Stable Algorithms for Updating the Gram-Schmidt QR Factorization," *Mathematics of Computation*, 30(136), 772–795.

Denison, D. G. T., B. K. Mallick, and A. F. M. Smith (1998): "Automatic Bayesian Curve Fitting," *Journal of the Royal Statistical Society B*, 60(2), 333–350.

Dongarra, J. J., J. R. Bunch, C. B. Moler, and G. W. Stewart (1979): *Linpack Users' Guide*. SIAM, Philadelphia.

Fernández, C., E. Ley, and M. F. Steel (2001): "Benchmark Priors for Bayesian Model Averaging," *Journal of Econometrics*, 100(2), 381–427.

George, E. I., and R. E. McCulloch (1997): "Approaches for Bayesian Variable Selection," *Statistica Sinica*, 7, 339–373.

Golub, G. H., and C. F. van Loan (1996): *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3 edn.

Goodnight, J. H. (1979): "A Tutorial on the Sweep Operator," *The American Statistician*, 33(3), 149–158.

Green, P. J. (1995): "Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination," *Biometrika*, 82(4), 711–732.

Hanson, R. J., and T. Hopkins (2004): "Algorithm 830: Another Visit with Standard and Modified Givens Transformations and a Remark on Algorithm 539," *ACM Transactions on Mathematical Software*, 30(1), 86–94.

Higdon, D. M. (1998): "Auxiliary Variable Methods for Markov Chain Monte Carlo with Applications," *Journal of the American Statistical Association*, 93(442), 585–595.

Hoeting, J., D. Madigan, A. E. Raftery, and C. Volinsky (1999): "Bayesian Model Averaging: A Tutorial," *Statistical Science*, 14(4), 382–417.

Jacobson, T., and S. Karlsson (2004): "Finding Good Predictors for Inflation: A Bayesian Model Averaging Approach," *Journal of Forecasting*, 23(7), 479–496.

Kohn, R., M. Smith, and D. Chan (2001): "Nonparametric Regression Using Linear Combinations of Basis Functions," *Statistics and Computing*, 11(4), 313–322.

Koop, G., and S. Potter (2004): "Forecasting in Dynamic Factor Models Using Bayesian Model Averaging," *Econometrics Journal*, 7(2), 550–565.

Leamer, E. E. (1978): *Specification Searches, Ad hoc Inference with Nonexperimental Data*. John Wiley, New York.

Madigan, D., and A. E. Raftery (1994): "Model Selection and Accounting for Model Uncertainty in Graphical Models Using Occam's Window," *Journal of the American Statistical Association*, 89(428), 1535–1546.

Madigan, D., and J. York (1995): "Bayesian Graphical Models for Discrete Data," *International Statistical Review*, 63, 215–232.

Nott, D. J., and P. J. Green (2004): "Bayesian Variable Selection and the Swendsen-Wang Algorithm," *Journal of Computational & Graphical Statistics*, 13(1), 141–157.

Nott, D. J., and D. Leonte (2004): "Sampling Schemes for Bayesian Variable Selection in Generalized Linear Models," *Journal of Computational & Graphical Statistics*, 13(2), 362–382.

Peskun, P. H. (1973): "Optimum Monte-Carlo Sampling Using Markov Chains," *Biometrika*, 60(3), 607–612.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992): *Numerical Recipes in FORTRAN: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 2 edn.

Raftery, A. E., D. Madigan, and J. A. Hoeting (1997): "Bayesian Model Averaging for Linear Regression Models," *Journal of the American Statistical Association*, 92(437), 179–191.

Raftery, A. E., D. Madigan, and C. Volinsky (1995): "Accounting for Model Uncertainty in Survival Analysis Improves Predictive Performance (with Discussion)," in *Bayesian Statistics 5*, ed. by J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, pp. 323–349. Oxford University Press, Oxford.

Reichel, L., and W. B. Gragg (1990): "Algorithm 686: FORTRAN Subroutines for Updating the QR Decomposition," *ACM Transactions on Mathematical Software*, 16(4), 369–377.

Smith, M., and R. Kohn (1996): "Nonparametric Regression Using Bayesian Variable Selection," *Journal of Econometrics*, 75(2), 317–343.

Visual Numerics, Inc. (1994): *IMSL Fortran Statistical Library, vol. 1.*Visual Numerics, Inc., Houston, Texas, 3 edn.

Appendix A Figures

Figure A.1 Relative accuracy for RSS, $T = 100, N = 50, k = 10$.

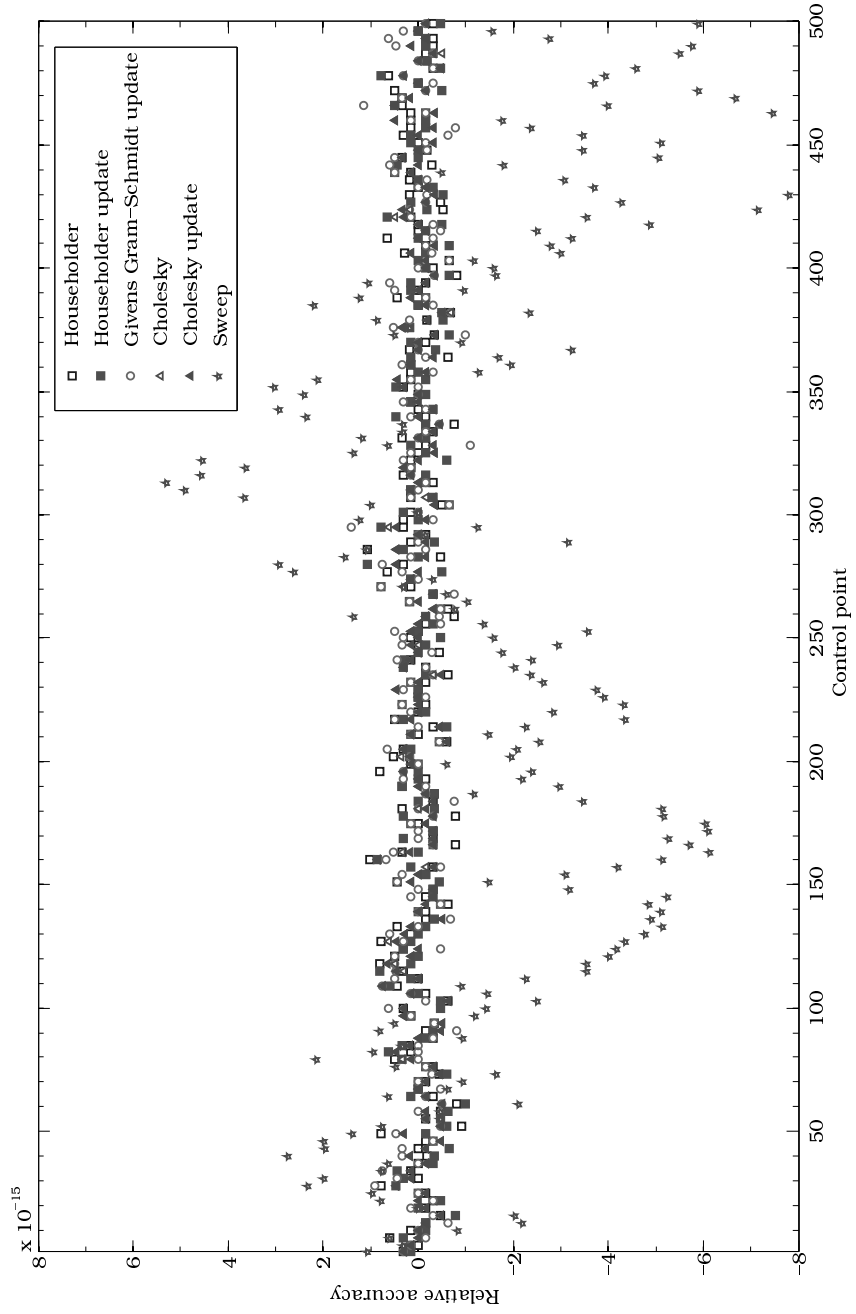


Figure A.2 Relative accuracy for RSS, $T = 250, N = 50, k = 10$.

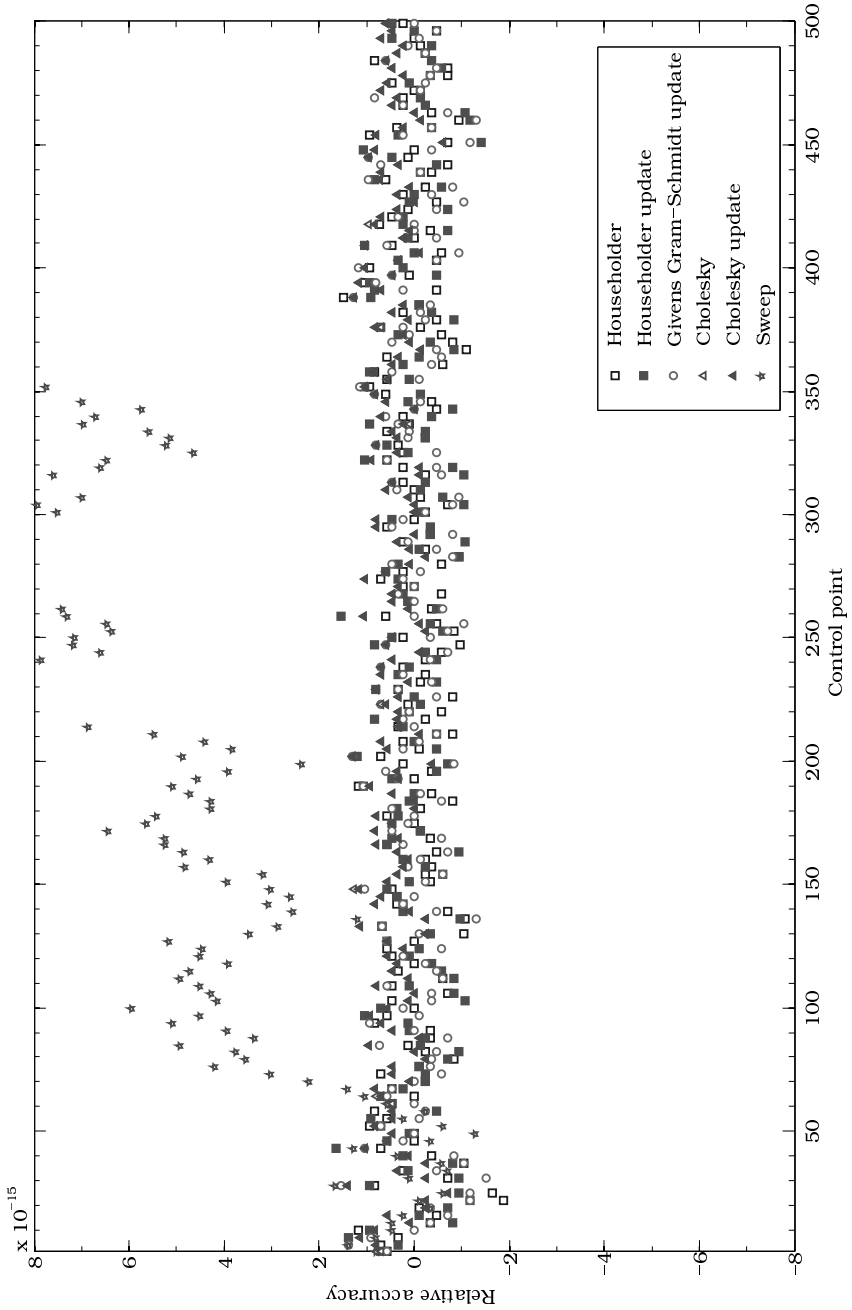
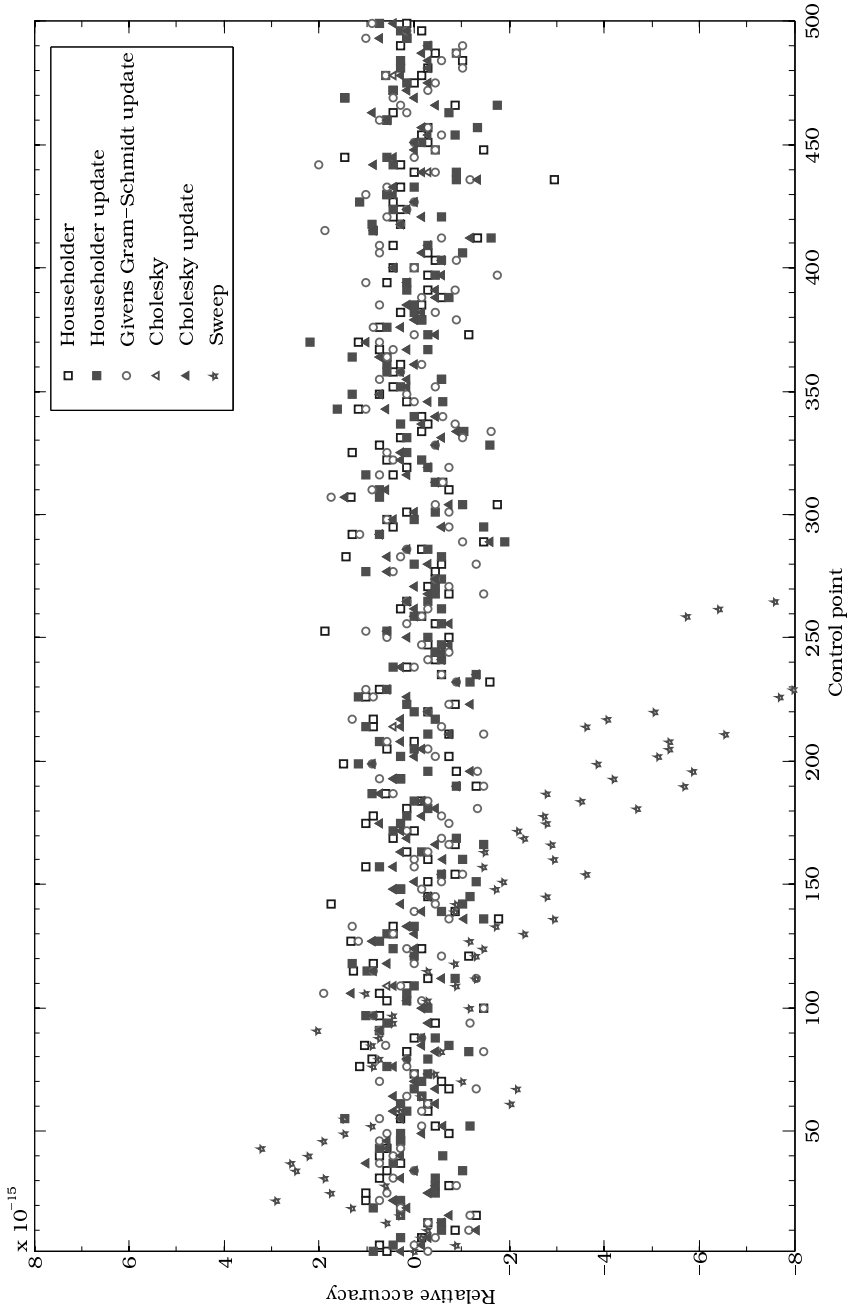


Figure A.3 Relative accuracy for RSS, $T = 400, N = 50, k = 10$.



Appendix B Tables

Table B.1 CPU time (approximation in seconds) for $T = 100$.

		k			
		5	10	15	20
$N = 25$	OLS	17.56	25.38	32.33	38.91
	Householder	4.31	6.62	8.39	9.52
	Householder update	4.26	6.17	7.20	7.24
	Givens Gram-Schmidt update	4.44	5.98	6.35	5.82
	Cholesky	3.78	5.16	5.47	4.78
	Cholesky update	3.79	5.11	5.31	4.41
	Sweep	3.88	5.06	5.06	3.89
	Overhead	3.68	4.86	4.86	3.69
$N = 50$	OLS	20.21	26.95	33.94	41.74
	Householder	6.91	8.18	9.94	12.24
	Householder update	6.87	8.13	8.76	10.00
	Givens Gram-Schmidt update	7.05	7.94	7.83	8.45
	Cholesky	6.40	6.70	7.03	7.54
	Cholesky update	6.42	6.67	6.87	7.15
	Sweep	6.80	6.93	6.94	6.95
	Overhead	6.29	6.42	6.43	6.44
$N = 100$	OLS	24.26	30.99	38.07	45.61
	Householder	10.91	12.55	13.96	16.27
	Householder update	10.92	11.72	12.78	14.07
	Givens Gram-Schmidt update	11.07	11.46	11.82	12.41
	Cholesky	10.38	10.67	11.01	11.57
	Cholesky update	10.41	10.63	10.86	11.15
	Sweep	11.76	11.85	12.23	11.91
	Overhead	10.29	10.35	10.38	10.39

Table B.2 CPU time (approximation in seconds) for $T = 250$.

		k			
		5	10	15	20
$N = 25$	OLS	37.66	55.38	73.57	91.87
	Householder	5.09	9.05	13.30	18.37
	Householder update	4.93	7.90	10.32	12.14
	Givens Gram-Schmidt update	5.34	7.40	8.25	8.61
	Cholesky	3.78	5.14	5.47	4.79
	Cholesky update	3.80	5.25	5.30	4.41
	Sweep	3.87	5.05	5.06	3.89
	Overhead	3.69	4.84	4.85	3.69
$N = 50$	OLS	40.31	57.52	75.11	95.01
	Householder	7.69	10.67	14.93	20.58
	Householder update	7.53	9.51	11.92	14.90
	Givens Gram-Schmidt update	7.95	8.88	9.66	11.10
	Cholesky	6.42	6.71	7.14	7.54
	Cholesky update	6.43	6.67	6.88	7.15
	Sweep	6.80	6.91	6.94	6.95
	Overhead	6.29	6.42	6.42	6.45
$N = 100$	OLS	44.34	61.18	79.25	98.51
	Householder	11.71	14.58	18.82	24.52
	Householder update	11.56	13.44	15.87	18.99
	Givens Gram-Schmidt update	11.96	12.85	13.63	15.02
	Cholesky	10.40	10.66	11.01	11.56
	Cholesky update	10.41	10.64	10.85	11.15
	Sweep	11.77	11.84	11.88	11.90
	Overhead	10.30	10.51	10.38	10.39

Table B.3 CPU time (approximation in seconds) for $T = 400$.

		k			
		5	10	15	20
$N = 25$	OLS	58.14	86.17	114.64	145.35
	Householder	5.86	11.43	18.02	26.07
	Householder update	5.59	9.61	13.38	16.96
	Givens Gram-Schmidt update	6.23	8.79	10.10	11.32
	Cholesky	3.78	5.17	5.49	4.78
	Cholesky update	3.80	5.12	5.32	4.43
	Sweep	3.88	5.08	5.08	3.90
	Overhead	3.69	4.88	4.87	3.71
$N = 50$	OLS	60.29	87.33	115.85	148.70
	Householder	8.45	12.97	19.70	28.77
	Householder update	8.20	11.09	14.88	19.67
	Givens Gram-Schmidt update	8.83	10.24	11.43	13.65
	Cholesky	6.41	6.72	7.06	7.56
	Cholesky update	6.43	6.68	6.89	7.17
	Sweep	6.81	6.93	6.95	6.96
	Overhead	6.31	6.42	6.45	6.44
$N = 100$	OLS	64.57	91.72	120.01	151.44
	Householder	12.51	17.02	23.77	32.88
	Householder update	12.26	15.20	19.03	23.96
	Givens Gram-Schmidt update	12.85	14.17	15.38	17.57
	Cholesky	10.41	10.68	11.06	11.57
	Cholesky update	10.46	10.66	10.89	11.18
	Sweep	11.80	11.89	11.92	11.94
	Overhead	10.32	10.37	10.40	10.41