NORGES TEKNISK-NATURVITENSKAPELIGE
UNIVERSITET

A Fast and Exact Simulation Algorithm for General Gaussian Markov Random Fields

by

Håvard Rue

PREPRINT
STATISTICS NO. 2/1999

NORWEGIAN UNIVERSITY OF SCIENCE AND
TECHNOLOGY
TRONDHEIM, NORWAY

This report has URL http://www.math.ntnu.no/preprint/statistics/1999/S2-1999.ps
Håvard Rue has homepage: http://www.math.ntnu.no/~hrue
E-mail: hrue@math.ntnu.no
Address: Department of Mathematical Sciences, Norwegian University of Science and Technology, N-7491
Trondheim, Norway.

A Fast and Exact Simulation Algorithm for General Gaussian Markov Random Fields

HÅVARD RUE
DEPARTMENT OF MATHEMATICAL SCIENCES
NTNU, NORWAY

DECEMBER 28, 1998

SUMMARY

This paper presents a fast and exact simulation algorithm for a general Gaussian Markov Random Field defined on a lattice. For a 5×5 neighborhood, the algorithm has initialization cost of $32n^2$ flops where n is the number of sites in the lattice. Exact samples are then available using $16n\sqrt{n}$ flops, and conditioned samples on k constraints costs only $3nk$ extra flops. The algorithm is easy to generalize to non-regular lattices and runs very efficient on the computer.

The presented simulation algorithm makes general Gaussian Markov Random Fields computational tractable, both as a class of models to approach non-stationary problems including incorporating covariates in spatial models, and as a part in a hierarchical spatial Bayesian models analyzed using Markov chain Monte Carlo.

KEYWORDS: Conditional Autoregressive models, Gaussian Markov random field, Markov chain Monte Carlo.
ADDRESS FOR CORRESPONDENCE: H. Rue, Department of Mathematical Sciences, The Norwegian University for Science and Technology, N-7034 Trondheim.
E-MAIL: Havard.Rue@math.ntnu.no
ACKNOWLEDGMENTS: The author thanks Arne Martinsen, Håkon Tjelmeland and Erik Bølviken for fruitful and stimulating discussions, and the EU-TMR project on Spatial Statistics (ERB-FMRX-CT960095) for support and inspiration.

1 INTRODUCTION

Gaussian random fields (GRF) are the main ingredient in many areas in statistics often modeling some spatial varying variable. The variable is either of its own interest or part of a larger stochastic model which makes it necessary to simulate this field either unconditional or conditional on some observations. The dimensions of the GRF are usually so large that CPU costs and hence algorithms for simulation really matters. General GRF can be simulated by computing the Cholesky decomposition of the covariance matrix, a operation that require $n^3/3$ floating point operations (flops) where n is the number of sites in the field. For large n this is unrealistic.

This paper presents a fast and exact algorithm for simulating a general (and non-stationary) Gaussian random fields with a Markov property (GMRF), also known as conditional auto-regressions. Besag & Kooperberg (1995) give references to GMRF in various applications. The algorithm depends *only* on the conditional independence structure induced by the Markov property and none assumptions (apart from validity) are made on the interaction terms. On a $n_1 \times n_1$ lattice with n variables and a 5×5 neighborhood. After the initialization, exact and independent samples are available at cost $16n\sqrt{n}$ flops. Conditioning on k observed variables or linear combinations of variables, only costs $3nk$ extra flops.

The presented simulation algorithm makes general Gaussian Markov Random Fields computational tractable in many important applications. Although the use of non-stationary Gaussian fields (on a lattice) is not yet wide-spread, the full generality of the GMRF allowed in the algorithm, makes GMRF a natural class of models to approach non-stationary spatial problems. Further, the inclusion of external covariates into spatial models is troublesome, while in GMRF it can be introduced directly into the conditional densities while preserving its Markov property. Hence, the algorithm can still be used to produce exact unconditional and conditional samples. GMRF is also a natural candidate to use as a part in a hierarchical spatial Bayesian models analyzed using Markov chain Monte Carlo, see for example Wikle, Berliner & Cressie (1998). A fast and exact algorithm for simulating general GMRF will be of interest as it may allow (practically) for large block updates of the GMRF.

Regular lattices is not an necessary assumption for our algorithm, as it can be extend to non-regular lattices at essential the same computational cost. Further, we can compute the likelihood and normalization constant for regular and non-regular lattices at $\mathcal{O}(n^2)$ flops. The normalization constant is important in hierarchical Bayesian models, when the domain of interest vary or the (inverse) covariance matrix depend on hyperparameters or covariates.

The plan of the paper is as follows. After some preliminary definitions in Section 2, we present our algorithm in Section 3 and conditioned simulation in Section 4. Section 5 discuss extension to non-regular lattices including the evaluation of the likelihood and normalization constant and Section 6 presents a small example.

2 PRELIMINARIES

2.1 GAUSSIAN MARKOV RANDOM FIELDS

Define the Gaussian random field (GRF) $\boldsymbol{x} = \{x_{ij}, (i,j) \in \Lambda\}$ on the $n = n_1 \times n_1$ lattice Λ. Denote by $\pi(\boldsymbol{x})$ the density of $\boldsymbol{x}$ and $\boldsymbol{\mu}$ the expected value. The GRF has a Markov property iff

$$\pi(x_{ij} \mid x_{kl}, (k,l) \in \Lambda \backslash_{ij}) = \pi(x_{ij} \mid x_{kl}, (k,l) \in \partial_{ij})$$

where ∂_{ij} is the neighborhood of (i,j). A GRF with a Markov property is a Gaussian Markov random field (GMRF). Typical, ∂_{ij} is a set of neighboring sites, for example

$$\partial_{ij} = \{(k,l) \neq (i,j) \ : \ \|(k,l) - (i,j)\| \leq d\}$$

where $\|\cdot\|$ is some norm. Common neighborhoods in the nearest (in Euclidean norm) have 4, 8 and 12 sites, or the nearest 24 using the maximum-norm. We are mainly interested in the neighborhood defined as the 5×5 window centered at each site and assume this throughout the paper. Simplification in the algorithm exists for a 3×3 neighborhood and generalizations for larger neighborhoods.

Define $\boldsymbol{\Sigma}$ and $\boldsymbol{Q}$ as the covariance and precision matrix (the inverse covariance matrix) for $\boldsymbol{x}$, respectively, each of dimension $n \times n$. A Markov property of a GRF induce a band-structure in the precision matrix, as follows:

$$x_{ij} \perp x_{kl} \mid \boldsymbol{x} \backslash_{ij,kl} \quad \Longleftrightarrow \quad Q_{ij,kl} = 0, \tag{1}$$

meaning that x_{ij} and x_{kl} are conditional independent given the others iff the corresponding element in $\boldsymbol{Q}$ is zero. Thus for our neighborhood system, (1) implies that only 25 elements in $Q_{ij,kl}$ are non-zero for each ij, and these corresponds to x_{ij}, $x_{i+1,j}$, $x_{i+2,j}$, $x_{i,j-1}$ and $x_{i,j-2}$ and so on. Appropriate changes at the boundary is needed. Hence, $\boldsymbol{Q}$ is a sparse block pentadiagonal matrix.

To study conditional densities, split $\boldsymbol{x}$ into $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ where $\boldsymbol{x} = (\boldsymbol{x}_1^T, \boldsymbol{x}_2^T)^T$, $\boldsymbol{\mu} = (\boldsymbol{\mu}_1^T, \boldsymbol{\mu}_2^T)^T$, and divide $\boldsymbol{\Sigma}$ and $\boldsymbol{Q}$ into appropriate blocks

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \quad \text{and} \quad \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_{11} & \boldsymbol{Q}_{12} \\ \boldsymbol{Q}_{21} & \boldsymbol{Q}_{22} \end{bmatrix}.$$

The natural relation between conditional independence/dependences and the precision matrix becomes clear when we restate the classic result about the conditional density $\boldsymbol{x}_1 \mid \boldsymbol{x}_2$, in terms of the $\boldsymbol{Q}_{ij}$'s instead of the $\boldsymbol{\Sigma}_{ij}$'s. The following result seems to be less well known than it ought to be.

THEOREM 1 *The conditional density for $\boldsymbol{x}_1 \mid \boldsymbol{x}_2$ is Gaussian with mean $\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 - \boldsymbol{Q}_{11}^{-1}\boldsymbol{Q}_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)$ and precision $\boldsymbol{Q}_{1|2} = \boldsymbol{Q}_{11}$.*

The proof is simply to use the block equations derived from $\boldsymbol{Q}\boldsymbol{\Sigma} = \boldsymbol{I}$ and the standard results about conditional mean and variance.

2.2 Solving a Symmetric Block Pentadiagonal Linear System

Apart from Theorem 1 we will need in our algorithm fast linear solvers for solving symmetric block pentadiagonal linear systems of general size $nm \times nm$. We will however describe the simpler tridiagonal case as the pentadiagonal case is a straight forward extension. This is the field of numerics, so we only need to grasp the basic idea. Golub & van Loan (1996) provide background, details and algorithms.

Consider the following symmetric block tridiagonal linear problem $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$, where $\boldsymbol{A}$ is a $nm \times nm$ tridiagonal block matrix. The approach is to compute the block Cholesky decomposition of $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$, where

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{B}_1^T & & & \\ \boldsymbol{B}_1 & \boldsymbol{A}_2 & \boldsymbol{B}_2^T & & \\ & \ddots & \ddots & \ddots & \\ & & & & \boldsymbol{B}_{m-1}^T \\ & & & \boldsymbol{B}_{m-1} & \boldsymbol{A}_m \end{bmatrix} \quad \text{and} \quad \boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_1 & & & & \\ \boldsymbol{C}_1 & \boldsymbol{L}_2 & & & \\ & \ddots & \ddots & & \\ & & & & \\ & & & \boldsymbol{C}_{m-1} & \boldsymbol{L}_m \end{bmatrix} \tag{2}$$

Further, $\boldsymbol{A}_k$ is tridiagonal, $\boldsymbol{B}_k$ is diagonal, $\boldsymbol{C}_k$ is upper Hessenberg and $\boldsymbol{L}_k$ is lower triangular matrix. Each block has dimension $n \times n$ and there are m blocks. To compute $\{\boldsymbol{L}_k\}$ and $\{\boldsymbol{C}_k\}$, we expand $\boldsymbol{L}\boldsymbol{L}^T$ and obtain

$$\boldsymbol{A}_k = \boldsymbol{C}_{k-1}\boldsymbol{C}_{k-1}^T + \boldsymbol{L}_k\boldsymbol{L}_k^T, \quad \text{and} \quad \boldsymbol{B}_k^T = \boldsymbol{L}_k\boldsymbol{C}_k^T \tag{3}$$

for $\boldsymbol{k} = 1, \dots, m$, and where $\boldsymbol{C}_0 = \boldsymbol{0}$. Using (3), the algorithm goes as follows:

for $k = 1, \dots, m,$ do
  compute $\boldsymbol{L}_k$ from the Cholesky decomposition $\boldsymbol{L}_k\boldsymbol{L}_k^T = \boldsymbol{A}_k - \boldsymbol{C}_{k-1}\boldsymbol{C}_{k-1}^T$
  for column $i = 1, \dots, n,$ do
    solve $(\boldsymbol{C}_k^T)_i$ by forward substitution $\boldsymbol{L}_k(\boldsymbol{C}_k^T)_i = (\boldsymbol{B}_k^T)_i$.

When $\boldsymbol{L}$ is found, we can solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by using one forward and one back-substitution, $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$ and $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{y}$.

Our main objective is to solve a block pentadiagonal linear system, and compute the block Cholesky decomposition along the same lines as in the block tridiagonal case. A detailed flop count in the pentadiagonal case, give the cost of the block Cholesky decomposition of $\frac{7}{3}mn^3(1+o(1))$ flops, while the solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ require $8mn^2(1+o(1))$ flops. We can store $\boldsymbol{L}$ using about $2mn^2U(1+o(1))$ bytes, where U is the number of bytes to store one float. Further, note that $|\boldsymbol{A}| = |\boldsymbol{L}|^2$ and $|\boldsymbol{L}|$ is the product of the diagonal terms. Hence $|\boldsymbol{A}|$ is available in $\frac{7}{3}mn^3(1+o(1))$ flops. This has implications on the likelihood for GMRF, which we discuss shortly in Section 5.

3 THE ALGORITHM

Based on the preliminary results in Section 2 we can now derive our divide and conquer algorithm. We present first the basic procedure in Section 3.1, making the full algorithm in Section 3.2, and discuss repeated samples with a cost analysis in Section 3.3. To ease the presentation, define $\boldsymbol{x}_i$ as the ith column of $\boldsymbol{x}$ with mean $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_{ij}$ and $\boldsymbol{Q}_{ij}$ as the covariance and precision between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

3.1 THE BASIC PROCEDURE

Figure 1 illustrates our split and conquer approach to the problem. We do successive division of the lattice starting with the two middle columns c and $c+1$, where c is the integer value of $n_1/2$ when n_1 is odd. Let $\boldsymbol{x}_s = (\boldsymbol{x}_c^T, \boldsymbol{x}_{c+1}^T)^T$ denote the GMRF in the two columns. Define $\boldsymbol{x}_A = (\boldsymbol{x}_1^T, \dots, \boldsymbol{x}_{c-1}^T)^T$ and $\boldsymbol{x}_B = (\boldsymbol{x}_{c+2}^T, \dots, \boldsymbol{x}_n^T)^T$, so that $\boldsymbol{x} = (\boldsymbol{x}_A^T, \boldsymbol{x}_s^T, \boldsymbol{x}_B^T)^T$ and $\boldsymbol{\mu} = (\boldsymbol{\mu}_A^T, \boldsymbol{\mu}_s^T, \boldsymbol{\mu}_B^T)^T$. We split $\boldsymbol{Q}$ along the same lines

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_{AA} & \boldsymbol{Q}_{As} & \mathbf{0} \\ \boldsymbol{Q}_{sA} & \boldsymbol{Q}_{ss} & \boldsymbol{Q}_{sB} \\ \mathbf{0} & \boldsymbol{Q}_{Bs} & \boldsymbol{Q}_{ss} \end{bmatrix}.$$

Note that $\boldsymbol{Q}_{AB}$ is zero since $\boldsymbol{x}_s$ is a separating subset for $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$, $\boldsymbol{x}_A \perp \boldsymbol{x}_B \mid \boldsymbol{x}_s$. Assume for a moment that we can simulate $\boldsymbol{x}_s$ from its marginal density $\pi(\boldsymbol{x}_s)$. We will show later how to compute $\pi(\boldsymbol{x}_s)$ efficiently. A sample from the joint density $\pi(\boldsymbol{x})$ is then found by sampling $\boldsymbol{x}_A, \boldsymbol{x}_B \mid \boldsymbol{x}_s$ from the conditional density which is available from Theorem 1. The precision matrix and mean for $\boldsymbol{x}_A, \boldsymbol{x}_B \mid \boldsymbol{x}_s$ is

$$\boldsymbol{Q}_{AB|s} = \begin{bmatrix} \boldsymbol{Q}_{AA} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Q}_{BB} \end{bmatrix} \tag{4}$$

and

$$\boldsymbol{\mu}_{AB|s} = \begin{bmatrix} \boldsymbol{\mu}_{A|s} \\ \boldsymbol{\mu}_{B|s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_B \end{bmatrix} - \begin{bmatrix} \boldsymbol{Q}_{AA}^{-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Q}_{BB}^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_{As} \\ \boldsymbol{Q}_{Bs} \end{bmatrix} (\boldsymbol{x}_s - \boldsymbol{\mu}_s). \tag{5}$$

The important observations are the following.

- The conditional precisions equal $\boldsymbol{Q}_{AA}$ and $\boldsymbol{Q}_{BB}$ which are known already as sub-blocks of the $\boldsymbol{Q}$ matrix. Further, $\boldsymbol{Q}_{AA}$ and $\boldsymbol{Q}_{BB}$ has the same block pentadiagonal structure as $\boldsymbol{Q}$.
- The linear equation for the conditional mean (5) split into two separate equations, one for $\boldsymbol{\mu}_{A|s}$ and one for $\boldsymbol{\mu}_{B|s}$, since $\boldsymbol{Q}_{AB} = \mathbf{0}$.
- There is no need the invert $\boldsymbol{Q}_{AA}$ and $\boldsymbol{Q}_{BB}$ as we only need to compute $\boldsymbol{Q}_{AA}^{-1}$ times a vector, $\boldsymbol{y} = \boldsymbol{Q}_{AA}^{-1}\boldsymbol{z}$, say. This is done by solving the block pentadiagonal system $\boldsymbol{Q}_{AA}\boldsymbol{y} = \boldsymbol{z}$, see Section 2.2.

Equations (4) and (5) shows that *if* we are able to sample $\boldsymbol{x}_s$ fast, then we can compute the conditional mean and precision for $\boldsymbol{x}_{A|s}$ and $\boldsymbol{x}_{B|s}$ by solving two block pentadiagonal linear systems. Further, the conditional precision matrices $\boldsymbol{Q}_{A|s}$ $\boldsymbol{Q}_{B|s}$ have a block pentadiagonal structure and known as sub-blocks of $\boldsymbol{Q}$. We have now divided our problem into two separate smaller ones. Now we need to obtain the marginal density for $\boldsymbol{x}_s$. Since only the precision matrix $\boldsymbol{Q}$ is available, we need to compute the covariance matrix $\boldsymbol{\Sigma}_{ss}$ for $\boldsymbol{x}_s$. However, the block pentadiagonal structure of $\boldsymbol{Q}$ makes it possible to compute $\boldsymbol{\Sigma}_{ss}$ relatively fast. The idea is as follows: Since $\boldsymbol{Q}\boldsymbol{\Sigma} = \boldsymbol{I}$, we can write out the equations for block-column s in $\boldsymbol{\Sigma}$

$$\boldsymbol{Q}\begin{bmatrix}\boldsymbol{\Sigma}_{1s}\\ \vdots\\ \boldsymbol{\Sigma}_{ss}\\ \vdots\\ \boldsymbol{\Sigma}_{n_1 s}\end{bmatrix} = \begin{bmatrix}\boldsymbol{0}\\ \vdots\\ \boldsymbol{I}\\ \vdots\\ \boldsymbol{0}\end{bmatrix} \tag{6}$$

The right hand side is zero except for an $2n_1 \times 2n_1$ identity matrix $\boldsymbol{I}$. Note that (6) defines $2n_1$ block pentadiagonal linear equation systems of size $n \times n$. Hence, for the kth solution $\boldsymbol{y}^{(k)}$, column k of $\boldsymbol{\Sigma}_{ss}$ is available as $(y^{(k)}_{(c-1)n_1+1}, \cdots, y^{(k)}_{(c-1)n_1+2n_1})^T$. Computing $\boldsymbol{\Sigma}_{ss}$ costs $\mathcal{O}(n^2)$ flops. We can then simulate $\boldsymbol{x}_s$ by computing the Cholesky decomposition of $\boldsymbol{\Sigma}_{ss} = \boldsymbol{L}\boldsymbol{L}^T$ at the cost $\mathcal{O}(n\sqrt{n})$ flops, and then use the identity $\boldsymbol{x}_s \overset{d}{=} \boldsymbol{\mu}_s + \boldsymbol{L}\boldsymbol{z}$ where $\boldsymbol{z}$ is a $2n_1 \times 1$ vector of iid standard Gaussian variates.

3.2 The Full Algorithm

We are now able to present our full algorithm based on the basic procedure which goes as follows:

1. Solve the $2n_1$ block pentadiagonal linear equation systems in (6) to obtain $\boldsymbol{\Sigma}_{ss}$.
2. Compute the Cholesky decomposition of $\boldsymbol{\Sigma}_{ss} = \boldsymbol{L}\boldsymbol{L}^T$ and simulate $\boldsymbol{x}_s \overset{d}{=} \boldsymbol{\mu}_s + \boldsymbol{L}\boldsymbol{z}$ where $\boldsymbol{z}$ is a vector of iid standard Gaussian variables.
3. Solve the two block pentadiagonal linear equation systems in (5) to obtain $\boldsymbol{\mu}_{A|s}$ and $\boldsymbol{\mu}_{B|s}$.

Our $n_1 \times n_1$ problem is now divided into two separate problems of size $n_1 \times c-1$ and $n_1 \times n_1 - c - 1$, *identical* to our original one. Note that all dependency of $\boldsymbol{x}_s$ is included in $\boldsymbol{\mu}_{A|s}$ and $\boldsymbol{\mu}_{A|s}$, so that $\boldsymbol{x}_s$ is not used later in the algorithm. We can therefore use our algorithm again at each of the two separate problems and proceed similarly by each time dividing the largest size (either a row or a column) until either the size is so small that it can not be divided into two smaller ones (i.e. 1×1, 1×2, ..., 3×3), or it is more efficient to do a direct simulation. At this final stage we compute the covariance matrix, the Cholesky decomposition and then simulate by the classic procedure. Figure 1 illustrate the first three steps in the algorithm. After we have finished the A part in Figure 1, we continue with the B-part in the same manner. Thus our algorithm is complete.

3.3 Repeated Samples and Cost analysis

The algorithm in Section 3.2 provides a significant speedup for generating repeated samples from the same density. To take advantage of this, we divide our algorithm into an *initialization*-part and an *operation*-part.

Initialization-part This part of the algorithm is done once only for each density from which we want to sample. It consists of the part of the algorithm in Section 3.2 that does not depend on the random numbers used. In the initialization part we compute the block Cholesky decomposition of all block pentadiagonal linear systems, solve the linear systems to obtain $\boldsymbol{\Sigma}_{ss}$'s and compute their Cholesky decompositions. We compute also the Cholesky decomposition of the final. All (block) Cholesky decomposition are then stored.

Operation-part This part of the algorithm generate samples from the target density. Since all Cholesky decompositions are known from the initialization-part, we need only sample $\boldsymbol{x}_s$ and then solve (by one forward and one backward substitution) the two linear systems to obtain $\boldsymbol{\mu}_{A|s}$ and $\boldsymbol{\mu}_{B|s}$, and additionally sample the blocks at the final stage.

Cost Analysis A detailed cost analysis of the number of flops required in the initialization and operation-part of the algorithm, shows that

$$\begin{aligned}\text{Initialization-part} &= 32n^2(1+o(1)) \text{ flops}\\ \text{Operation-part} &= 16n\sqrt{n}(1+o(1)) \text{ flops.}\end{aligned}$$

$4n\sqrt{n}U(1+o(1))$ bytes, where U is the number of bytes to store one float. Section 6 provide some concrete numbers as CPU seconds for my current implementation of the algorithm.

4 Conditional Simulation

This section describes how we can use the algorithm in Section 3 to generate conditional samples.

We consider the general case where we want to generate samples from the target density condition on $\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon} = \boldsymbol{b}$ where $\boldsymbol{A}$ is a $k \times n$ matrix with rank k. The Gaussian noise is denoted by $\boldsymbol{\epsilon}$ and has zero mean and known covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$. Introduce the new variable $\widetilde{\boldsymbol{x}} = (\widetilde{\boldsymbol{x}}_1^T, \widetilde{\boldsymbol{x}}_2^T)^T = (\boldsymbol{x}^T, (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon})^T)^T$ of dimension $n+k$. Let $\widetilde{\boldsymbol{\Sigma}}$ denote the covariance matrix of $\widetilde{\boldsymbol{x}}$. Simulating from the unconditional density of $\widetilde{\boldsymbol{x}}$ is simply to simulate $\boldsymbol{x}$ and $\boldsymbol{\epsilon}$ unconditionally, and then use $\widetilde{\boldsymbol{x}} = (\boldsymbol{x}^T, (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon})^T)^T$. A sample of $\widetilde{\boldsymbol{x}}_1 \mid \widetilde{\boldsymbol{x}}_2 = \boldsymbol{b}$ can be obtained using the following identity

$$\widetilde{\boldsymbol{x}}_1 \mid \widetilde{\boldsymbol{x}}_2 = \boldsymbol{b} \quad \overset{d}{=} \quad \widetilde{\boldsymbol{x}}_1 - \widetilde{\boldsymbol{\Sigma}}_{12}\widetilde{\boldsymbol{\Sigma}}_{22}^{-1}(\widetilde{\boldsymbol{x}}_2 - \boldsymbol{b}). \tag{7}$$

A direct calculation will show that (7) is correct. Eq. (7) is commonly refered to as conditioning using Kriging, see Cressie (1993, Sec. 3.6.2). By expanding the right hand side of (7), we obtain

$$\boldsymbol{x} - \boldsymbol{Q}^{-1}\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{A}^T + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}\right)^{-1}(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon} - \boldsymbol{b})\,. \tag{8}$$

We need to compute $\boldsymbol{v} = \boldsymbol{Q}^{-1}\boldsymbol{A}^T$, which requires solving $\boldsymbol{Q}\boldsymbol{v}_i = (\boldsymbol{A}^T)_i$ for each of the k columns. Then the remaining parts of (8) are straight forward. We can save computation by dividing the conditioning step into an initialization-part and operation-part. In the initialization-part we compute $\boldsymbol{v}$ and then $\boldsymbol{A}v$, which costs $8kn\sqrt{n}$ flops (since the block Cholesky decomposition of $\boldsymbol{Q}$ is already available). Inverting the symmetric $k \times k$ matrix costs k^3 flops, so in total we use $8kn\sqrt{n} + k^3$ flops. The operation-part is only to calculate $\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon} - \boldsymbol{b}$, pre-multiply with the matrix found in the initialization, negate and add $\boldsymbol{x}$, which costs $3nk$ flops.

5 GENERALIZATIONS

We will now discuss some further generalizations of the presented algorithm, concerning neighborhood, general lattices, likelihood and normalization constants.

NEIGHBORHOOD Extending the neighborhood to say a 7×7 window require a block heptadiagonal solver and $\boldsymbol{x}_s = (\boldsymbol{x}_{c-1}^T, \boldsymbol{x}_c^T, \boldsymbol{x}_{c+1}^T)^T$, again a straight forward extension of the block pentadiagonal case. The cost analysis give new constants, but the rate is still the same.

LIKELIHOOD Likelihood based inference and testing, are easily available for general GMRF. The term $\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ is available in $\mathcal{O}(n)$ flops while the computation of the determinant $|\boldsymbol{Q}|$ costs $\frac{7}{3}n^2$ flops. However, it is a major improvement to the general case requiring $\mathcal{O}(n^3)$ flops.

GENERAL LATTICE We made the basic assumption that $\boldsymbol{x}$ is defined on the $n_1 \times n_2$ lattice Λ. Let us now relax this assumption. Suppose the region of interest is the non-regular lattice Λ. Since Λ is not rectangular, we will not obtain the nice block pentadiagonal structure needed for the algorithm to work. The solution to this is simple. Let Λ' be the smallest rectangular lattice covering Λ. Extend $\boldsymbol{x}$ to $\widetilde{\boldsymbol{x}}$ by adding independent standard Gaussian variates for each site in $\Lambda' - \Lambda$. The precision matrix for $\widetilde{\boldsymbol{x}}$, $\widetilde{\boldsymbol{Q}}$, has now the nice block pentadiagonal structure needed for the algorithm, although some of the off-diagonal terms are zero. Our algorithm (including conditioning) now applies and the subset of $\widetilde{\boldsymbol{x}}$ in Λ has the desired marginal density.

NORMALIZATION CONSTANT Using the same argument as in the previous paragraph, we obtain that $|\boldsymbol{Q}| = |\widetilde{\boldsymbol{Q}}|$ since the added independent variates in $\Lambda' - \Lambda$ have unit precision. We are then able to compute in $(n')^2$ flops (where $n' = |\Lambda'|$) the normalization constant and likelihood for general

GMRF on an arbitrary lattice Λ. This is important. In spatial hierarchical models, the domain of interest Λ or the precision matrix $\boldsymbol{Q}$ can depend on a hyper-parameter or covariate ξ. Such models are usually analyzed using a Markov chain Monte Carlo algorithm, where ξ is random and the ratio of the normalization constants for the GMRF is needed to compute the acceptance rate. This ratio required two computations of the determinant, each costs $\frac{7}{3}n^2$ flops.

6 An Example

To give an idea of the CPU costs for the algorithm, I will present the results for my current implementation of the algorithm on a 100×100 lattice ran on a Sun Ultra 2 Model 1296 with a 296 MHz SUNW UltraSPARC-II processor.

With a 3×3 neighborhood the algorithm used 6.8 seconds CPU in the initialization-part and produced then iid samples at the cost of 0.43 seconds CPU in the operation-part. When I increased the neighborhood to 5×5, the algorithm used 23.0 and 0.51 seconds, respectively. The results for a 7×7 neighborhood was 52.7 and 0.61 seconds. To compute the normalization constants, the algorithm used 0.72, 2.5 and 5.5 seconds CPU depending on the neighborhood.

Then I added 100 linear conditions requiring that the sum of each of the 100 columns is zero. The initialization-part of the conditioning took 15.0, 22.6 and 31.4 seconds for a 3×3, 5×5 and 7×7 neighborhood, respectively. The corresponding numbers for the operation-part were all 0.08. Figure 2 shows one of the conditioned samples using a 7×7 neighborhood, where I (for simplicity only) set all the 48 off-diagonal elements to -1 and the diagonal elements to 48.0005 to ensure positive definiteness.

My current implementation makes about 136 (initialization-part) and 31 (operation-part) Megaflops on a 5×5 neighborhood, which is fast. The core of the implementation are calls to efficient Lapack-routines written in Fortran (Anderson et al. 1995) and this part is totally dominating in the initialization-part. In the operation-part, 50% of the CPU is used to more slow bookeeping and setup tasks, and hence the speed decrease. To compare the numbers, the plain loop $y_i = y_i + \beta x_i$, makes on the same computer 180 and 195 Megaflops in C and Fortran compiled with SUN compiliers using options `-fast -xO5`, respectively.

I end this paper by pointing out that Lavine (1998) also reports an $\mathcal{O}(n^2)$ (unconditional) simulation algorithm for GMRF with 4 nearest neighbors and invariant interactions (the elements in the $\boldsymbol{Q}$-matrix) by running a special Bayesian dynamic model. However, for his case we can do even better by making use of the fast block-tridiagonal solver of Swarztrauber (1974) and derive (essentially) an $\mathcal{O}(n\log(n))$ simulation algorithm. However, special situations in terms of values of the interactions are usually of less interest compared to a general algorithm.

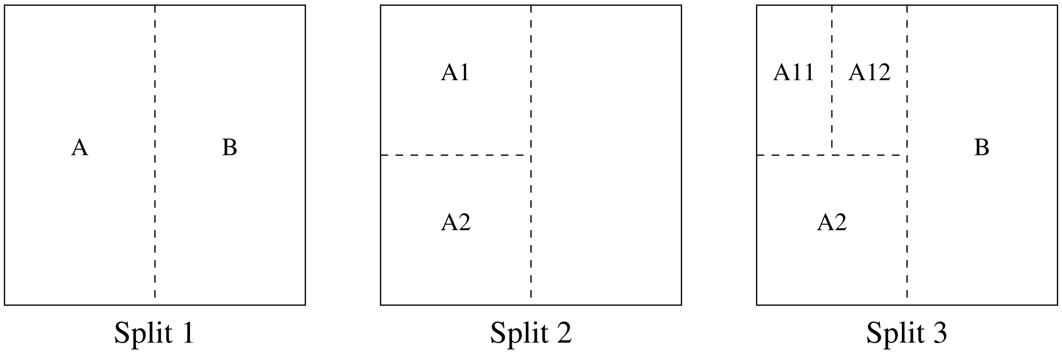

FIGURE 1: The three first step in the algorithm.

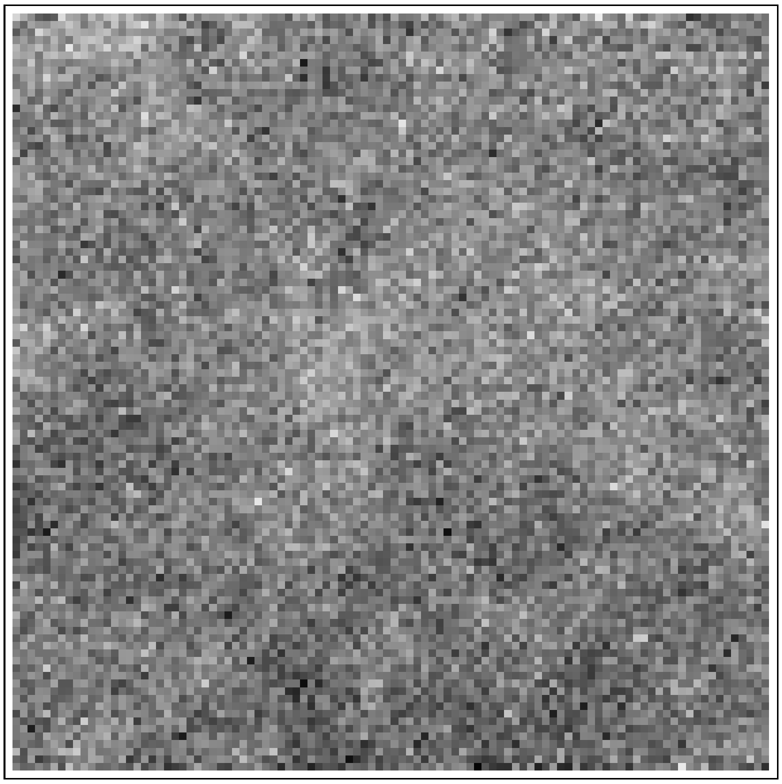

FIGURE 2: A exact sample from a 100×100 GMRF with a 7×7 neighborhood, conditioned on each of the 100 columns sum up to zero.