NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

Parallel sampling of GMRFs and geostatistical GMRF models

by

Ingelin Steinsland

PREPRINT
STATISTICS NO. 7/2003

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
TRONDHEIM, NORWAY

This report has URL http://www.math.ntnu.no/preprint/statistics/2003/S7-2003.ps
Ingelin Steinsland has homepage: http://www.math.ntnu.no/∼ingelins
E-mail: ingelins@stat.ntnu.no
Address: Department of Mathematical Sciences, Norwegian University of Science and Technology, N-7491 Trondheim, Norway.

Parallel sampling of GMRFs and geostatistical GMRF models

Ingelin Steinsland

Norwegian University of Science and Technology
and
Trinity College Dublin

Abstract

In this report the main focuses are geostatistical Gaussian Markov random field (GMRF) models and parallel exact sampling of GMRFs. There are also brief overviews of parallel computing and Markov chain Monte Carlo (MCMC) methods, and a literature review of parallel MCMC. The geostatistical GMRF models are constructed by discretising the domain region using a lattice. Instead of giving this lattice a Gaussian random field prior, that corresponds to a Gaussian process, a GMRF that is an approximation to the GRF is chosen. More computational benefits are achieved through the nice parallelisation possibilities of GMRF sampling end evaluation. The computationally expensive part of GMRF sampling is Choleskey decomposition of the precision matrix. Parallelisation is done with parallel algorithms from linear algebra for sparse symmetric positive definite matrices. The parallel GMRF sampler is tested for graphs and lattices, and gives both good speed-up and good scalability.

A parallel one-block updating scheme Metropolis-Hastings sampler for latent GMRF models is constructed using a GMRF approximation to $\pi(x|y, \theta)$ as proposal for the latent field. It is used for a geostatistical GMRF model with binomial likelihood, and shows good mixing for both the latent field and the hyper-parameters, as well as good speed-up from the parallelisation.

1 Introduction

More powerful computers allow us to use larger and -hopefully- more powerful statistical models. The development of computers over the last couple of decades has given us the opportunity to fit models that are not analytically tractable. The Bayesian approach to modelling, especially, has gained from these developments. Through the computational expensive Markov chain Monte Carlo (MCMC) sampling methods it is possible to make inferences given "any" model, see e.g. [35], [19] or [27]. But models and sampling methods can always benefit from more computational resources. More computational resources can be achieved by using two or more state-of-the-art processing units. This leads naturally towards parallel computing. In particular spatial statistics is extremely demanding with respect to both CPU and memory resources. This report focuses on using parallel computing to geostatistical models. In geostatistics we have observations with known position. The observations are somehow connected to an underlying (latent) Gaussian surface, and we want to make inferences about this surface.

This chapter introduces a geostatistical problem, the model-based geostatistics and how Gaussian Markov random field (GMRFs) can be used in geostatistics. The next chapter gives a brief introduction to parallel computing, and chapter 3 contains a brief introduction to MCMC. Chapter 4 reviews some parallel MCMC algorithms known in the literature. In chapter 5 we introduce and test a parallelisation of exact sampling of Gaussian Markov random fields. Chapter 6 contains a case study of the problem introduced in chapter 1.1. Inference is done using a GMRF approach with the parallel sampling method from chapter 5. The report is closed with a discussion in chapter 7.

1.1 A geostatistical problem: Campylobacter infections in north Lancaster

As a motivation to geostatistical models, we introduce an example of a geostatistical problem first of all. We have chosen the campylobacter infections in north Lancaster dataset. This dataset consists of the dates and the unit post-code location of all reported outbreaks of campylobacter, salmonella and cryptosperidia between April 29th and December 31st 1994 in post-code sectors LA8, LA9, LA10, LA21, LA22 and LA23. The dataset has previously been analysed in [11]. The locations of the outbreaks are shown in figure 1, and large areas without any outbreaks can be observed.

Two or more persons can get the disease from the same source infection, and we think of this as one outbreak. Cases in the same post-code region with dates of onset within five days of each other are therefore considered one case. The dataset now consists of 399 outbreaks in 236 different locations, and 234 of the cases are campylobacter. We want to estimate the probability that an enteric outbreak is campylobacter. The data are extra binomial, and this could come from spatial variations of outbreaks. We assume there is an underlying latent relative risk surface that an outbreak is campylobacter and it is this surface we want to estimate.

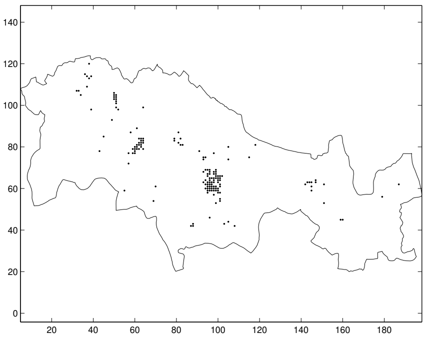

Figure 1: *The locations of the enteric outbreaks*

1.2 Model based geostatistics

Geostatistics is the part of spatial statistics that deals with continuous spatial variation. The phenomena to be study, $s(x)$, is a surface defined on a region $A \subset R^2$. The phenomena is generally not directly observable, and can be modelled as a realisation of a *stochastic process* $S = \{S(x) : x \in A\}$. An example of a geostatistical problem is to estimate a geological horizon when it is only observed through wells at a few location. Then $s(x)$ denotes the depth of the horizon at location x. An other example is the case introduced in section 1.1. Here $s(x)$ is a relative risk surface. The available data are indirect observations of $S(x_i)$, also named measurements $Y_1, Y_2, \ldots, Y_n$ taken at locations $x_1, x_2, \ldots, x_n$, where $x_i \in A$, $i = 1, 2, \ldots, n$. The measurements are often noisy versions of $s(x)$, e.g. $Y_i \sim N(s(x_i), \sigma^2_{noise})$, but can also have other connections, e.g. for count data can $Y_i \sim Po(\exp(s(x_i)))$, Poisson distributed with mean $\exp(s(x_i))$. The locations $x_1, \ldots, x_n$ are considered either fixed or sampled independent of $S(x)$ throughout this report. The term *model based geostatistics* was introduced in [11] to be the branch of geostatistics that apply *explicit parametric stochastic models and formal likelihood-based methods of inference to geostatistical problems* [12]. Modeled-based geostatistics can be used both in a classical and a Bayesian framework. We will only focus on the Bayesian approach. For a more detailed introduction to model-based geostatistics see [12].

Our data are of the form (x_i, y_i) for $i = 1, \ldots, n$, where $x_i \in A$ are locations and $y_1, \ldots, y_n$ observations associated with these locations. The observations are from a *measurement process* $Y(x)$. We assume there exists an unobservable stochastic process $S(x)$, the *signal process*, and a relation between $S(x)$ and $Y(x)$, e.g. that $Y(x_i)$ is a noisy version of $S(x_i)$. The joint distribution of the signal process and the measurement process, $\pi_{S,Y}$ can be specified through the marginal distribution of S and the conditioned distribution of Y given S;

$$\pi_{S,Y|\theta} = \pi_{Y|S,\theta} \pi_{S|\theta} \pi_\theta$$

where θ are hyper-parameters that will appear. For notation reasons θ is sometimes suppressed when conditioned on. We have to model π_S, the distribution of the signal process,

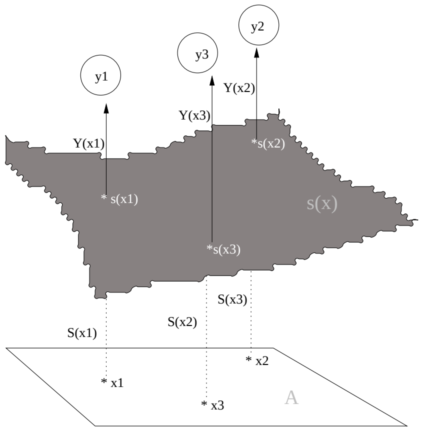

Figure 2: *Illustration of the geostatistical setting. The surface of interest is $s(x)$ which exists over a region A. In this illustration there are three observations; y_1, y_2 and y_3, at locations x_1, x_2 and x_3. The observations come from a measurement process Y which most often is modelled as a stochastic function of the signal surface at the observation location; $Y(x_i) = Y(s(x_i))$.*

and $\pi_{Y|S}$, the conditional distribution of the observations given the signal process. Two common assumptions are 1) that S is a *Gaussian process* and 2) that conditioned on the underlying process S the observations are independent. For an illustration see figure 2.

1.2.1 Gaussian modelling of the signal process

We model S to be a stationary Gaussian process with expectation $\mathrm{E}(S(x)) = \mu$, variance $\mathrm{Var}(S(x)) = \sigma_S^2$ and correlation function $\mathrm{Corr}(S(x), S(x')) = \rho(u)$, where u is the Euclidean distance between x and x', $\|x - x'\|$.

The correlation function $\rho(u)$ has to be specified next. The smoothness of the signal process in the Gaussian case is determined by $\rho(u)$'s properties at $u = 0$. A process $S()$ is *mean square continuous* if $\mathrm{E}[(S(x) - S(x'))^2] \to 0$ as $\|x - x'\| \to 0$ for all x. Further is $S()$ *mean square differentiable* if there exists a process $S'()$ such that for all x

$$E\left[\left(\frac{S(x) - S(x')}{\|x - x'\|} - S'(x)\right)^2\right] \to 0$$

as $\|x - x'\| \to 0$. Note that continuously and differentiability in mean square do not imply continuously and differentiability in the realisations. A popular family to choose a correlation function from is the Matérn family,

$$\rho(u) = (\frac{u}{\phi})^\kappa \frac{1}{2^{\kappa-1}\Gamma(\kappa)} K_\kappa(u/\phi)$$

where $\kappa > 0$ and $\phi > 0$ are parameters, and K_κ denotes a Bessel function of order κ. A nice property of the Matérn family is that it makes the process $S()$ $\lceil \kappa - 1$ times mean square differentiable. Here $\lceil \kappa$ denotes the largest integer less or equal to κ. This makes κ a smoothness parameter. Two well known members of the family are the *exponential correlation function* $\rho(u) = \exp(-u/\phi)$ (set $\kappa = 0.5$), and the *Gaussian* or *squared exponential correlation function* $\rho(u) = \exp(-(u/\tilde{\phi})^2)$ ($\kappa \to \infty$ and $\phi = \tilde{\phi}/2\sqrt{u+1}$).

Another correlation function family is the powered exponential,

$$\rho(u) = \exp(-(\frac{u}{\phi})^\kappa)$$

where $\phi > 0$ and $0 < \kappa \leq 2$.

A property for both the Mantérn and the exponential families is that

$$\rho(u; \phi, \kappa) = \rho(\frac{u}{\phi}; 1, \kappa)$$

Hence ϕ can be interpreted as a scaling parameter for the dependence. The effective

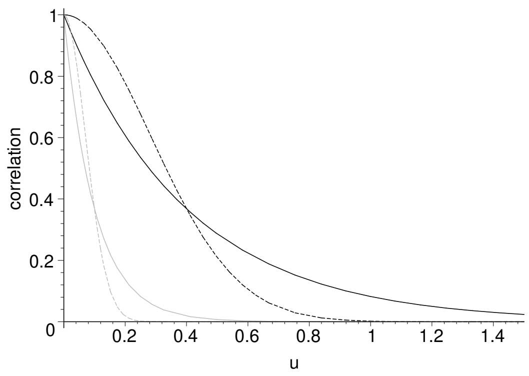

Figure 3: *The powered exponential correlation function for* $\phi = 0.1$ *(in gray) with* $\kappa = 1$ *(solid line) and* $\kappa = 2$ *(doted line) and* $\phi = 0.4$ *(in black) with* $\kappa = 1$ *(solid line) and* $\kappa = 2$ *(dashed line).*

range of correlation, or simply the *range* r, is defined as the distance where the correlation drops below 0.05. For the exponential correlation function $r = \phi/3$. In both models κ is a smoothness parameter. Figure 3 shows the exponential and the Gaussian correlation function with range parameters $\phi = 0.1$ and $\phi = 0.4$. In sparse and noisy cases it is often not possible to identify κ, and κ has to be fixed. Thus three hyper-parameters have arisen; μ, ϕ and, if not fixed, κ. For these a prior $\pi(\theta)$ ($\theta = (\mu, \phi, \kappa)$) most be specified.

1.2.2 Modelling the conditional measure processes given the signal process

We have just established a Gaussian model for the signal process S. In the most popular form of Gaussian geostatistical models also $Y|S$ is modelled as Gaussian. Let Y conditioned on S be mutually independent, and set

$$Y_i|S \sim N(S(x_i), \sigma_Y^2)$$

for $i = 1, \ldots, n$. A measurement at location x_i, Y_i, is Gaussian with expected value the signal process at the same location and variance σ_Y^2. An equivalent formulation is

$$Y_i = S(x_i) + Z_i$$

for $i = 1, \ldots, n$, where $Z_1, \ldots, Z_n$ are independent identical Gaussian distributed $Z_i \sim N(0, \sigma_Y^2)$. The marginal distribution of Y is then multivariate Gaussian;

$$Y \sim N(\mu\mathbf{1}, \sigma_S^2 R + \sigma_Y^2 I)$$

where $\mathbf{1}$ is a row vector of size n consisting of ones and R is a $n \times n$ matrix with $R_{ij} = \rho(\|x_i - x_j\|)$. We have introduced a new hyper-parameter here, the noise-variance σ_y^2, which needs a prior $\pi_{\sigma_y^2}$.

Often it is unnatural to model the observations as a noisy version of a latent Gaussian process S. We want to be able to model the expectation of Y_i as a function of $S(x)$ and possible some known covariates. This can be formalised as a *generalised linear spatial model* (GLSM), see chapter 2.9 in [12]. We will describe two models, one for count data and one for binomial data.

Often count data follow a Poisson distribution;

$$Y_i|S \sim Po(\lambda(x_i)))$$

where $\lambda(x_i) = c_i \exp(S(x_i))$ and c_i is a known positive location specific constant. If data are obtained from counting over different time periods c_i can be the time spend for Y_i. In a setting where y_i is the number of cases of a disease in an area i, c_i can be related to the population of area i.

Spatial binomial data comes in triplets (x_i, y_i, n_i), where x_i is the location, y_i the number of successes out of n_i Bernoulli trails with probability of success $p_i = p(x_i)$. We may use a probit link between $p(x_i)$ and $S(x_i)$,

$$\log(\frac{p(x_i)}{1 + p(x_i)}) = S(x_i)$$

and

$$Y_i|S \sim \mathrm{Bin}(n_i, p(S(x_i)))$$

The case introduced in section 1.1 is an example with spatial binomial data. The observations consist of a post-code location (x_i), the number of enteric infections (n_i) and how many of these that are campylobacter (y_i).

The spatial count data model and the spatial binomial data model specified above are both generalised linear spatial models, and results for GLSM are valid. There are no new hyper-parameters introduced in these two models. Though from an implementation point of view, we can move the expected value from the Gaussian process S to our measurement model. In the spatial binomial model we assume $S(x)$ is a zero mean Gaussian process and introduce a location independent level β

$$\log(\frac{p(x)}{1+p(x)}) = S(x) + \beta$$

in the measurement model.

1.2.3 Estimation

We have now build geostatistical models, and want to use these for estimation. Our interest are one or both of the latent process and some of the hyper-parameters θ describing our model. We may also have an interest in the signal process $S(x)$ for some specified locations. But more often the interest is in the process itself. It is not possible to find a closed form of $\pi_{S,\theta|Y}$. Instead typically a fine grid over our study region is used as the locations of interest denoted z_j for $j = 1, \ldots, N$, see figure 4. To make inference we sample from $\pi_{S,\theta|Y}$

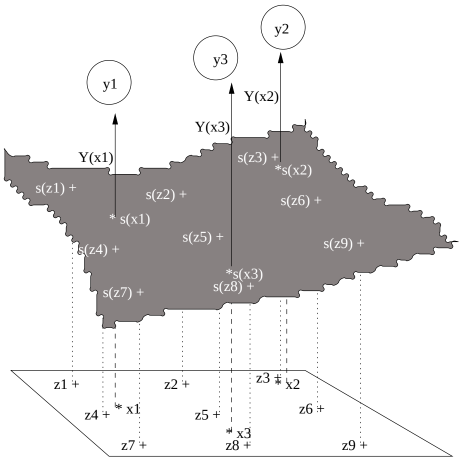

Figure 4: *Grid of locations used for estimation. The signal process S is typically estimated on a grid of locations. Here the locations are $z_1, z_2, \ldots, z_9$, and $S(x)|y$ is estimated for $s(z_1), s(z_2), \ldots, s(z_9)$*

using Markov chain Monte Carlo (MCMC). For an introduction to MCMC see chapter 3. The simulation process is often computationally very expensive. In the next section we are going to introduce a small change in the model of $\pi_{S|\theta}$. Under the new model computations needed, such as; evaluation of the posterior, to find an approximation to the posterior (for fixed hyper-parameters) and to sample from this approximation, are cheaper. The change in model will therefore enable us to do fast simulations with an one-block updating scheme

Metropolis-Hastings algorithm. For details of the simulation method see chapter 5 and chapter 6.

1.3 Using Gaussian Markov random fields in geostatistics

The computational demanding simulation methods needed to evaluate the geostatistical models described above restrict both the dimension of problems we are able to evaluate, and the quality of the analysis. This section takes a more pragmatic approach when modelling. We have in mind the purpose of our modelling, estimation, and the restrictions of computationally expensive simulation for inference. We build a model that gives very similar estimators to the models described above, but which is orders faster to do simulations for. What we gain is speed-up and the possibility of evaluating larger models. The focus of this section is the grid of estimation points. Index the grid points $j = 1, \ldots, N$ where N is the number of grid points. Denote the grid locations $(z_1, \ldots, z_N)$ and the signal process at these locations $S = (S_1, \ldots, S_N)$. If $S(x)$ is a Gaussian process with mean μ, variance σ_S^2 and correlation function $\rho(u)$ then S is multivariate Gaussian

$$S \sim N(\mu\mathbf{1}, \sigma_S^2 R)$$

where R is a $N \times N$ correlation matrix and $R_{ij} = \rho(\|x_i - x_j\|)$. Calculations involved in simulations are expensive because both R and its inverse generally are full matrices. Multivariate Gaussian distribution is also known as *Gaussian random field* (GRF). A special version of Gaussian random fields is *Gaussian Markov random fields* (GMRFs). For GMRFs the inverse of R is sparse which makes the necessary computations cheaper. This would be beneficial for us. For an introduction to GMRFs see chapter 5. On a grid it is possible to make a GMRF that is a good approximation to a GRF for most commonly used correlation function, see [38]. We are able to approximate the Gaussian random field of the estimation grid to a Gaussian Markov random field, but what about the measure process Y? The approximation is for a regular grid (lattice) only and the locations of the observations generally do not fit into a regular grid. The pragmatic solution to this problem is to use the estimation grid only and "move the location" of each observation to its nearest grid point. If the grid is fine relative to the distances between locations this approximation only introduces a minor change.

A formal way of doing the convenient change above is to remodel the underlying process S. Instead of having a continuous process $S(x)$ we now discretise it by placing a lattice above the domain A, see figure 5.

The lattice has a grid point in the centre of each element. Each element is viewed as an unit, and is assumed to have a constant value for the process S. The elements are indexed $j = 1, \ldots, M$, their centre locations (the grid points) $z_1, \ldots, z_M$ and the latent signals $S_1, \ldots S_M$. The model for S has changed from a process $\{S(x) : x \in A\}$ to a field $S = [S_1, S_2, \ldots, S_M]^T$. The field S is now modelled as a GMRF that is an approximation to the GRF with correlation function $\rho(\|z_i - z_j\|)$. Here z_i and z_j are the centre locations of element i and j. The conditional distribution of the measurements given the signal field,

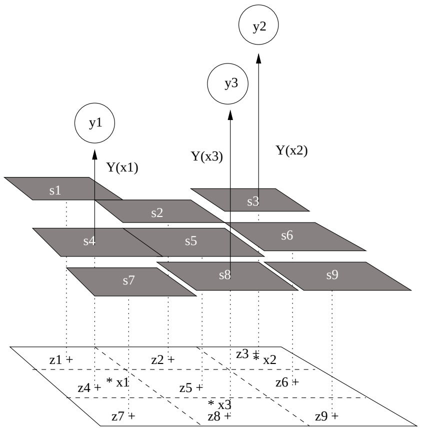

Figure 5: *Illustration of the GMRF model. The area A is divided into blocks, here 9 blocks. The surface S is modelled as constant on a block, i.e. it only has 9 levels, $s_1, s_2, \ldots, s_9$.*

$Y|S$ is modelled as before. A measurement Y_i is now connected to the field variable S_j where element j covers Y_i's location.

We have changed the model for the signal surface S from being a constantly changing surface to be build up of small square elements of constant values. This seems to be a conceptual change, but in our setting we are only able to estimate the signal process for a discrete grid in any case. If we use a GRF model and the observation locations are a subset of the grid we have the same estimators. For computation reasons we use a GMRF that is an approximation to the GRF. Further the observation locations are most often not a subset of the grid, and the discretion of the signal model will influence the estimators through the observations.

2 Parallel Computing - A brief introduction

A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem. [15]

Two familiar problems for almost every computer user are tasks that take computers ages to finish, and, even worse, can not be solved at all because of insufficient memory. So our computational problem remains unsolved or at least can not be solved within a reasonable time. But if a house can not be built quick enough by one builder, maybe two or more can. One builder may not be capable at building the house alone, there could be items too heavy for him. But two or more of them can lift the items together. The house will be more expensive then first thought, but it is now possible to build it. The same can be done with computers. Most computational problems can be divided up such that more computers can work on it at the same time. This is known as parallel computing. The problem of "slow" computers and computers without the required memory is as old as computers them-selves. Since the 1960s parallel computing has been a significant branch of computer science. But it was not until the late 1980s that software and hardware had improved so much it could be used in large scale outside its own academic branch and in industry. Much has been written about parallel computing, for an introduction see e.g. [15] or [10].

Parallel computer architecture can be divided into abstraction levels, see figure 6. At the bottom we have the hardware, the physical components a computer consist of and the communication medium. The bridge between hardware and the user is the operation system, and for parallel computer this must support interaction between computers. Above the operation system there are libraries and compilers for communication. These again are used under a certain programming model. At the top is our parallel program. We will focus on the higher levels, only describing programming models, design and performance analysis of parallel applications in some detail. But to understand this, we need some basic knowledge about how a computer works. For a more detailed introduction see e.g. [10].

2.1 Hardware

The main components in a computer are the processing unit, the memory and the connection between them, the bus. There are different types of memory, generally speaking it is along the line of quick expensive memory to slow cheap ones. To take advantage of both the speed that quick memory can give and the quanta the cheap affords us to have, most computers have a memory hierarchy, see figure 7. The quickest memory is the register. It is a part of the CPU and stores instructions and data about to be processed. The cache is a layer between the ordinary local memory with slower access and the register. It has a much quicker access time than the local memory, and is used for storage of frequently used data or instructions. The cache is used for storing data close in time and space. Often addresses just accessed and close addresses are asked for. In modern computers the cache is often divided in two layers, with a smaller quicker and more expensive closest to the

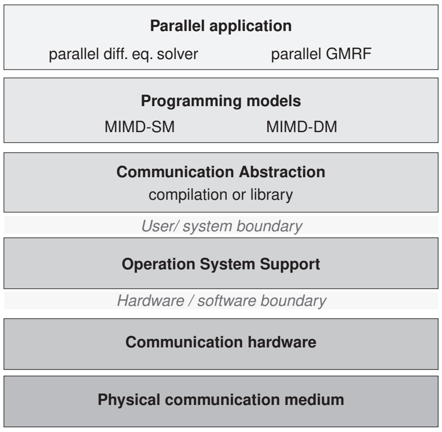

Figure 6: *Diagram of abstraction levels in parallel computer architecture.*

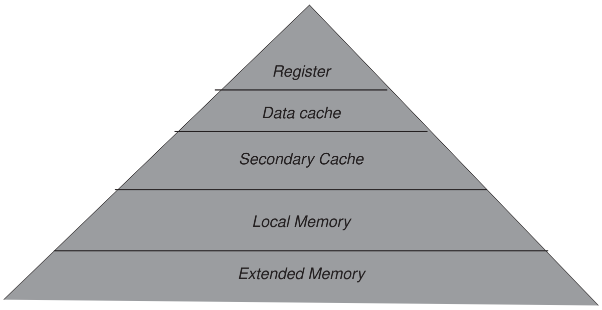

Figure 7: *Diagram of memory hierarchy.*

register. Data will only be sent to the processor from the register, and memory is passed to and from the register via the cache as needed. Most operation systems will not allow us to control the memory hierarchy, but optimise it itself. Sometimes we can help the compiler, i.e. by accessing data that are close at the same time. We can get speed-up (or slow-down) by changing the order of a nested loop. If we gain speed-up, the new loop is accessing memory in the same order as it is stored. The lowest level of the memory hierarchy is the extended memory, this could be a floppy disk, a tape, a CD or some other cheap "add-on" memory. For parallel computers we have in addition some kind of connection between the processors, so they can pass information to each other. From each processors' point of view communication with other processors can be thought of as accessing extended memory. A simple model for the time (T_c) it takes to pass a message is

$$T_c = T_{IC} + \frac{n_{Data}}{B}$$

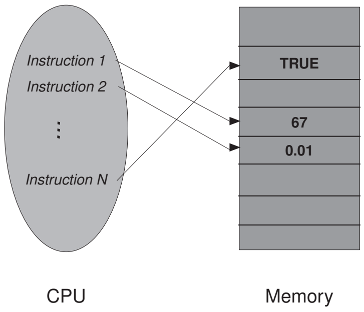

Figure 8: *Diagram of the von Neumann computer model.*

Here T_{IC} is the initialisation time for the communication, i.e. a constant part, n_{Data} the number of data (e.g. in bytes) and B the bandwidth (e.g. bytes per second) for the connection. The important properties of this model is that communication has a constant, here T_{IC}, and a cost that depends on the amount of data, here $\frac{n_{Data}}{B}$. Access time to memory can be modelled the same way. The initialisation time T_{IC} will decrease, and the bandwidth B will increase going upwards in the memory hierarchy. Modern processors uses *pipelining* and *overlapping* (see [13]) as an internal parallelisation. This implies it does not take n times as long time to do n (e.g.) float point operation as one. The second operation is started before the first one has finished, and so forth.

2.2 Parallel Computer Models

When we develop a sequential algorithm we do not think about which computer we are going to implement it on. In most cases we can even port one implementation from one computer to an other without any conceptual problems. This is because both the computers and their software are designed under the same model, the *von Neumann model*. Von Neumann modelled a single computer as a *central processing unit* (CPU) and a *storage unit* (memory). The CPU executes a predefined sequence of write and read operations to the memory (figure 8). This model has been so successful that we do not even think about it as a model when we develop algorithms. The von Neumann model works so well because it is simple and close enough to the true architecture. We do not gain much by designing machine specific algorithms. For parallel computers there has not been such a de facto model. There are many parallel computer models, and often a new computer architecture does not fit into old classification categories. This is about to change, models have started converging. The classification scheme suggested by Flynn in 1972 [14] has traditionally been widely used. It divides computers into four categories, two program instruction categories times two data access categories. The first class of computers is SISD -*Single Instruction Single Data*. This is not a parallel computer, but the traditional von Neumann model for sequential computers. A SIMD- or *Single Instruction Multiple Data* computer is a computer that applies the same instructions to different data. This

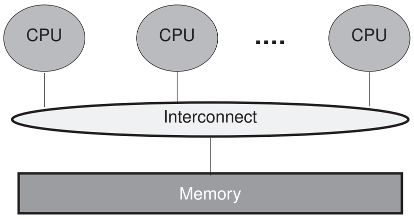

Figure 9: *Diagram of a MIMD-SM (Multiple Instruction Multiple Data - Shared Memory) machine.*

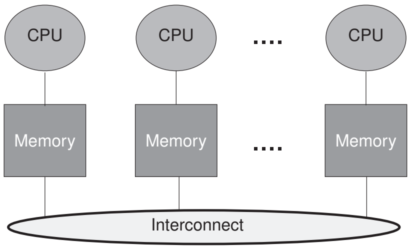

Figure 10: *Diagram of a MIMD-DM (Multiple Instruction Multiple Data - Distributed Memory) machine.*

kind of architecture was popular when programming had to be done closer to machine level and there was much human time saved when writing only one program. The third class, MISD -*Multiple Instruction Single Data* has, as far as we know, never been constructed. While the fourth class, MIMD or *Multiple Instruction Multiple Data* is the most common one. In these machines different processors can run different programs, execute instruction on their own data, and communicate with each other. There are two subclasses of MIMD, MIMD-SM (*Shared Memory*) and MIMD-DM (*Distributed Memory*).

The shared memory architecture involves multiple processors with a common memory pool, while in distributed memory architecture each processor has its own memory. See figure 9 and 10 for the principal differences. MIMD-DM machines are also sometimes called MPP -*Massively Parallel Processing*. SMP -*Symmetric Multiprocessor* is an other name (or subclass) of MIMD-SM, and refers to the equal (or symmetric) access to the shared memory. Modern high performance computers are often a combination of architectures. The nowadays popular *clusters* are often clusters of SMP-machines, connected using very high-speed interconnects, see figure 11.

The two dominating models today are MIMD-SM and MIMD-DM. The differences between them can be view as how the processors communicates. In the shared memory model processors communicate through a *shared address space*, while those in the distributed memory model communicate through *message passing*.

Which model you choose to program under depends on the computer you are using,

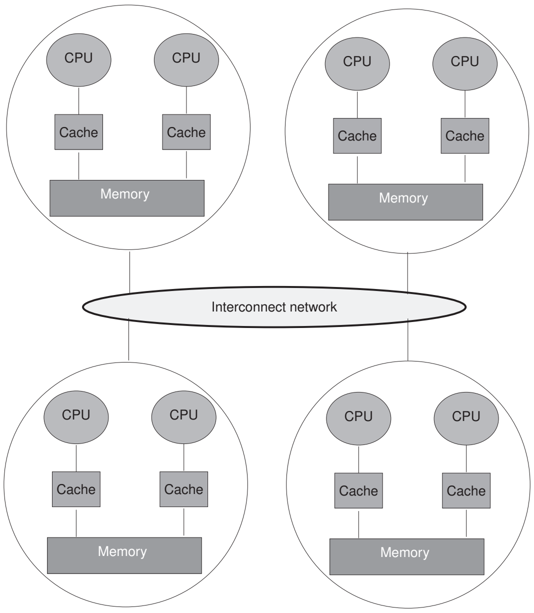

Figure 11: *Diagram of a cluster. A cluster consists of off-the shelve well performing personal computers connected with high-speed interconnects.*

but more importantly which compilers and libraries are available and also under which model it is easiest to program. Today there are compilers for both shared address and message passing models for most machines. We have chosen to use a message passing model because we are used to programming under this model. Also this makes it clear when communication is done, a benefit because communication is expensive and should only be done when necessary.

2.3 Designing parallel algorithms

> *"Parallel algorithm design (...) it requires the sort of integrative thought that is commonly referred as "creativity"."* [15]

When we start designing a parallel algorithm we often have a sequential version of it. In the easiest cases this algorithm suggests a good parallel version, but more often we have to change our approach to the problem.

Our main goal is, of course, one or both of speed-up and being able to handle larger problems. But there are also other properties we would like our parallel algorithm to have:

- *Scalability:* Even if we have a fixed sized problem and a maximum of processors available, our algorithm should be able to take advantage of more processors either for speeding up or for solving larger problems.
- *Unique solution:* There should exist a sequential algorithm that always gives the same result.

There are two main ways of parallelisation; *functional decomposition* and *domain decomposition*. With functional decomposition the processors are given different tasks to solve, but the input to their functions would be the same as in the sequential case. When the domain is decomposed each processor solves the same tasks, but on different parts of the problem, or domain. In a cooking analogy with two chefs (processors), seven courses (functions) and 50 guests (domain) the functional decomposition would be that one chef made three of the course for all the guests, the other four. While a domain decomposition approach would be that they both made all the courses for 25 guests. Most practical parallel solutions are a mixture of both, and it is often not even clear what is domain and what is function.

Newman and Barkema [34] give a third parallelisation class; *trivial parallelisation*. They define trivial parallelisation to be problems that can be splited up into tasks that are completely unrelated. No communication is needed, and what we do is essentially to run the same sequential program on different computers with different input parameters.

2.4 Performance analysis

The only reason we do parallel computing is to improve performance, either to get faster programs, or to be able to handle bigger problems. The most used performance measure is *speed-up*:

$$\text{Speed-up}(p) = \frac{time_1}{time_p}$$

Where $time_1$ is the run-time for one processor, preferable using the best sequential algorithm and $time_p$ is the run-time for p processors. The ideal, or rather optimal, speed-up is p, i.e. linear with the number of processors. In cases where the goal of the parallelisation is to solve larger problems, *scaled speed-up* can be a good measure.

$$\text{Scaled-speed-up}(p) = \frac{p \times time_1(A)}{time_p(p \times A)}$$

where $time_1(A)$ is the run-time on one processor for a problem of size A, and $time_p(p \times A)$ is the run-time for p processors for a problem of size $p \times A$.

The speed-up is a measure of the actual implementation. For the possible degree of parallelisation for an algorithm we have Amdahl's law:

$$\text{Speed-up} = \frac{1}{(1-f) + \frac{f}{p}}$$

where f is the fraction of the algorithm that is parallelisable. When the number of processors p approaches infinity, the speed-up approaches $\frac{1}{1-f}$. So the possible speed-up of an algorithm is limited by its sequential fraction $1 - f$. If 20% of an algorithm is sequential, then the maximum speed-up will be five. We can sometimes observe speed-up that is better, and even much better, then theoretical achievable. This kind of speed-up is known as *super linear speed-up*. If there is not enough room for the data which need to be stored during calculations in the cache a significant part of the computation time is spent shuffling data between the cache and the local memory. Using more processors can decrease the size of the computational problem enough so only the cache is used. In addition to the speed-up we theoretical get from the parallelisation, we will be given the time used for sending data between memory levels, thus observing a super linear speed-up.

Two important issues for performance is *load balance* and *communication overhead*. The load balance is good if processors have about the same amount of work to do and it is done concurrently. If processors are often idle this wastes CPU-time, and limits the performance of the parallelisation. Except in trivial case, parallelisation always involves some kind of communication between processors. Communication is expensive (time consuming), and in most cases is the major part of the extra cost that comes with parallelisation.

Often the sequential part of parallel algorithms requires some calculations that are based on input from all the processors, and all processors need the result in their further calculations. An obvious way of doing this is to let a master processor do these calculations, and communicate them to the other processors afterwards. But because of load balance and communication overhead it is often more beneficial to let all processors do these calculations. The other processors are idle when the master processor does the calculation, and there are communication to be done afterwards.

2.5 Some available software

There are two main branches of parallel application, those which involves parallelisation of linear algebra, and those which involves parallelisation of numerical solvers of differential equations.

Linear algebra is most relevant for the work we presented in this report. Much research has been done on parallel numerical linear algebra, for an introduction see [13].

Many numerical parallel linear algebra functions are collected in ScaLAPACK (Scalable Linear Algebra PACKage). This library is for distribute memory computers, and versions for most systems can be down loaded from `http://www.netlib.org/scalapack`. Most parallel computer vendors will have their own linear algebra library optimised for their architecture. IBM has their Parallel ESSL (engineering and scientific subroutine library), which contains a subset of ScaLAPACK. MPI (Message Passing Interface, [40]) has become the standard library for communication, and is available for both parallel computers and clusters of work stations.

3 Markov Chain Monte Carlo - A brief introduction

Analysing complex systems will in most cases result in complex models. For a stochastic model integration is almost always involved when making inference about its parameters. Only in a few special situations we are able to do this integration analytically. If we do not want to restrict our models to those few we can handle analytically we have to solve these integrals numerically.

For one dimensional reasonably well behaving functions there are well behaving and computational cheap numerical integration methods with known accuracy. There are two main groups of methods. The most popular way is by calculating $f(x)$ for some values of x, $(x_1, x_2, ..., x_n)$, and to use an integrable function $g(x)$ to interpolate between neighbouring points. The estimate is then

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} g_i(x)dx$$

The evaluation points are here fixed. An other option is to choose the evaluation points at random from a distribution proportional to $f(x)$ on the interval we want to integrate, and to use

$$\int f(x) \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

as our estimate. Where $x_i \sim cf(x)$, c is the normalisation constant and n is the number of samples. These methods are known as *Monte Carlo integration*. If we want to calculate $\int f(x)\pi(x)dx$ this can also be done by Monte Carlo integration:

$$\int f(x)\pi(x)dx \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

where x_i's are independent samples from $\pi(x)$. The law of large number ensures convergence for the Monte Carlo integration.

In statistical analysis we often have high dimensional spaces (> 1000 occurs) we want to integrate over. For a fixed evaluation point method applied to a p dimensional integral $\mathcal{O}(n^p)$ evaluation points are needed, where n is the number of points needed to get the same accuracy in one dimension. This approach is computational impossible for high dimensional integrals.

The error for the Monte Carlo method for a p dimensional problem with n sample is $\mathcal{O}_p(n)$, and hence independent of the dimension. But are we able to sample from $f(x)$?

The term *Monte Carlo sampling* was first introduced by Nicholas Metropolis in 1949. He and some fellow researchers working on the Manhattan project were the first to use computers for making random samples, and to make statistical inference from these. But *statistical sampling*, which was the old term, had been known for a century as a way to do numerical integration over poorly behaved one dimensional functions. For an interesting historical review of Monte Carlo methods, see Chapter 1 in [34].

3.1 Exact simulation methods

If we assume we can sample independent samples from an uniform distribution between 0 and 1, $u \sim U(0,1)$, then there are two principal ways of doing exact simulation, by transformation and by accept/reject.

3.1.1 Transformation

If the distribution function $F(x)$ for $x \sim f(x)$ is invertible it is possible to transform a sample u from $U(0,1)$ to a sample from $f(x)$ by

$$x_i = F^{-1}(u)$$

where F^{-1} is the inverse of F, i.e. $F(F^{-1}(u)) = u$. See [35] for a detailed introduction and examples. Commonly used distributions it is possible to sample from through transformations are exponential, Poisson and of course uniform distributions, both continuous and discrete and with other ranges then $(0,1)$. Combinations of exact sampled variables can in addition give us Gaussian and Chi-square distributed samples.

3.1.2 Accept / Reject

If $F(x)$ is not invertible it could be possible to use the accept/ reject algorithm to get exact samples. We have to find a density function $g(x)$ that we can sample from and a constant K such that

$$f(x) \leq Kg(x) \qquad \forall x$$

Then we can sample exact from $f(x)$ by algorithm 1.

Algorithm 1 *Accept / reject sampler*

- Given x^0
- While(accept==0)
  - Sample $y \sim g(y)$
  - Set $x = y$ with probability $\frac{f(x)}{Kg(x)}$
  - If accepted accept=1;
- Return x

A nice property of accept/reject sampling is that it is possible to sample from a distribution even if the density only is known up to a normalisation constant. The expected number of samples needed for each acceptance is K. In practice it is often hard to find a $g(x)$ with a small enough K to give a reasonable expected time to acceptance. For a more detailed introduction, extension and examples we again refer the reader to [35].

3.2 Markov Chain Monte Carlo

In most cases when we want to use of Monte Carlo integration we are not able to sample exact from our distribution $\pi(x)$. A solution is then to use *Markov Chain Monte Carlo*($MCMC$) simulation instead. The idea is to construct a *Markov Chain* (X^i) we are able to simulate from and that has $\pi(x)$ as its stationary distribution. Samples from this Markov chain x^i $i = 1 \dots n$, will, after a *burn-in* period b, be dependent samples from $\pi(x)$. We can use these samples in our Monte Carlo integration, and

$$\int f(x)\pi(x)dx \approx \frac{1}{n-b}\sum_{i=b+1}^{n} f(x^i)$$

as our estimate.

3.2.1 Markov Chains

We will now give a brief overview of Markov chain theory relevant for Markov chain Monte Carlo simulations. For more details, see chapter 3 and 4 in [19], chapter 4 in [35] and references therein.

A *Markov chain* is a discrete stochastic time process $\{X_0, X_1, \dots\}$ for which $X_i \in \chi$ conditioned on all $X_t = x_t, t < i$, only depends on X_{i-1}. I.e.

$$\pi_x(X_t|X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \dots, X_0 = x_0) = \pi_x(X_t|X_{t-1} = x_{t-1})$$

For a general state space Markov chain on χ, the chain is driven by a transition kernel $K(x, y)$ defined on $\chi \times \mathcal{B}(\chi)$ where $\mathcal{B}(\chi)$ is a Borel set on χ. As a function of any $x \in \chi$ $K(x, \cdot)$ is a probability measure (in the continuous case a density function). For all $A \in \mathcal{B}(\chi)$ $K(\cdot, A)$ is measurable. And we get

$$P(X^{t+1} \in A|X^t = x^t) = \int_A K(x_t, dy)$$

for any $A \in \mathcal{B}(\chi)$. We define the n-step transition kernel $K^n(x, y)$ by recursion;

$$K^n(x, A) = \int_A K^{n-1}(y, A)K(x, dy)$$

where $K^1(x, y) = K(x, y)$. A Markov chain is *reversible* if $\pi(X^t|X^{t-1} = x) = \pi(X^t|X^{t+1} = x)$. I.e. it can not be seen in which direction the chain runs. It will also be useful to define stopping-time at A, τ_A. We consider $A \in \mathcal{B}(\chi)$, and define

$$\tau_A = \inf\{n \geq 1; X^n \in A\}$$

i.e. the first n for which the chain visits A. The *number of visits* to A is

$$\eta_A = \sum_{n=1}^{\infty} \mathbb{I}_A(X^n)$$

In MCMC a Markov chain that converges to a specified stationary distribution is constructed. This chain has to fulfil to *irreducibility* and *recurrence*. But first we will define *stationary distribution*. A distribution π is stationary for a Markov chain with kernel $K(x, y)$, if

$$\pi(A) = \int_{\chi} K(y, A)\pi(dy)$$

for any $A \in \mathcal{B}(\chi)$.

Let ρ be a measure. A Markov chain (X^i) with transition kernel $K(x, y)$ is *ρ-irreducible* if for every $A \in \mathcal{B}(\chi)$ with $\rho(A) > 0$ there exists an n such that $K^n(x, A) > 0\ \forall x \in \chi$. The chain is *strongly irreducible* if $n = 1$ for all measurable A. Irreducibility ensures that the chain can reach any interesting part of χ within a finite number of steps. I.e. which parts of χ the chain can reach does not depend on the starting value, and does not change for different states x of the chain. An irreducible chain with a stationary distribution is said to be *positive*.

Recurrence ensures that the stationary distribution does not depend on the initial value x_0 of the Markov chain, and that the chain visits every set $\mathcal{B}(\chi)$ often enough. A ρ-irreducible Markov chain (X^i) is recurrent if, for every $A \in \mathcal{B}(\chi)$ with $\rho(A) > 0$, it has $\mathbb{E}_x(\eta_A) = \infty$ for all $x \in A$ (i.e. for ρ almost all x).

Theorem 4.1 in [19] states that if a Markov chain (X^i) is irreducible and has π as a stationary distribution it is π-irreducible, π is the unique stationary distribution and (X^i) is recurrent.

This theorem ensures us an unique stationary distribution for π-almost all chains. But the Markov chain is stared from an initial value, i.e. from a null set. *Harris recurrence* eliminates this problem.

An ρ-irreducible Markov chain (X^i) is Harris-recurrent if for every $A \in \mathcal{B}(\chi)$ with $\rho(A) > 0$ we have

$$P(\eta_A = \infty) = 1$$

for all $x \in \chi$.

The difference between recurrence and Harris-recurrence is the difference between π-almost all x and all x.

A positive Harris-recurrent Markov chain is also known as an ergodic Markov chain. For an ergodic Markov chain there are some convergence results. We use the *total variation norm*;

$$\| \mu_1 - \mu_2 \|_{TV} = \sup_A |\mu_1(A) - \mu_2(A)|$$

where μ_1 and μ_2 are distribution functions. For an ergodic Markov chain with transition kernel K and stationary distribution π then

$$\lim_{n \to \infty} \| K^n(x, \cdot) - \pi \|_{TV} = 0$$

More importantly for our purposes any ergodic Markov chain (X^i) with initial distribution μ the following holds,

$$\lim_{n \to \infty} |\mathbb{E}_\mu[h(X^n)] - \mathbb{E}_\pi[h(X)]| = 0$$

for every bounded h.

3.2.2 Metropolis-Hastings

A method suggested by Metropolis, Rosenbluth, Rosenbluth, Teller and Teller in 1953 [30] and extended by Hasting in 1970 [23] gives a way of constructing a Markov chain that has $\pi(x)$ as its stationary distribution. A Markov chain (X_n) with transition kernel $K(x,y)$ is *detail balanced* if there exists a π such that

$$K(y,x)\pi(y) = K(x,y)\pi(x)$$

for $\forall x, y$. If $\pi(x)$ satisfies the detailed balance condition, then $\pi(x)$ is (X_n)s stationary distribution, and the chain is reversible. Detail balance is a sufficient but not necessary condition for $\pi(x)$ to be the stationary distribution. The Metropolis-Hasting algorithm, algorithm 2, makes such a Markov chain. q is called the *proposal* (or *instrumental*) distribution. We will prove that the Metropolis-Hasting algorithm for every q who's support includes χ, produces a Markov chain with stationary distribution π. The transition kernel is

Algorithm 2 *Metropolis-Hastings Algorithm*

- Given x^0
- for $i = 0 : (n-1)$
  - Sample $y \sim q(y|x^i)$
  - $\alpha(x,y) = \min(\frac{\pi(y)q(x^i|y)}{\pi(x^i)q(y|x^i)}, 1)$
  - $x^{i+1} = \begin{cases} y, & \text{with probability } \alpha \\ x^i, & \text{with probability } 1-\alpha \end{cases}$
- Return x

$$K(x,y) = \alpha(x,y)q(y|x) + (1 - r(x))\delta_x(y)$$

where $r(x) = \int \alpha(x,y)q(y|x)dy$ is the probability of rejecting y. δ_x is the Dirac mass in x. We can now establish detailed balance by verifying that

$$\alpha(x,y)q(y|x)\pi(x) = \alpha(y,x)q(x|y)\pi(y)$$

and

$$(1 - r(x))\delta_x(y)\pi(x) = (1 - r(y))\delta_y(x)\pi(y)$$

which implies that $\pi(x)$ is the stationary distribution of the Markov chain (X^n) generated by the Metropolis-Hasting algorithm.

A convergence property associated with Metropolis-Hasting chains is that if π is bounded and positive on every compact set of its support χ, and there exists positive numbers ϵ and δ such that $q(y|x) > \epsilon$ if $|x - y| < \delta$ then the Metropolis-Hasting chain is π-irreducible, aperiodic and Harris recurrent. Then:

1. If $h \in L^1(\pi)$,

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} h(X^n) = \int h(x)\pi(x)dx$$

2. If in addition (X^n) is aperiodic, then

$$\lim_{N\to\infty} \| \int K^n(x, \cdot)\mu(dx)\|_{TV} = 0$$

for every initial distribution μ.

This result enable us to use samples from the Metropolis-Hasting algorithm to estimate expectations under π. We are going to describe two subclasses of Metropolis-Hasting, *random walk* and *independence sampler*.

For the independence sampler, see algorithm 3, the proposal $q(y|x)$ does not depend on the current value of the chain x; $q(y|x) = q(y)$. The proposal is the same distribution every iteration.

Algorithm 3 *Independent Metropolis-Hasting*

- Given x^0
- for $i = 0 : (n-1)$
  - Sample $y \sim q(y)$
  - $\alpha(x, y) = \min(\frac{\pi(y)q(x^i)}{\pi(x^i)q(y)}, 1)$
  - $x^{i+1} = \begin{cases} y, & \text{with probability } \alpha \\ x^i, & \text{with probability } 1-\alpha \end{cases}$
- Return x

Note that even if the proposed y is independent of the x^i, the resulting sample x^{i+1} is not independent of x^i. The probability of accepting y as x^{i+1} depends on x^i. Another natural way of constructing a proposal y, is to let y be a perturbation of x

$$y = x^i + \epsilon$$

where $\epsilon \sim g()$, and $g()$ does not depend on x_i. This makes the proposal distribution depend only on the change; $q(y|x) = q(x - y)$. Metropolis-Hastings algorithms with this kind of proposal are known as random walk Metropolis-Hastings. The algorithm proposed by Metropolis et. al. in [30], see algorithm 4, is the special case of random walk Metropolis-Hastings, where $g()$ is symmetric. This gives $q(y|x) = q(x|y)$.

Algorithm 4 *Metropolis Algorithm*

- Given x^0
- for $i = 0 : (n-1)$
  - Sample $y \sim q(x-y)$
  - $\alpha(x,y) = \min(\frac{\pi(y)}{\pi(x)}, 1)$
  - $x_{i+1} = \begin{cases} y, & \text{with probability } \alpha \\ x_i, & \text{with probability } 1-\alpha \end{cases}$
- Return x

3.2.3 Gibbs sampler

Under the positivity condition (defined below) the joint distribution π satisfies

$$\pi(x_1, \ldots, x_p) \propto \prod_{j=1}^{p} \frac{\pi^{(l_j)}(x_j|x_1, \ldots, x_{j-1}, x'_{j+1}, \ldots, x'_p)}{\pi^{(l_j)}(x'_j|x'_1, \ldots, x'_{j-1}, x_{j+1}, \ldots, x_p)}$$

for every permutation l on $\{1, 2, \ldots, p\}$ and every $x \in \chi$.

Let $\pi^{(i)}$ denote the marginal distribution of x_i. π satisfies the *positivity condition* if $\pi^{(i)}(x_i) > 0$ for every $i = 1, \ldots, p$ implies that $\pi(x_1, \ldots, x_p) > 0$. The result implies that a multi-dimensional distribution $\pi(x) = \pi(x_1, x_2, \ldots, x_p)$ is uniquely specified from all its full conditional distributions $\pi(x_i|x_{-i})$, $i = 1, 2, \ldots, p$ and $x_{-i} = (x_i, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p)$. The *Gibbs sampler*, see algorithm 5, is a Markov Chain Monte Carlo sampler that only uses the conditional distributions. The method came from statistical physics, and was introduced to the statistical society (and given the name 'Gibbs sampler') by Geman and Geman in 1984 [18].

Algorithm 5 *Gibbs sampler (fixed scan)*

- Given x^0
- for $i = 0 : (n-1)$
  - for $j = 1 : p$
    - Sample $y_j \sim \pi(x_j|y_0, \ldots, y_{j-1}, x^i_{j+1}, \ldots, x^i_p)$
  - $x^{i+1}_j = y$
- Return x

Each x_i here can be either one or multi-dimensional. If some of the x_i's are multi-dimensional we have a *block Gibbs sampler*. The Gibbs sampler can be seen as a combination of p Metropolis-Hasting steps, each with acceptance probability $\alpha = 1$. But it most be stressed that the Gibbs sampler above is not a reversible Metropolis-Hasting algorithm: The proposal distribution q changes for each dimension, and the resulting chain is not reversible.

If the full dimensional transition kernel is absolutely continuous with respect to π, and its Markov chain (X^n) is aperiodic, then

$$\lim_{n\to\infty} \parallel K^n(x,\cdot)\mu(dx) - \pi \parallel_{TV} = 0$$

for every initial distribution μ.

The Gibbs sampler given above is a fixed scan Gibbs sampler, because the elements of Y is updated in the same order in every iteration. For proofs and a presentation of *random scan* Gibbs samplers see chapter 7 in [35]

If it is not possible to sample from the full conditional distribution, there is an option to *Metropolise* the Gibbs sampler, see algorithm 6. In each step y_i is sampled from a proposal distribution $q_i(y_j|y_1,\dots,y_{j-1},x^i_{j+1},\dots,x^i_p)$, and is accepted or rejected. This algorithm is also known as *single site Metropolis-Hastings*.

Algorithm 6 *Metropolised Gibbs sampler (fixed scan)*

- Given x^0
- for $i = 0 : (n-1)$
  - for $j = 1 : p$
    - Sample $y_j \sim q_j(x_j|y_0,\dots,y_{j-1},x^i_{j+1},\dots,x^i_p)$
    - $\alpha(x,y) = \min(\frac{\pi_j(y_j|y_0,\dots,y_{j-1},x^i_{j+1},\dots,x^i_p)q_j(x_j|y_0,\dots,y_{j-1},x^i_{j+1},\dots,x^i_p)}{\pi(x^i_j|y_0,\dots,y_{j-1},x^i_{j+1},\dots,x^i_p)q_j(y_j|y_0,\dots,y_{j-1},x^i_{j+1},\dots,x^i_p)},1)$
    - $x^{i+1}_j = \begin{cases} y_j, & \text{with probability } \alpha \\ x^i_j, & \text{with probability } 1-\alpha \end{cases}$
- Return x

3.2.4 Convergence Diagnostics

If we have an ergodic Markov chain with π as an invariate measure we are assured convergence, but do not know how fast. The questions we have to ask is 1) has the chain converged, and 2) how long chain do we need, i.e. how good is it mixing. To determine the burn-in period, and the necessary length of the Markov chain for our estimate to converge is a major and difficult task in MCMC. There are theoretical measures for how fast the chain converges (for example for discrete and finite chains, how close the second largest eigenvalue of the transition matrix is to 1). But in most cases it is not possible to find these kind of qualitative parameters easily, and for all practical proposes we have to base our diagnostics on the samples the chain gives us.

For a start, trace plots for some of the variables are made and visually inspected. The goal of almost all MCMC simulations is to find the expected value for some function f. To plot the average $\bar{f}_j = \frac{1}{j}\sum_{i=1}^{j} f(x_i)$ for $j = 1, 2, \ldots, n$, and see if it has stabilised is an obvious next step. Simulations can only tell us about areas they have visited (and for Metropolis-Hastings the areas the chain tried to visit). Especially this is a problem when the target distribution is multi-modal. We can start in a local mode and 'never' (in our simulation time) get to the main mode. That move is very unlikely to be suggested. Something that may help us discover both multi- modality and slow mixing is running several chains with different initial values. If the chains give different estimates and/or do not visit the same areas they have not converged or not mixed enough. Estimating the auto-correlation function for succeeding samples for some of the variables and relevant functions can help us quantify the mixing. If the auto-correlation is large (e.g. for lag one) the chain is moving slowly and we have slow mixing.

Metropolis-Hastings algorithms often have some parameters in the proposal distribution that the user has to set. The tuning of these parameters is often a trade-off between good mixing (suggesting large steps) and high acceptance probability (small steps gives higher probability of acceptance). Gelman, Roberts and Gilks [17] suggest an acceptance rate of 50% for dimension 1 - 2, and 25% for high-dimensional models.

There are several review articles written about MCMC diagnostics, e.g. [29], [8] and [3].

3.2.5 Proposals and updating schemes

Even though it has been known for some years that updating variables in blocks can improve the mixing ([4] [28]) single site updating schemes have been the most used ones. Results in [26] suggest that all variables should be updated in one block for latent variable models, due to the fact that the hyper-parameters and latent variables are strongly correlated.

Construction of useful high dimensional proposal distributions, and sampling from these has been a problem. The fast sampling algorithm [36] by Rue and the method of construction proposals in [37] are recent contributions that makes it more feasible to use one-block updating scheme Metropolis-Hastings samplers.

4 Parallel MCMC -A literature review

4.1 Introduction

Markov chain Monte Carlo methods are powerful in their generality, but computationally very expensive. Using more than one processor is therefore a tempting and obvious solution to be able to get faster programs and make inference in larger and more complex models. Geman and Geman mentioned parallel processing as an opportunity in [18] and a few different possible parallelisations in statistical inference are described by Schervish in [39].

MCMC is an iterative method, and hence in nature sequential. Parallelisation most therefor be done on domain level, within each iteration. An exception is multiple chains, with or without interactions. The most used parallelisation is probably to let independent chains run on different processors, without any interaction. For the pragmatic user this is the most powerful way of parallelisation -the best you can get when the number of chains equals the number of processors.

Parallelisation is often mentioned in MCMC literature, but very seldom done. We will give an overview of the few implemented parallel MCMC methods we have found, and their results. A detailed description and new parallel implementations of both parallel chains and parallelisations of single chains for models with Markov properties are done in [44].

4.2 From Statistical Physics

MCMC was first developed by statistical physicists and many of the more specialised methods were also first suggested by them. Chapter 14 in [34] and [1] present the same methods for parallel simulation of the Ising model. The Ising model is a Markov random field with neighbourhood of order one.

Their first parallelisation method is a single site random scan Gibbs sampler. The domain is divided into p blocks (p is the number of processors), and each processor updates its own block. The challenge here is the borders between the blocks. They have three different solutions. The first one is to communicate with the corresponding neighbour each time a board element is updated. This causes a lot of communication overheads and slows down the algorithm. This is not only due to the communication, but also due to the synchronisation that is needed. A way to avoid a lot of this communication is to let each processor have an extra buffer with its neighbours borders. This combined with using the same random scan for all processors decreases the need of communication. Only when a border element has been updated it is necessary to send it from one processor to its neighbour. Their third method is to make a buffer between the blocks, that are not updated and to change the borders/blocks to achieve ergodicity. For the simple method they report a maximum speed-up of 27 for 40 processors. This is for a 200×200 lattice. The method with buffer borders and the same scan for all processors outperforms the first method by a factor of 1.6.

Single site updating converges very slow for the Ising model near the critical temperature, which are the interesting cases for the statistical physicist. The Swendsen-Wang algorithm [41] deals with this by updating clusters of elements at the same time. Barkema and MacFarland [1] have made a parallel version of Swendsen-Wang. They again divide the domain, one part to each processor. Each processor makes clusters on its own, but all the clusters that contains a border element is assigned to a *master processor*. The method gains its highest speed-up of 3.2 for 9 processors.

4.3 Spatial Gaussian Models

Whiley and Wilson [43] have developed parallel simulation methods for latent spatial Gaussian models (as introduced in chapter 1). They have three approaches, two exact and one approximation. Their first method is to parallelise the matrix calculations only. A Gaussian process model gives a full covariance matrix Σ that depends on hyper-parameters θ, $\Sigma(\theta)$. Sampling from the latent field x and evaluating the distribution involves calculating the inverse, $\Sigma(\theta)^{-1}$, the determinant, $|\Sigma(\theta)|$ and $y^T\Sigma(\theta)^{-1}y$. For large fields these calculations are the most time consuming part of the algorithm. Whiley and Wilson's first method is to parallelise only these parts of the algorithm. In practice calculating the Choleskey factor L, $\Sigma(\theta) = LL^T$ is the computational most expensive part. They have used well-known methods and software from numerical linear algebra. Their results give some speed-up, about 2 for 4 processors, and 3.5 for 16 processors.

Their second method is a domain decomposition way of parallelisation. The field is divided into $M \cdot K$ blocks, see figure 12. The idea is that blocks that are far away from

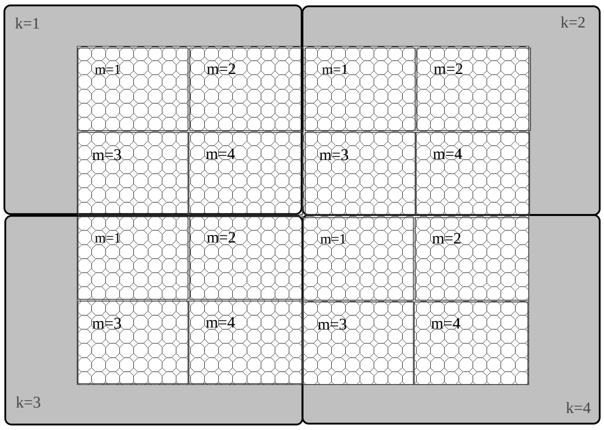

Figure 12: *Illustration of the geostatistical setting.*

each other are approximately independent and can be updated concurrently. The blocks are allocated to M groups in such a way that blocks in the same groups are not neighbours. Further are blocks in the same group approximately independent given the elements in the other groups. Blocks in the same group are updated in parallel without any interaction within an iteration. This method makes a very scalable parallel algorithm. Also the speed-up for a fixed size problem is very good (10 for 16 processors). But the algorithm is not exact and the samples are treated as if they were.

The third method is an exact version of the second. The full conditional distribution is now used for each block. This means that blocks can not be updated simultaneous. But the evaluation of the conditional distribution of a block can be done simultaneously as sampling from an other block. This kind of "sequential" parallelisation gives poor theoretical scalability and speed-up properties. Though for a realistic number of processors and problem size it can work well, and their results confirm that.

4.4 Parallel Simulated Annealing for Markov Fields

In [5] Jeng, Woods and Rastogi have presented a method for parallel simulation on Markov Fields in a simulated annealing setting. The algorithm was suggested by Murray, Kashko and Buxton in [32]. They have written it in a SIMD machine setting, but it can easily be used under other parallel programming models. The parallelisation is done within each temperature iteration, so from a parallelisation point of view they are parallelising single site updating of a Markov random field. They do this by domain decomposition, and along the same lines as method two of Whiley and Wilson. They divide the Markov field into disjunct subsets, and are updating conditional independent blocks at the same time. Since Jeng, Woods and Rastogi, unlike Whiley and Wilson, work with a Markov field they have conditional independent blocks, and the method is exact.

5 Parallel Exact Sampling of Gaussian Markov Random Fields

Multivariate Gaussian distribution is the multidimensional version of Gaussian distribution. In recent years the importance of block-update in MCMC has been discovered, see e.g. [26]. The challenge is often to find a multi-dimensional proposal distribution that gives "any" accept rate, and that is reasonable cheap to sample from. Multivariate Gaussian would be an obvious choice, but to sample from it costs $\mathcal{O}(n^3)$, where n is the dimension of the variable to be sampled from. This section describes a way of fast parallel exact simulation from a special version of multivariate Gaussian distribution, Gaussian Markov Random Fields (GMRFs). The cost of exact sampling from a spatial GMRF is $\mathcal{O}(n^{3/2})$.

5.1 Gaussian Markov Random Field

A multivariate Gaussian distribution is also referred to as a Gaussian random field (GRF), and $x = (x_1, x_2, \ldots, x_n)^T$ is a GRF if

$$x \sim N_n(\mu, \Sigma)$$

where μ is the expected value and Σ is the covariance matrix. Spatial data have positions attach to them, and the elements s_{ij} of Σ are often a function of the positions of the variables x_i and x_j. These kind of functions are known as correlation functions. For an introduction to spatial models and GRFs see e.g. [9]. As mentioned in section 1.3 the distribution of fixed locations of a Gaussian process forms a GRF.

Gaussian Markov random fields are Gaussian random fields with a Markov property. The Markov property for multi-dimensional variables is defined through conditional dependence and a neighbourhood structure, see [6] for a detailed introduction to Markov fields. Assume our variable $x = (x_1, x_2, \ldots x_n)$ has a neighbourhood structure, denote $i \sim j$ if x_i and x_j are neighbours, and let $\mathcal{N}_i$ denote the neighbourhood of element i, i.e. $\mathcal{N}_i = \{\forall j : i \sim j\}$. A GRF is a GMRF if

$$\pi(x_i | x_{\mathcal{N}_i}, x_j) = \pi(x_i | x_{\mathcal{N}_i}) \qquad \forall j \notin \mathcal{N}_i$$

If x_i and x_j are conditional dependent given all the other variables in a GMRF they are neighbours. Image restoration and disease mapping are examples of where GMRF have been extensively used, see e.g. [2], [20], [31] and [45] .

The choice of modelling with GRF or GMRF is a choice of working with dependence or conditional dependence. For a GRF we are modelling the covariance matrix Σ. For a GMRF we model the inverse covariance matrix, known as the precision matrix. If x is a GMRF, we often write

$$x \sim N_n(\mu, Q^{-1})$$

where Q is the precision matrix.

A characteristic feature of the precision matrix is that element (i, j) in Q, q_{ij} is non-zero if and only if x_i and x_j are neighbours or $i = j$. Therefore the neighbourhood structure

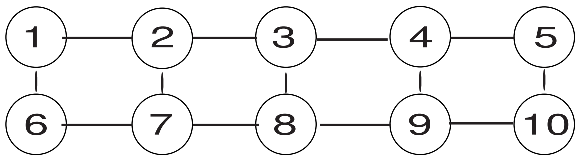

Figure 13: *Graph of a GMRF.*

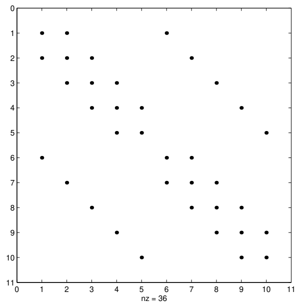

Figure 14: *Non-zero elements in the precision matrix corresponding to the graph in figure 13*

is reflected in Q, and with relatively small neighbourhoods Q is sparse, a matrix mainly containing zeros. Further the precision matrix is, as the covariance matrix, symmetric and positive definite (for improper distributions semi-definite). Remark that a sparse precision matrix generally gives a full covariance matrix. The sparse structure makes the computations cheaper, so there are pragmatic reasons for choosing GMRFs. Interpretation of the non-zero elements in a precision matrix is non-intuitive, and correlation functions are often preferred. In [38] Rue and Tjelmeland describe how GRFs can be approximated with GMRFs. It is possible to get good approximations using surprisingly small neighbourhoods.

GMRFs can easily be visualised with graphs. Each element of the variable is a node, and there are edges between neighbours. See figure 13 and 14 for a graph and its corresponding precision matrix.

5.2 Sampling from GMRFs

An n-dimensional GRF $x \sim N_n(0, Q^{-1})$ can be sampled from n independent identical distributed standard Gaussian variables $z \sim N_n(0, I)$. We know that

$$\pi(x) \propto \exp(-\frac{1}{2}x^T Q x)$$

And that the precision matrix Q has a Choleskey decomposition;

$$Q = LL^T$$

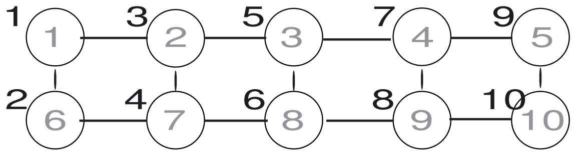

Figure 15: *A reordering of the graph in figure 13 used for faster sequential sampling.*

where the Choleskey factor L is a lower triangular matrix. Hence

$$\pi(x) \propto \exp(-\frac{1}{2}x^T LL^T x) = \exp(-\frac{1}{2}(L^T x)^T I(L^T x))$$

where I is the identity matrix. The standard multivariate Gaussian variable's density function is proportional to

$$\pi(z) \propto \exp(-\frac{1}{2}z^T Iz)$$

hence $L^T x$ is standard Gaussian, and we can get a sample from x, by solving

$$L^T x = z$$

for x.

For a full Q matrix the cost of both the Choleskey decomposition and of solving the triangular system is $\mathcal{O}(n^3)$.

5.2.1 Faster sequential sampling of GMRFs

Rue [36] suggests a potentially computational cheaper method for exact sampling of GMRFs. We know the Markov property gives us a sparse precision matrix Q. A sparse Q can give us a sparse Choleskey factor L, but this depends on the *bandwidth* b of Q. The bandwidth is the maximum of the distances between the diagonal elements and their rows' furthest out non-zero element. The bandwidth of the matrix in figure 14 is $b = 5$. The computational complexity of the Choleskey decomposition is $\mathcal{O}(nb^2)$. The basic idea of Rue's algorithm for sampling GMRF is that the ordering (or indexing) is dummy, and a *reordering* can give us a smaller bandwidth. Figure 15 and 16 show a reordering of the graph in figure 13 and its precision matrix. We observe that the bandwidth is now only $b = 2$, and the computational costs has decreased with a factor 2.5. It is never guaranteed that the reordering will give a speed-up, but the bandwidth is $\mathcal{O}(\sqrt{n})$. Hence is the cost of the sampling now $\mathcal{O}(n^2)$.

An other measure of the cost is *fill-in*. Fill-in is the number of non-zero elements in L that are zero in Q. Figure 17 and 18 show the fill-ins of the Choleskey factors of the original and reordered graph. The reordering has reduces the fill-in from 16 to four. A method that is based on this measure is the multi-frontal algorithm for sparse matrix factorisation (described in [22]). The cost is $\mathcal{O}(n^{3/2})$ for a 2D-graph and $\mathcal{O}(n^2)$ for a 3D-graph, and is hence cheaper then the bandwidth method.

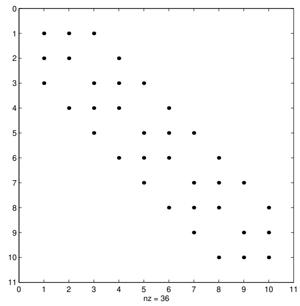

Figure 16: *Non-zero elements of the precision matrix corresponding to the graph in figure 15.*

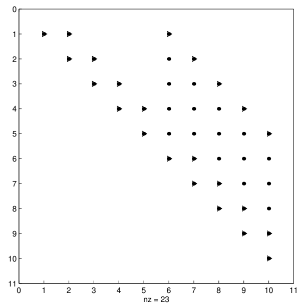

Figure 17: *Non-zero elements of the Choleskey factor L^T corresponding to the graph in figure 13. The non-zero elements of Q are marked with triangulars.*

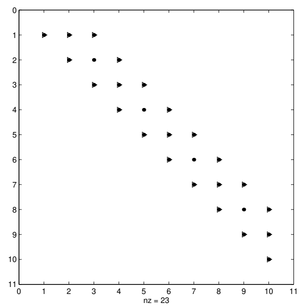

Figure 18: *Non-zero elements of the Choleskey factor L^T corresponding to the graph in figure 15. The non-zero elements of Q are marked with triangulars.*

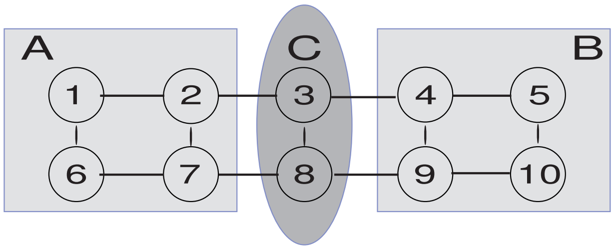

Figure 19: *The graph in figure 13, with a segmentation set.*

The non-zero structure of the Choleskey factor L^T can be given an interpretation in probability terms. An element q_{ij} of the precision matrix is zero if x_i and x_j are independent given all the other elements of x. An element l_{ji} $(j > i)$ of the corresponding Choleskey factor L is zero if x_i and x_j are independent given all x_k where $k > i$ and $k \neq j$, i.e. all x_k with a higher index then i.

The optimal ordering problem is well known in the computer science community, and is known to be a NP-complete. Heuristic algorithms are used for the reordering.

5.2.2 Parallel exact sampling of Gaussian Markov Random Fields

The main ingredients in the exact sampling method described above are also the main ingredients in *solving sparse systems of linear equations with symmetric coefficient matrix*. These equation systems occur in many engineering and scientific computing application, and have been researched for years. When solving a system $Ax = b$ for a symmetric positive definite A, the Choleskey factorisation is first found, $A = LL^T$, followed by solving two triangular systems, first $Ly = b$ for y, then $L^T x = y$ for x. The first and last step are the two main step in our sampling method. The numerical methods used in the exact sampling method described below are from numerical analysis, and is described in Gupta et. al. [22] and references therein.

The basic idea of the parallel exact GMRF sampler is to find segmentation set(s). If a segmentation set x_C that makes a buffer between x_A and x_B, then $x_A|x_C$ is independent of $x_B|x_C$. See figure 19. Hence if we have a sample x_C from π_{x_C}, we can simulate x_A from $\pi_{x_A|x_C}$ and x_B from $\pi_{x_B|x_C}$ simultaneously. The sample $x = (x_A, x_B, x_C)$ is then a sample from the target distribution π_x.

We start with parallelisation of the last part of the sampling algorithm, solving the triangular system. See figure 20 for a visualisation. This system is solved calculating from

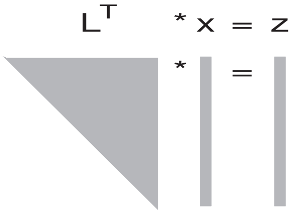

Figure 20: *A visualisation of the triangular system.*

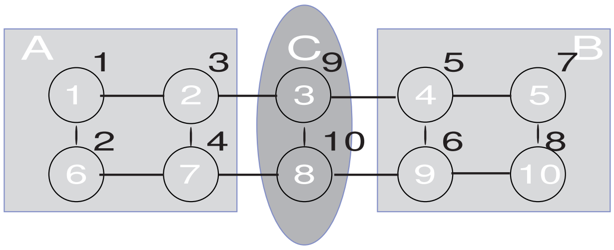

Figure 21: *Reordering of the graph in figure 13 for the parallel case.*

the bottom upwards:

$$\begin{aligned} l_{nn}x_n = z_n \quad &\Rightarrow \quad x_n = \frac{z_n}{l_{nn}} \\ &\vdots \\ \sum_{j=i}^{n} l_{ij}x_j = z_i \quad &\Rightarrow \quad x_i = \frac{z_i - \sum_{j=i+1}^{n} l_{ij}x_j}{l_{ii}} \\ &\vdots \\ \sum_{j=1}^{n} l_{1j}x_j = z_1 \quad &\Rightarrow \quad x_1 = \frac{z_1 - \sum_{j=2}^{n} l_{1j}x_j}{l_{11}} \end{aligned}$$

where x_n is a sample from π_{x_n}, x_i is a sample from $\pi_{x_i|x_{\{k>i\}}}$ and x_1 is a sample from $\pi_{x_1|x_{\{k>1\}}}$.

That means sampling from x_C first, is equivalent to placing the elements of x_C at the end of x. From the stochastical interpretation of L^T we see that $l_{ij} = 0$ for $i \in A$ and $j \in B$ if $x = (x_A, x_B, x_C)$, and x_C is a segmentation set. See figure 21 for the reordered graph, and figure 22 for its Choleskey factor. After the system $L^T x = z$ is solved for x_C the zero-block structure of L^T allows us to solve it for x_A and x_B independently, and therefore in parallel.

The first step of the algorithm, the Choleskey decomposition, is often by far the computationally most expensive. It is therefore important to have a good parallelisation of it.

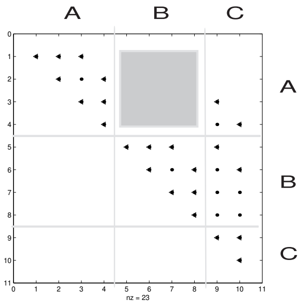

Figure 22: *The Choleskey factor corresponding to the graph in figure 21.* The gray area indicates the zero-block important to parallelisation

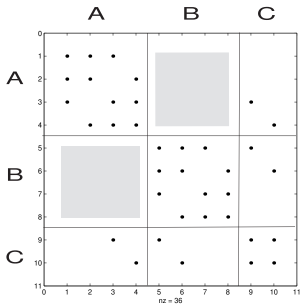

Figure 23: *The precision matrix corresponding to the graph in figure 21.* The gray areas indicate the zero-block important to parallelisation

The precision matrix of the reordered graph (as in figure 21) is in figure 23. We get the same kind of zero-block structure between x_A and x_B in Q as in L^T. Choleskey decomposition is done by column wise right looking elimination. Elimination can be done for the columns in A and B independently / in parallel. For the elimination of the C columns we need columns from both A and B, so this has to be done sequentially and as the last step.

We observe that the fill-in in the Choleskey factor for the parallel reordering is eight. This is four more then in the sequential optimal reordering. We have to pay for the parallelisation with more fill-ins.

For extension to more than two processors and technical details, see [22] and references therein. We only mention that the communication overhead is $\mathcal{O}(n\sqrt{p})$ and the scalability is $\mathcal{O}(p^{1.5})$ for a problem with n nodes and $\mathcal{O}(\sqrt{n})$-node separators, p is the number of processors.

Gupta et al.[22] divide the algorithm into four main steps:

1. Reorder - find segmentation set(s).
2. Symbolic factorisation - find the non-zero structure of L.
3. Numerical factorisation - calculate the values of the non-zero elements
4. Solve triangular system(s)

The two first steps, reordering and symbolic factorisation, are unique for each graph. In MCMC application we often have one graph, and want samples from different precision matrices $Q(\theta)$ for different values θ. Then only the numerical factorisation and solving the triangular system need to be done in each iteration.

5.3 Implementation and performance

The implementation of the parallel exact GMRF sampler has been done using WSMP - Watson Sparse Matrix Package [21]. This package uses a modified version of the multifrontal algorithm. The program has been run on Queens University Belfast's SP/2 system. A 48 node system with 160MHz Power2CS processors, each with between 256Mb and 1Gb main memory. We did not require any special memory for the node due to very long waiting time else. This can cause inconsistent results.

5.3.1 Speed-up and scaled speed-up for a fixed size lattice

The first performance test is for a 400×400 lattice (160000 nodes), where each non-border element has four neighbours (see figure 26). Table 1 and figure 24 gives the test results.

The speed-up is calculated using time spend on Choleskey decomposition and solving the triangular system only. This is presented because these are the only steps done for each iteration in MCMC. WSMP is made such that the parallelisation is best for 2^k processors (where k is a natural number). This can been seen in figure 24. Two processors give a speed-up of 2.3, four of 4.1, eight of 6.4 and 32 of 12.9. For two and four processors

# proc.	order	sym.fac.	num.fac.	tri.sys.
1	16.7	0.94	5.92	0.28
2	9.67	1.20	2.58	0.16
3	9.71	1.39	2.57	0.16
4	7.12	1.14	1.42	0.09
6	7.06	1.30	1.14	0.42
8	5.96	1.11	0.92	0.05
12	6.02	1.21	0.94	0.05
16	5.66	1.21	0.64	0.04
32	5.79	1.21	0.45	0.03

Table 1: *Performance results for one GMRF sample of a* 400×400 *lattice.*

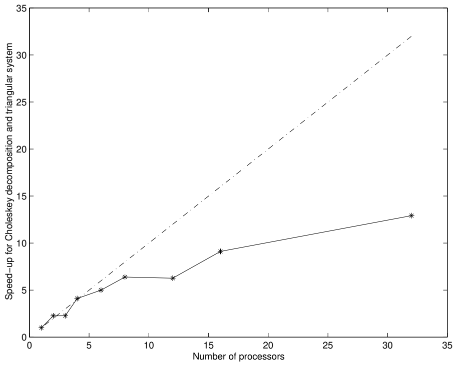

Figure 24: *Speed-up (*-) and optimal speed-up (–) for the time used for Choleskey decomposition and solving the triangular system in table 1*

this is "better than perfect", and should not be possible under idealised circumstances. We assume we get this result either because of different memory resources on different nodes, or it is a normal super linear speed-up case. We get very good speed-up until eight processors. Also a speed-up of 12.9 for 32 processors is quite good.

Further a scalable speed-up test is done. Four-neighbour GMRFs was sampled for approximately $400\sqrt{p} \times 400\sqrt{p}$ lattices using p processors. The results are given in table 2 and figure 25.

The cost of the Choleskey decomposition is $\mathcal{O}(n^{3/2})$ where n is the number of elements in the lattice. From this point view the scaled speed-up is good, 1.7 for two processors, and 2.1 for four. For eight processors (and eight times as big lattice) the problem not is solved due to lack of memory. An interesting feature with this test is that the 565×565 case (two processors) is too big to be solved on one processor.

# proc.	size	order	sym.fac.	num.fac.	tri.sys.
1	400×400	16.7	0.94	5.92	0.28
2	565×565	19.6	2.36	6.80	0.33
3	693×693	31.2	4.04	11.7	0.44
4	800×800	31.8	4.41	11.0	0.35
6	980×980	49.8	7.54	65.73	12.0

Table 2: *Performance results for one GMRF sample of a $400\sqrt{p} \times 400\sqrt{p}$ lattice.*

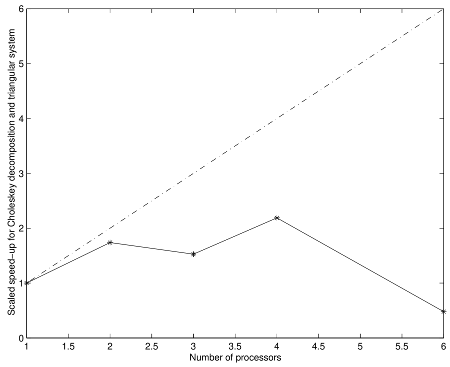

Figure 25: *Scaled speed-up (*-) and optimal scaled speed-up (–) for the time used for Choleskey decomposition and solving the triangular system in table 2*

5.3.2 Neighbourhoods

So far we have only used a neighbourhood where each element has at most four neighbours. We often want to use bigger neighbourhoods, and we want to investigate how that influences the sampling. Tests are done for three different neighbourhoods, with four, nine and 24 neighbours, see figure 26, on a 400×400 lattice. A bigger neighbourhood gives a less

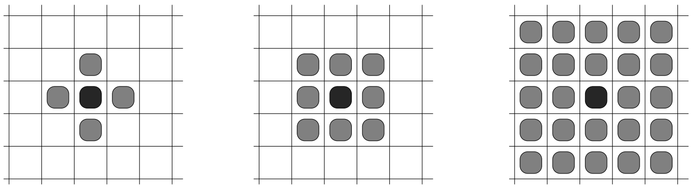

Figure 26: *The three neighbourhoods used in the tests.*

sparse system and more fill-in. Therefore the trend of bigger neighbourhoods implies more expensive simulations which can be observe in figure 27. Especially, the 24 neighbours

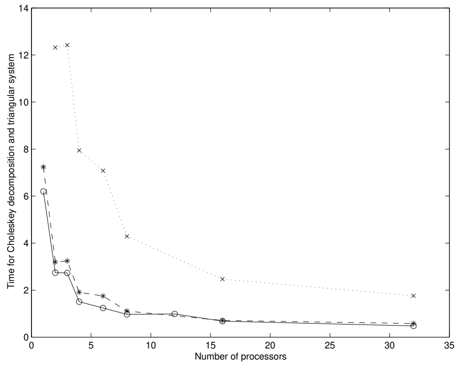

Figure 27: *Time used for Choleskey decomposition and solving the triangular system for different neighbourhoods: 4 neighbours (-o), eight neighbours (–*) and 24 neighbours (..x).*

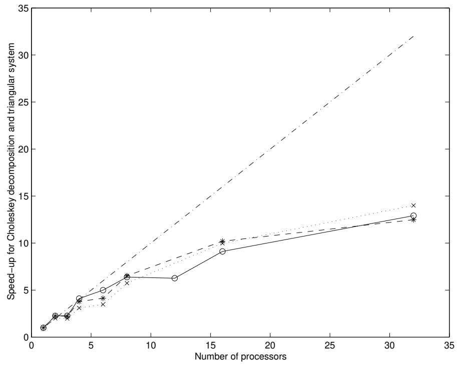

Figure 28: *Speed-up for Choleskey decomposition and solving the triangular system for different neighbourhoods: 4 neighbours (-o), eight neighbours (–*) and 24 neighbours (..x)*

system is too big to be solved on one processor. In figure 28 we can see that the speed-up is about the same for all the different neighbourhood. The bigger neighbourhood, the more work each processor does without communication. But it also gives thicker segmentation sets, and hence more sequential work. In our tests these effects are of the same order. Since the 24 neighbourhood case was too big to be solved on one processor, twice the time used on two processors has been used in the speed-up calculations.

5.3.3 Shape

The shape of the lattice may influences the speed-up. A long and narrow lattice has smaller segmentation sets, and is also sparser (because it has more border elements). Tests are done for four-neighbourhood GMRFs on lattices of size 400×400, 200×800 and 100×1600. note all the lattices have 160000 nodes. Figure 29 shows that the square case is the most expensive one, getting cheaper the longer and narrower the lattice is. From figure 30 we can see that the speed-up is best for the 100×1600 case. The other two have approximately equal speed-up.

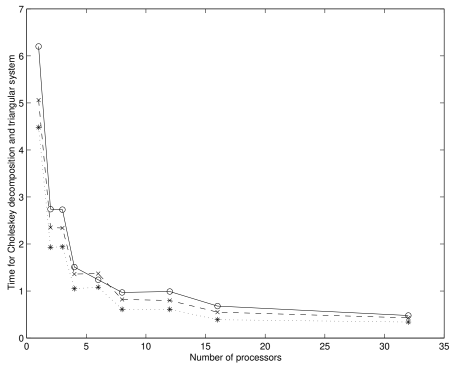

Figure 29: *Time used for Choleskey decomposition and solving the triangular system for different shapes:* 400×400 *neighbours (-o),* 200×800 *(–*) and* 100×1600 *(..x).*

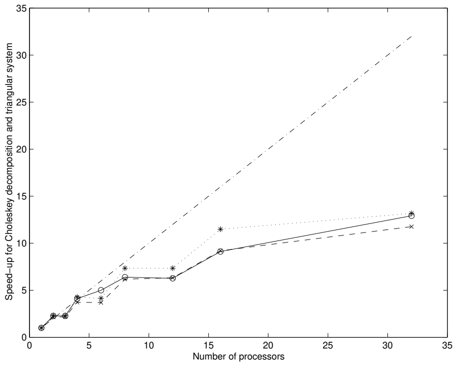

Figure 30: *Speed-up for Choleskey decomposition and solving the triangular system for different shapes:* 400×400 *(-o),* 200×800 *(–*) and* 100×1600 *(..x).*

5.3.4 Graphs

In some cases we have a GMRF on a non-regular graph. Such graphs can be made from political maps; each region is a node and regions are neighbours if they share a border. See figure 31 for an illustration of how to make this kind of graph for a toy-example. We have used one example of such a graph, *germany.graph*. It consist of 544 administration units in Germany, see figure 32. Each of the 544 administration units is a node, having only the nodes of the regions it shares a border with as its neighbours.

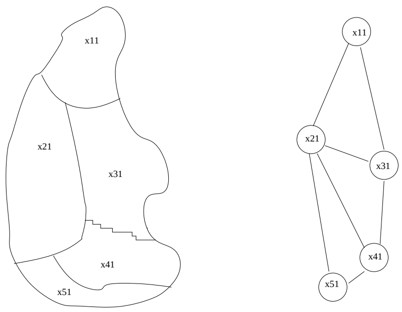

Figure 31: *Example of a mapping from regions to a graph. Each region is a node. Regions are consider neighbours if they share a border, indicated by an edge in the graph.*

Figure 32: *Administrative map of Germany.*

The regions have between 1 and 11 neighbours. This graph is used in [26] for disease mapping, and each variable is then a log relative risk.

Our second example graph is based on 558 regions in Brazil, see figure 33. This graph is used in [16] estimating a space-varying regression model. A space-varying regression model is a standard linear model where the regression parameters are allowed to change in space. In their model there are five regression parameters that are allowed to take different values for each of the 558 regions. This gives each region five nodes, one for each regression parameter. The prior of the regression parameters is such that each node has both the other nodes in its region and all the nodes of their neighbouring regions as its neighbours. We name the graph *brazil5.graph*. It has 2790 nodes, each with between 9

Figure 33: *Administrative map of Brazil.*

and 74 neighbours. Hence it is both larger (more nodes) and denser (more neighbours per node) than *germany.graph*.

Often data are collected on a regional level and collected regularly, e.g. annually. The temporal variation is interesting as well as the spatial, and the space model is extended to a space-time model. For examples of time-space models and applications see e.g. [24], [25] and [42]. We have extended both graphs to become time-space graphs, assuming a Markov property also in the time direction. These graphs consist of T of the original graphs. In addition to the neighbours in space, each element $x_{i,t}$ has the corresponding elements for the time step immediately before $x_{i,t-1}$ and after $x_{i,t+1}$ as its neighbours, see figure 34. A space-time graph is three dimensional and causes a denser neighbourhood structure. The computational complexity of exact sampling is $\mathcal{O}(n^2)$ and the segmentation sets are of size $\mathcal{O}(n^{2/3})$ (n is the total number of nodes). Hence is both the computation cost and the parallelisation more expensive then in the 2D case. To demonstrate how the size of the neighbourhood influence the computation time we have plotted the time spend on one processor for different length time-space graphs against the number of nodes, see figure 35. We observe that larger neighbourhoods gives, as expected, more expensive calculations. For a 100 time step long *brazil5.graph* the problem can not be solved on less than four processor. From the graph in figure 36 we first of observe that there is no speed-up for the five time steps of *germany.graph* or the one time step *brazil.graph*. Both these graphs have about 2700 nodes, and are too small to get any speed-up. For the other graphs tested here there are good speed-up. *germany.graph* with 100 time steps is slightly better than *brazil5.graph* with 10 time steps. They have a speed-up respectively of 1.75 and 1.4 for

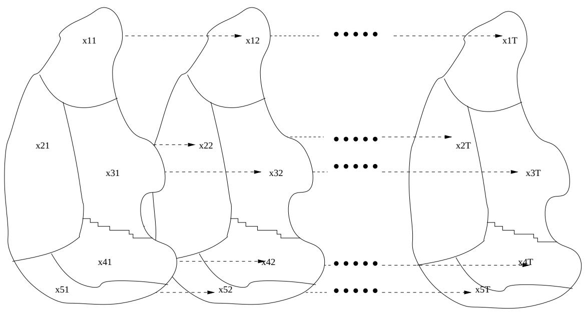

Figure 34: *Illustration of a time extension of a spatial graph model. An area is divided into five regions. Each region can be represented by a node, and regions are neighbours if they share a border. The time extension is done by "copying" the graph for each time-step and connecting them into a chain at region level.*

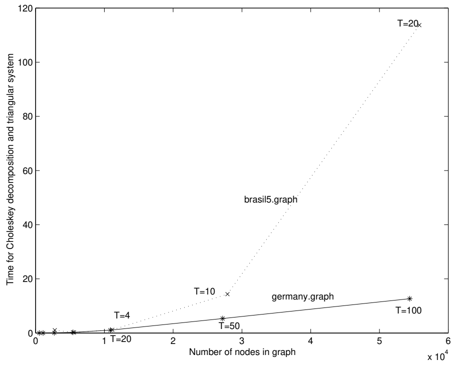

Figure 35: *Time spend on Choleskey factorisation and for solving the triangular system as function of the number of nodes for time-space graphs of germany.graph and brazil5.graph.*

two processors, and 8.9 and 7.8 for 32 processors. The curve for *brazil5.graph* with 20 time steps is placed with perfect speed-up for four processors (the lowest number of processors the problem is solvable). This makes it look much better than the others, but the speed-up from four to 32 processors is about the same for the three tests that give speed-up.

5.4 Discussion and conclusion

We have in this chapter described and tested a method for exact parallel sampling of Gaussian Markov random fields. The computationally demanding part of this is Choleskey decomposition of the precision matrix Q. The method takes advantage of the sparse structure through reordering of the indexes. From a probabilistic point of view the parallelisation

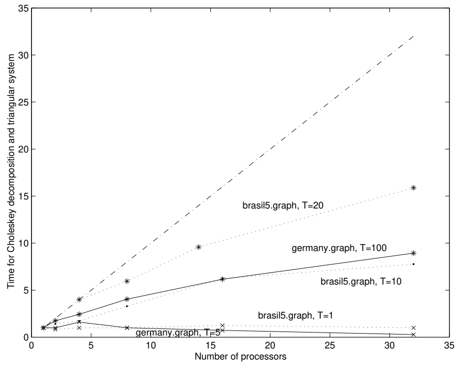

Figure 36: *Speed-up for Choleskey factorisation and for solving the triangular for different number of time steps of the time-space graphs of germany.graph and brazil5.graph.*

is done by finding segmentation sets that give us conditional independence between blocks of variables. For the independent blocks all calculation can be done in parallel. Only the calculations for the segmentation sets have to be done sequentially.

We have tested the parallel exact sampler for both lattices and graphs with different sizes, shapes and densenesses. Many of the tests are done for problems with approximately 150000 nodes, and all these show good speed-up. Sampling becomes more expensive as problems get larger and denser. How this effects the speed-up of the parallelisation depends on the ration of extra computational cost and possible larger segmentation sets. The sampling algorithm also has good scalability; more processors enable us to sample many times larger GMRFs.

Though, we have also seen that problems can be too small to get any speed-up from more processors. For the library we have used the graph should have at least 5000 nodes.

6 Campylobacter infections in north Lancaster - a GMRF approach

In this chapter the declustered campylobacter dataset introduced in section 1.1 and previously studied in [11] is reanalysed. It is based on all record cases of campylobacter, salmonella and cryptosopridia in north Lancaster. The declustered dataset consists of all outbreaks of enteric infections and how many of these that are campylobacter. Each outbreak has attached a location found from the post-code. There are 399 outbreaks of enteric infection in 248 different locations, and 234 of the outbreaks are cases of campylobacter, see figure 37.

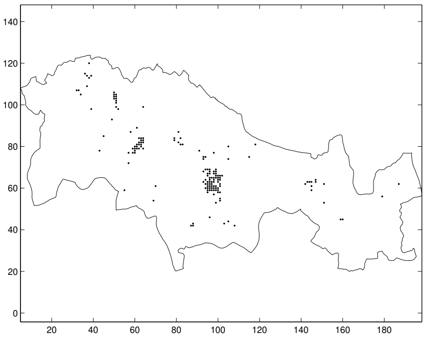

Figure 37: *The locations of the enteric outbreaks*

6.1 Traditional model and results

Diggle et al. [11] modelled the declustered dataset using a binomial trail approach for the "success" (that the outbreak is campylobacter), where the probability of success is independent for each location, given a latent zero-mean stationary Gaussian process $S(x)$. They then set the log relative success probability for location i to

$$\log(\frac{p_i}{1-p_i}) = \beta + s(x_i)$$

where β is a location independent constant and $s(x_i)$ is from the Gaussian process $S(x)$. The Gaussian process has variation σ^2, and a powered exponential correlation function

$$\rho(u) = \exp(-(\alpha u)^\delta)$$

where u is the distance, $\alpha > 0$ and $0 < \delta < 2$.

They set $\delta = 0$ due to identification problems otherwise. This gives hyper-parameters $\theta = (\beta, \sigma^2, \alpha)$, and they use independent uniform priors $[-1.5, 3.5] \times [0, 7] \times [0, 50]$ for these.

A MCMC algorithm (single site) was used to estimate the parameters. The Gaussian process was estimated for each of the 509 locations in the area. They report that their simulations gave posterior marginal mode estimates $\theta = (0.62, 1.0, 6.5)$ and posterior mean estimates $\theta = (0.68, 1.2, 12.6)$. The estimated spatial variation in the log-odds are given in figure 38.

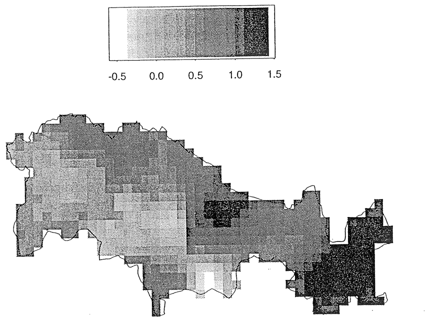

Figure 38: *The estimated log-odds from Diggel et al [11].*

6.2 Geostatistical GMRF model, simulations and results

We will here use the geostatistical GMRF model suggested in chapter 1.3 for the declustered campylobacter dataset. Instead of a latent Gaussian process, we are using a Gaussian Markov Random Field. This is a small change of model that will enable us to use more powerful simulation methods.

The probability for success in the Bernoulli trail is still

$$\log(\frac{p_i}{1-p_i}) = \beta + x_i$$

where β is a location independent constant. Note the change in notation; x_i is here the value of the latent field for lattice element i, and $x = (x_1, x_2, \ldots, x_n)$ is a GMRF. The area is discretised using a 150×210 lattice. This includes a buffer of 25 elements between the outer most observations and the boundary of the lattice for avoiding boundary effects. Some of the out-break locations are in the same element in the lattice. These are then added and treated as one. This reduces the number of outbreak locations to 156. We use an extended second order neighbourhood (each element has 24 neighbours) and the

GMRF-approximation to a GRF suggested in [38]. Coefficients for the elements of Q in the Markov model are fitted to the exponential correlation function

$$\rho(u) = \exp(\frac{-3u}{r})$$

where r is the *range*, which is calculated for $0.2 \leq r \leq 100.4$ in steps of 0.2, and u is the distance when each lattice element has size 1×1. The GMRF precision is set to τ. This gives us three hyper-parameters; $\phi = (\beta, r, \tau)$. See the sections below for our choices of priors for ϕ. The parameters and the latent field are estimated using an one-block update scheme Metropolis-Hastings algorithm, see algorithm 7

Algorithm 7 *One-block update scheme Metropolise-Hastings algorithm*

- Given ϕ^0
- for $i = 0 : (n-1)$
  - Sample $\phi^{new} \sim q(\phi^{new}|\phi^i)$
  - Find mode x^{mode} of $\pi(x|\phi^{new}, y)$
  - Find a GMRF approximation, $\pi_{approx}(x|\phi^{new}, y)$, to $\pi(x|\phi^{new}, y)$ in x^{mode}.
  - Sample $x^{new} \sim \pi_{approx}(x|\phi^{new}, y)$
  - accept/reject
  - if (accept)
    - $\phi^{i+1} = \phi^{new}$ and $x^{i+1} = x^{new}$
  - else
    - $\phi^{i+1} = \phi^i$ and $x^{i+1} = x^i$
- Return $(\phi^0, \phi^1, \ldots, \phi^n)$ and $(x^0, x^1, \ldots x^n)$

First the hyper-parameters are proposed. They are proposed independently, β from a Gaussian distribution with expected value β_{old}, τ uniformly $[\frac{1}{f}\tau, f\tau]$, and r from a discrete uniform distribution $\mod_{[0.2,100.4]}[r - \Delta r, r + \Delta r]$. The proposal for r is modulised such that $0.2 \leq r^{new} \leq 100.4$. Next the mode in $\pi(x|r, \tau, \beta, y)$ is found using Newton's method, and a multivariate Gaussian distribution approximated to $\pi(x|r, \tau, \beta, y)$ at the mode. Using a GMRF as prior for $x|\phi$ also this approximation is a GMRF, see appendix A.1. Our proposal for the latent field is this GMRF. All the parameters are accepted / rejected and updated in one block. A key feature of how good this algorithm is is how close $\pi(x|\phi, y)$ is its Gaussian approximation. Several tests are done for fixed hyper-parameters, see table 3. Given the hyper-parameters this is a Metropolis-Hastings independent sampler, and the acceptance rate is a measure of how close the proposal distribution is the target

	$\tau = 0.1$	$\tau = 1$	$\tau = 10$	$\tau = 100$
$r = 0.2$	0.016	0.653	0.983	1.000
$r = 10$	0.070	0.726	0.976	1.000
$r = 100$	0.400	0.900	0.984	0.999

Table 3: *Acceptance rates for fixed hyper-parameters ($\beta = 0.35$), all based on 1000 iterations.*

Parameter	mean	median	mode
β	0.40	0.37	0.32
τ	10.43	6.13	2.61
r	47.1	45.4	11.7

Table 4: *Posterior marginal mean, median and mode estimates based on the Markov chain with the first 10 000 iterations omitted.*

distribution. Our results suggest that the target distribution is close to Gaussian, except for small values of τ.

Two different sets of priors, prior 1 and prior 2, have been used. Prior 1 is a proper prior. Prior 2 is an improper prior that gives an improper posterior distribution. Trying to sample from a improper distribution is a test of the mixing: If the sampler mixes well, normal convergence diagnostic should indicate lack of convergence. For reference purposes a synthetic dataset with the same data locations and total number of cases n_i as in the Lancaster campylobacter dataset is made. The GMRF model has also been used to analyse the Lancaster campylobacter data in [37], but then with β fixed.

6.2.1 The Lancaster campylobacter data and prior 1 (proper prior)

The result we have in mind when modelling is a smooth underlying field, i.e. a relative large r, with some variance, $\frac{1}{\tau}$. We believe more in a smooth field with low precision than a field with little dependence and high precision. To map this belief into the priors a flat prior for r over its discretation, and a gamma distribution for τ with mean 5 and variance 100 are chosen. A flat truncated prior is used for β (hence we also truncate the proposal). We have chosen to truncate it for $[-4.60, 4.60]$ which corresponds to $0.01 < p < 0.99$. The sampling algorithm described above was used, and the Markov chain ran for 100000 iterations. See figure 39 for trace-plots, histograms and scatter-plot of the simulated values of the hyper-parameters. The cumulative mean for the hyper-parameters are plotted in figure 40. For marginal estimates of the mean, median and mode see table 4. The plots are based on all the samples while the results in table 4 omit a burn-in of a 10000 samples. In the posteriori histograms in figure 39 the prior is included (for β drawn as if it was only defined on $[-1, 3]$).

From the trace-plots in figure 39 there seems to be good mixing. These plots together

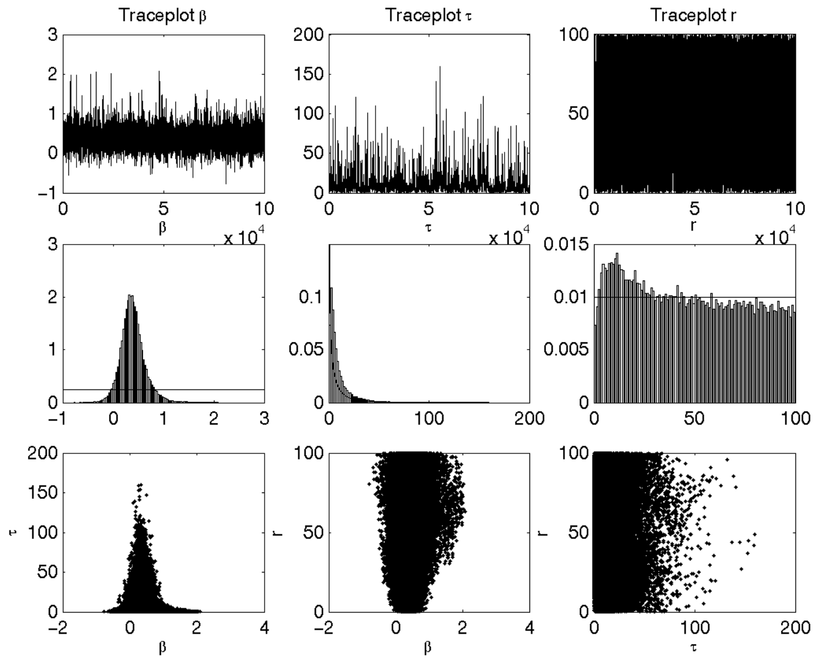

Figure 39: *Trace plots and histograms for the hyper-parameters $\theta = (\beta, \tau, r)$ for the Lancaster dataset with informative priors .*

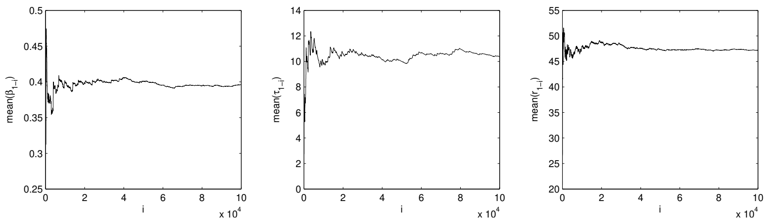

Figure 40: *Cumulated mean for the hyper-parameters $\theta = (\beta, \tau, r)$ for the Lancaster dataset with informative priors .*

with the cumulated means in figure 40 convince us that the Markov chain and the mean estimates for the hyper-parameters have converged after a burn-in of 10000-20000 iterations. The histograms in row two of figure 39 reflects our marginal posteriori distribution for the hyper-parameters. The level parameter β has a mean a bit above what the ratio estimate for p would give ($p_{ratio} = 235/399 = 0.59$ and $\beta_{ratio} = \log(p_{ratio}/(1 - p_{ratio})) = 0.35$). From the histogram the posterior for β seems to have quite light tails, the mode is well defined and the distribution quite symmetric (the right tail is a bit heavier than the left). The posterior for the precession is heavy tailed with a defined mode relatively close to zero. The range parameter r has a posterior very close to its flat prior, though, it has undoubtly a mode between 5 and 25. From the scatter plots on the bottom line in figure 39 we first observe that there are only high precession values (τ) for β values close to β_{ratio}. Extreme βs have to be explained with small precession. The scatter plot of β and r has a fan-shape. Small values of r, i.e. low spatial dependence, only comes with β close to β_{ratio}. With stronger dependence the level of a sampled field x dependents less on β. The last scatter plot does not show much dependency between τ and r. But there are no extreme values of τ for small r. This can be explained from the relationships between β and τ, and β and r mentioned above. Figure 41 contains the estimated mean of the latent field. This

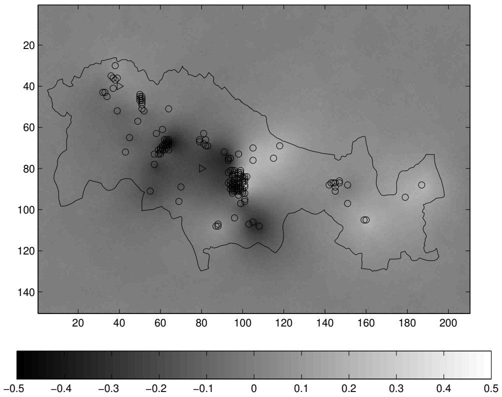

Figure 41: *Estimated mean for the latent field for the Lancaster dataset with informative prior. The locations of the observations are indicated with circles, and* $x_{40,40}$ *and* $x_{80,80}$ *with triangles. .*

estimate is made out of 10 000 iterations. For two lattice elements $(40, 40)$ and $(80, 80)$ the trace-plots and cumulated trace-plots are found in figure 42. The mixing is much faster for the field variable then for the hyper-parameters, a result reported by others, e.g. [12] (note we have only 10 000 iterations for the field variables).

Comparing our results with those in [11] can not be done directly because we have used a different parametrisation. Though the β parameter should be the same, and we are surprised by this difference. We suspect 5000 iterations was not enough for convergence with their sampling method (single site). Also others have reported problems reproducing their results, see [7]. For the latent field (our in figure 41 and their in figure 38) the same trends can be observed. Note that the colour scales are opposite.

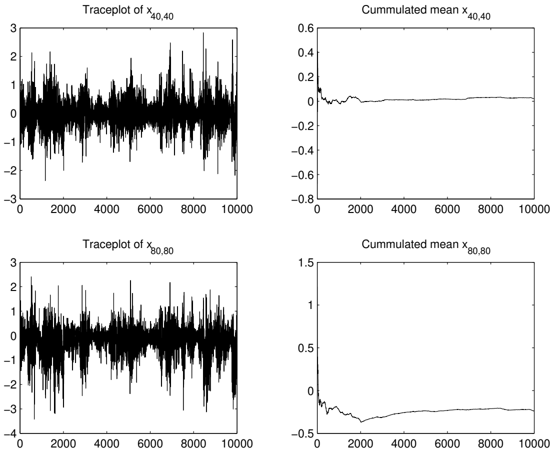

Figure 42: *Trace plots and cumulated mean for the latent field variables $x_{40,40}$ and $x_{80,80}$ for the Lancaster dataset with informative priors.*

Parameter	mean	median	mode
β	0.64	0.64	0.65
τ	9.19	6.11	2.20
r	53.8	56.2	85.6

Table 5: *Posterior marginal mean, median and mode estimates based on the Markov chain with the 10 000 first iterations omitted.*

6.2.2 Synthetic dataset with prior 1

To test the simulation method we have also performed the same analysis using a simulated dataset with $r = 20$, $\tau = 1$ and $\beta = 0.35$. The locations and number of enteric infections are the same as the Lancaster campylobacter dataset. The simulated dataset gave 268 outbreaks of campylobacter out of the 399 enteric infections. This gives a ration value of $p_{ratio} = 268/399 = 0.67$ and the corresponding $\beta_{ratio} = 0.72$. The model and simulation algorithm were identical to those used for the real dataset. See figure 43 for trace-plots, histograms and scatter-plots for the hyper-parameters. The cumulative mean for the hyper-parameters are plotted in figure 44. For marginal estimates for mean, median and mode see table 5. The plots and estimates are presented similarly to those of the real dataset.

Comparing the results for the hyper-parameters for the synthetic data with the real example, the similarities are obvious. The mixing is good (trace-plots in figure 43), mean estimates have converged (figure 44), for β the mean is close to the ratio estimate, and the marginal posterior distributions and the scatter-plots have the same shape in figure 43 and figure 39. Focusing on the differences, poorer mixing of r is observed from the trace-

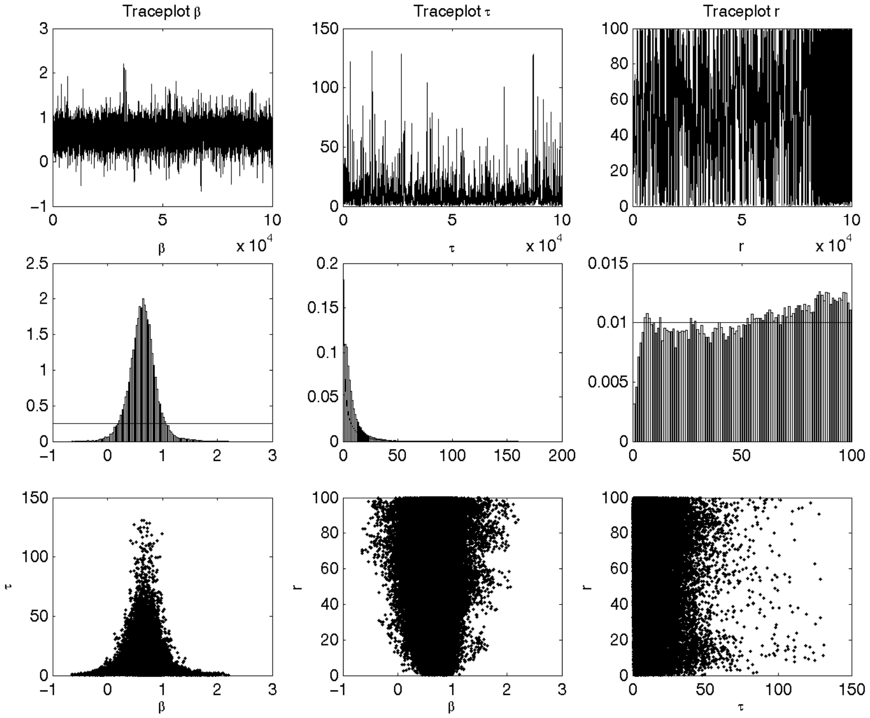

Figure 43: *Trace plots and histograms for the hyper-parameters $\theta = (\beta, \tau, r)$ for the synthetic dataset.*

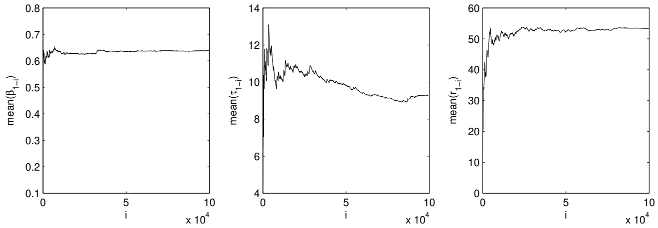

Figure 44: *Cumulated mean for the hyper-parameters $\theta = (\beta, \tau, r)$ for the synthetic dataset with informative priors .*

plot, and slower convergence for τ from the cumulated mean plot. From the cumulated mean plot in figure 44 one can doubt convergence. But omitting the first 10 000 iterations changes this impression. The histogram for r in figure 43 we find an apparently uniform posterior. For the synthetic dataset the posterior of β is almost perfectly symmetric. This causes a more symmetric fan shape in the scatter plot for β and r.

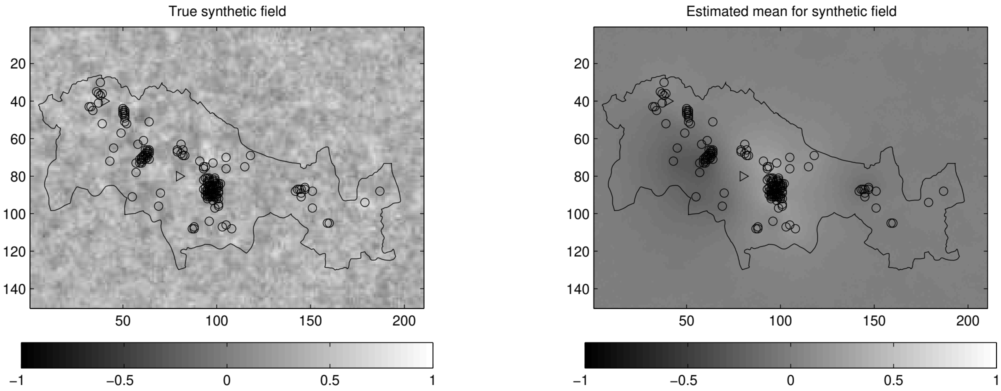

Figure 45: *The simulated latent field (left) and the estimated mean (right). The locations of the observations are indicated with circles, and $x_{40,40}$ and $x_{80,80}$ with triangles.*

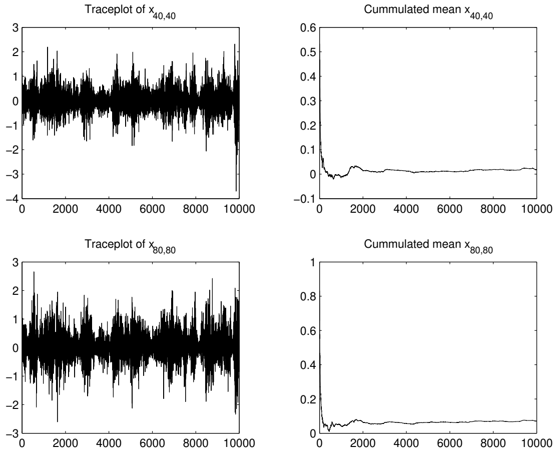

Figure 46: *Trace plots and cumulated mean for the latent field variables $x_{40,40}$ and $x_{80,80}$ for the synthetic dataset.*

The synthetic latent field is given in figure 45 together with the estimate. The mean value of the simulated field is 0.37, its minimum value -0.13 and its maximum value 0.78. From these values and figure 45 it is seen that it has a non-zero level. In the posterior estimates most of this level is moved to β, and the posterior mean estimate of x has of course level zero. It is also much smoother than the true field, as we would expect for a mean estimate with few data locations. Trace-plots and cumulated mean plots for element $(40, 40)$ and $(80, 80)$ are found in figure 46. As for the real data the latent field variables have a short burn-in and mix well.

Results in [37] suggest that the data do not contain much information about r and τ. Our simulations for both the real and the synthetic dataset agree with this, the posteriors for these parameters are close to their priors.

6.2.3 The Lancaster campylobacter data with prior 2 (improper prior)

We want to test the sampling method on a case where the posterior distribution is improper. If a realistic number of simulations indicates an improper distribution it is a sign of good mixing and fast "convergence" for the sampler. The priors of the hyper-parameters are independent. We use a flat prior for β, for r $\pi(r) \propto \frac{1}{r}$ and $\pi(\tau) \propto \frac{1}{\tau}$. This gives an improper posterior distribution, see appendix A.2. Note this model should then be very similar to the model proposed by Diggel et al. and would be an obvious choice for the not careful enough analyst.

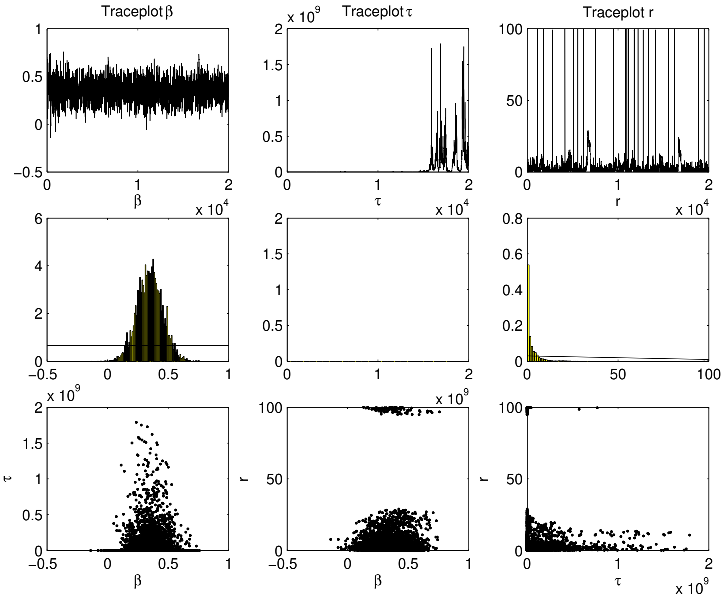

Figure 47: *Trace plots and histograms for the hyper-parameters $\theta = (\beta, \tau, r)$ for the Lancaster campylobacter dataset with prior 2.*

Running the one block update Metropolis-Hasting sampler described above 20000 times gave trace plots and histograms for ϕ as shown in figure 47. The trace-plot and histogram

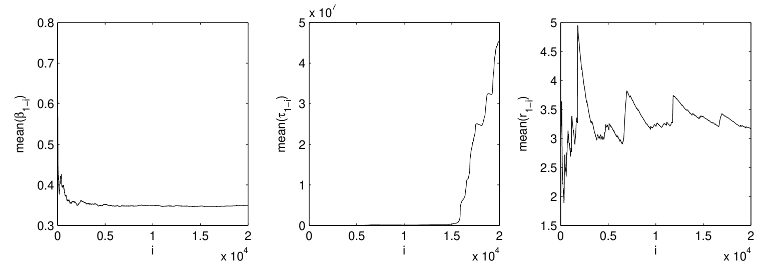

Figure 48: *Cumulated mean for the hyper-parameter τ for the Lancaster campylobacter dataset with prior 2. Note the change in scaling as the number of iterations increase.*

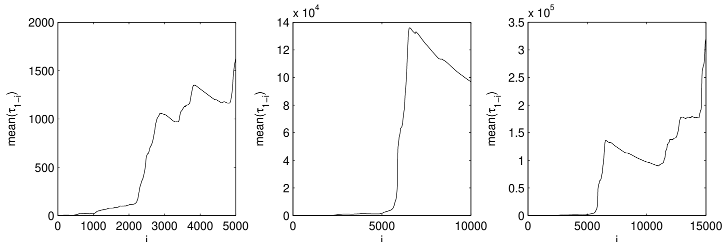

Figure 49: *Trace plots and histograms for the hyper-parameters $\theta = (\beta, \tau, r)$ with for the Lancaster campylobacter dataset with prior 2 .*

for τ indicate that $\pi(\tau|y)$ is not a proper distribution. From the histogram it seems to have too heavy tails, and the estimated mean of τ is not converging. In figure 49 the cumulated mean is plotted for different numbers of iterations. Already after 5000 iterations divergence is indicated. We further observe that the mean parameter β mixes well and has stabilised. The sample mean is $\bar{\beta} = 0.35$, which equals the β we would get using the ratio as our estimate of p.

We have run several Markov chains with different initial values. After a burn-in period, which can last for some thousand iterations with initial value for $r \approx 200$, they all indicate the same result; a very low r and an unbounded τ.

Up-dating all variables in one block has enabled us to see the lack of convergence in the hyper-parameters, and to suspect an improper posterior distribution.

6.3 Parallelisation and performance

The parallelisation is done on domain level. Each processor is dedicated to a part of the lattice and the data located in this part of the region, see figure 50 for an illustration.

The processors have to communicate when the mode is found, during sampling and

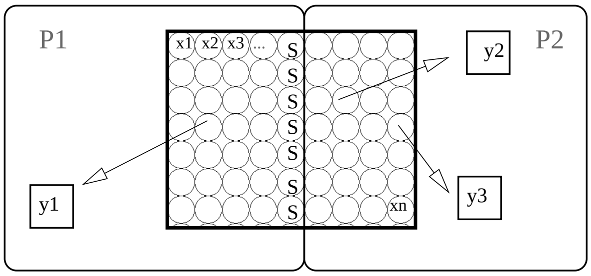

Figure 50: *How the latent field* $(x_1, x_2, \ldots, x_M)$*, and the data,* $(y_1, y_2, \ldots, y_n)$ *can be split between processors.*

evaluation of the sample. Each processor evaluates the likelihood for its own data. Information from all processors has to be gathered before the acceptance probability is found. Accepting/ rejecting a sample, and proposing new hyper-parameters is done by a master processor. This could have been done by each processor if they had a common random number stream (in addition to an unique one). For speed-up for the Choleskey decomposition and solving the triangular system see figure 51.

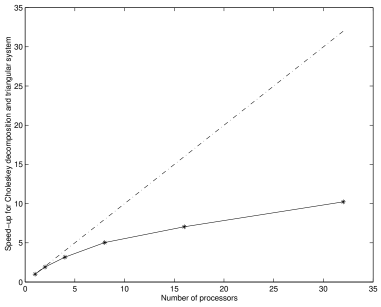

Figure 51: *Speed-up for Choleskey decomposition and solving the triangular system for the lattice used in the Campylobacter infections example.*

The simulations have been run on four processors, and 10 000 iterations take about 15 hours, or 11 iterations per minute. The speed-up is then close to perfect, and the gained speed makes a difference for the user. To wait a week for the simulation results is much better then waiting for four weeks.

7 Discussion and Conclusion

The purpose of this report was twofolded:

1 To introduce the geostatistical GMRF models.

2 To introduce a parallelisation of a robust one-block Metropolis-Hastings sampler for the geostatistical GMRF model.

The MCMC simulation is often the critical stage when making inference from geostatistical models. Slow converges is hard to detect, and the Markov chains are often not run long enough due to restricted access of computer resources. The geostatistical GMRF models are a class of geostatistical models that are close to the traditional models, but are cheaper to simulate from. Instead of assuming a latent Gaussian process we discretise using a lattice and make a latent Gaussian Markov random field (GMRF). This GMRF is given a prior such that it imitates a Gaussian process.

Mixing is a frequent problem for MCMC users. Empirical results show that one-block updates of all the parameters can improve the mixing. The geostatistical GMRF model enable us to do one-block updates. The simulations are still computational expensive, and a way of dealing with this is to use more then one processor. An extra advantage for the geostatistical GMRF models is its nice parallelisation properties. By fare the most expensive parts of the GMRF sampling is doing Choleskey decomposition of the sparse precession matrices whose structure is given by the neighbourhood of the latent field. This is a well known problem in numerical linear algebra. Libraries for efficient exact solutions are available, also for parallel computers. We have shown through experiments that a parallel implementation can give us both good speed-up and enable us to sample and evaluate larger GMRFs then one processor could.

WSMP for parallel solving of sparse linear systems for linux clusters has recently been released. This will make the GMRF simulation more available. Few have access to a super-computer, but a cluster can be made up of colleges computers, and else unused computational resources are available for our demanding simulations.

We believe the more robust simulations the geostatistical GMRF models enable us to do improves the reliability of our analysis. We further see parallelisation as the future way of getting more computational resources.

Acknowledgements

The research was founded by grants from the Research Council of Norwegian (project 133695/432) and much of the work was done while I was visiting Department of Statistics at Trinity College Dublin as a part of the IITAC-project (The Institute for Information Technology and Advanced Computation). Thanks to the Trinity Centre for High Performance Computing and to Anshul Gupta at IBM Watson Research Center for their technical support, and to Håvard Rue, Matt Whiley and Simon Wilson for interesting discussions.

A Appendix

A.1 Gaussian approximation to $\pi(x|y,\phi)$ in x_ϕ^{mode}

We will in this appendix show that the Gaussian approximation to $\pi(x|y,\phi)$ when $\pi(x|\phi)$ is a GMRF is a GMRF with the same non-zero structure of the precision matrix as for the prior. In our example data are assumed mutually independent given the latent field x, and can be written as;

$$\pi(y|x) \propto \prod_{i=1}^{N} \exp(g(x_i))$$

Hence we can write

$$\begin{aligned}
\pi(x|y,\phi) &\propto \pi(x|\phi)\pi(y|x) \\
&\propto \exp(-\frac{1}{2}\tau x^T Q x + \sum_{i=1}^{N} g(x_i)) \\
&\approx \exp(-\frac{1}{2}\tau x^T Q x + \sum_{i=1}^{N}(a_i + b_i x_i + c_i x_i^2)) \\
&= \exp(-\frac{1}{2}x^T(\tau Q + \mathrm{diag}(c))x + \sum_{i=1}^{N}(a_i + b_i x_i)
\end{aligned}$$

where $a_i + b_i x_i + c_i x_i^2$ is the 2nd order Taylor approximation of $g(x_i)$ in x_ϕ^{mode} and $\mathrm{diag}(c)$ is a diagonal matrix containing $(c_1, c_2, \ldots, c_n)$. We see that the precision matrix of the approximation is $(\tau Q + \mathrm{diag}(c))$, and hence the approximation is a GMRF with the same conditional dependence structure as the prior $\pi(x|\phi)$.

A.2 Proof that $\pi(\tau) \propto \frac{1}{\tau}$ and $\beta \propto 1$ causes improper posterior distribution

This proof is done along the same lines as the proof of theorem 1 in [33]. We assume the same model as in section 6.2, but move the level β from the likelihood to the latent field:

$$\log(\frac{p_i}{1-p_i}) = x_i$$

and

$$x \sim N(\tilde{\beta}, (\tau Q(r))^{-1}$$

where $\tilde{\beta}$ is a vector of length n and all elements have value β. The posterior distribution of τ and β is proper iff the marginal distribution of the data

$$\pi(y) \propto \int \pi(y|x)\pi(x|r,\tau,\beta)\pi(r)\pi(\beta)\pi(\tau)d\beta d\tau dr dx$$

is integrable. We set $\pi(x|r,\tau,\beta) \sim N_x(\beta\mathbf{1},(\tau Q(r))^{-1})$ where $\mathbf{1}$ is a of size n consisting of ones. We will focus on β and τ, and therefore suppress ϕ in $Q(\phi)$. First we rewrite $\pi(x|r,\tau,\beta)$ as a function of β:

$$\begin{aligned}
\pi(x|r,\tau,\beta) &= N_x(\beta\mathbf{1},(\tau Q)^{-1}) \\
&= (2\pi)^{-\frac{n}{2}}\tau^{\frac{n}{2}}\det(Q)\exp-(\frac{1}{2}\tau(x-\beta\mathbf{1})^T Q(x-\beta\mathbf{1})) \\
&= (2\pi)^{-\frac{n}{2}}\tau^{\frac{n}{2}}\det(Q)\exp-(\frac{1}{2}\tau(\beta-\mathbf{1}^{-1}x)^T\mathbf{1}^T Q\mathbf{1}(\beta-\mathbf{1}^{-1}x) \\
&= (2\pi)^{-\frac{n-1}{2}}\tau^{\frac{n-1}{2}}|\det(Q)|^{1/2}|\mathbf{1}^T Q\mathbf{1}|^{-1/2}N_\beta(\mathbf{1}^{-1}x,(\mathbf{1}^T Q\mathbf{1})^{-1}) \\
&= \text{const}(\phi)\tau^{\frac{n-1}{2}}N_\beta(\mathbf{1}^{-1}x,(\mathbf{1}^T Q\mathbf{1})^{-1})
\end{aligned}$$

Hence can $\pi(x|r,\tau,\beta)$ be written as a product of a constant (as a function of x and β) and a Gaussian density for β. We write up the expression for $\pi(y)$, and integrate out β:

$$\begin{aligned}
\pi(y) &\propto \int\int\int \pi(y|x)\pi(r)\pi(\tau)[\int \pi(x|\beta,\phi,\tau)\pi(\beta)d\beta]d\tau d\phi dx \\
\pi(y) &\propto \int\int\int \pi(y|x)\pi(r)\pi(\tau)\text{const}(\phi)\tau^{\frac{n-1}{2}}\underbrace{[\int_{-\infty}^{\infty} N_\beta(\mathbf{1}^{-1}x,(\mathbf{1}^T Q\mathbf{1})^{-1})d\beta]}_{=1}d\tau d\phi dx \\
\pi(y) &\propto \int\int \pi(y|x)\pi(\phi)\text{const}(\phi)[\int_0^\infty \tau^{\frac{n-3}{2}}d\tau]d\phi dx
\end{aligned}$$

which is not integrable for τ. Hence the posterior $\pi(x,\theta|y)$ is not proper.

References

[1] G. T. Barkema and T. MacFarland. Parallel simulation of the Ising model. *Physical Review E*, 50(2):1623–1628, 1994.

[2] J. Besag, J. York, and A. Mollie. Bayesian image restoration, with two applications in spatial statistics (with discussion). *Annals of the Institute of Statistical Mathematics*, 43(1):1–59, 1991.

[3] S. Brooks and G. Roberts. Convergence assessment techniques for Markov chain Monte Carlo. *Statistics and Computing*, 1998.

[4] N. Carter and R. Kohn. On Gibbs sampling for state space models. *Biometrika*, 1994.

[5] R. Chellappa and A. Jain. Compound Gauss-Markov Random Fields for parallel image processing. In F. C. Jeng, J. W. Woods, and S. Rastogi, editors, *Markov Random Field, Theory and Application*. Academic Press, 1991.

[6] R. Chellappa and A. Jain, editors. *Markov Random Field, Theory and Application*. Academic Press, 1991.

[7] O. F. Christensen, J. Møller, and R. Waagepetersen. Analysis of spatial data using generalized linear mixed models and Langevin-type Markov chain Monte Carlo. Technical report, Department of Mathematical Sciences, Aalborg University, 2000.

[8] M. Cowels and B. Carlin. Markov chain Monte Carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 1996.

[9] N. Cressie. *Statistics for Spatial Data*. Whiley, New York, 2 edition, 1993.

[10] D. E. Culler and J. P. Singh, editors. *Parallel Computing Architecture*. Morgan Kaufmann Publishers, San Francisco, California, 1999.

[11] P. J. Diggel, J. A. Tawn, and R. A. Moyeed. Model-based geostatistics (with discussion). *Applied Statistics*, 47:299–350, 1998.

[12] P. J. Diggle, P. J. Ribeiro jr, and O. F. Christensen. An introduction to model-based geostatistics. In J. Møller, editor, *Spatial statistics and Computational methods*. Springer Verlag New York, 2003.

[13] J. J. Dongarra and I. S. Duff. *Numerical linear algebra for high performance computers*. SIAM, Addison, 1998.

[14] M. J. Flynn. Some computer organizators and their effectiveness. *IEEE Transactions on Computing*, 21:948–960, 1972.

[15] I. Foster. *Designing and Building Parallel Programs*. Wesley, Addison, 1995.

[16] D. Gamerman, A.R.B. Moreira, and H. Rue. Space-varying regression models: specifications and simulations. *Computational Statistics and Data Analysis*, 2003. to appear in Special Issue on Computational Economics.

[17] A. Gelman, G. O. Roberts, and W. R. Gilks. Efficient Metropolis jumping rules. In J. Bernardo, J. Berger, A. Dawid, and A. Smith, editors, *Bayesian Statistics Vol. 5*. Oxford University Press, 1995.

[18] S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution and Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1984.

[19] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman & Hall, London, 1996.

[20] P. J. Green. MCMC in image analysis. In W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors, *Markov chain Monte Carlo in practice*, pages 381–400. Chapman & Hall, 1996.

[21] A. Gupta. WSMP: Watson sparse matrix package (part-1: direct solution of symmetric sparse systems). *Technical Report RC 21886 98462*, 2000. http://www.cs.umn.edu/ agupta/wsmp.html.

[22] A. Gupta, M. Joshi, and V. Kumar. WSMP: A high-performance serial and parallel sparse linear solver. *Technical Report RC 22038 (98932)*, 2001. http://www.cs.umn.edu/ agupta/wsmp.html.

[23] W. K. Hasting. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[24] L. Knorr-Held. Bayesian modelling of inseparable space-time variation in disease risk. *Statistics in medicine*, 19:2555–2567, 2000.

[25] L. Knorr-Held and J. Besage. Modelling risk from a disease in time and space. *Statistics in medicine*, 17:2045–2060, 1998.

[26] L. Knorr-Held and H. Rue. On block updating in Markov random field models for disease mapping. *Scandinavian Journal of Statistics*, 29(4):597–614, 2002.

[27] J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Verlag New York, 2001.

[28] J. S. Liu, W.H. Wong, and A. Kong. Covariance structure of the Gibbs sampler with applications to the compairison of estimators and augmentation schemes. *Biometrika*, 1994.

[29] K. Mengersen, C. P. Robert, and C. Guihenneuc-Jouyaux. MCMC convergence diagnostics. a reviewww (with discussion). In J. Bernardo, J. Berger, A. Dawid, and A. Smith, editors, *Bayesian Statistics 6*. Oxford University Press, 1999.

[30] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculation by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[31] A. Mollie. Bayesian mapping of disease. In W.R. Gilks, S. Richardson, and D.J. Spiegelhalter, editors, *Markov chain Monte Carlo in practice*, pages 359–380. Chapman & Hall, 1996.

[32] D.W. Murray, A. Kashko, and H. Buxton. A parallel approach to the picture restoration algorithm of Geman and Geman on an SIMD machine. *Image and Vision Comput.*, 1986.

[33] R. Natarajan and R.E. Kass. Reference Bayesian methods for generalized linear mixed models. *Journal of the American Statistical Association*, 95:227–237, 2000.

[34] M. E. J. Newman and G. T. Barkema. *Monte Carlo Methods in Statistical Physics*. Oxford University Press, 1999.

[35] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer Verlag, New York, 1999.

[36] H. Rue. Fast sampling of Gaussian Markov random fields. *Journal of the Royal Statistical Society, Series B*, 63(2):325–338, 2001.

[37] H. Rue, I. Steinsland, and S. Erland. Approximating hidden Gaussian Markov random fields. Preprint Statistics 1/2003, Norwegian University of Science and Technology, 2003.

[38] H. Rue and H. Tjelmeland. Fitting Gaussian Markov Random Fields to Gaussian Fields. *Scandinavian Journal of Statistics*, 29, 2002.

[39] M.J. Schervish. Applications of parallel computation to statistical inference. *Journal of the American Statistical Association*, 1988.

[40] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI - The Complete Reference: Volume 1, The MPI Core (Second ed.)*. MIT Press, Cambridge, Massachusetts, 1998.

[41] R.H. Swendsen and J.S. Wang. Nonuniversisal critical dynamics in Monte Carlo simulations. *Phys. Rev. Lett.*, 1987.

[42] L.A. Walter, B.P. Carlin, H. Xia, and A.E. Gelfald. Hierarchical spatio-temporal mapping of disease rates. *Journal of the American Statistical Association*, 92:607–617, 1997.

[43] M. Whiley and S.P. Wilson. Parallel algorithm for Markov chain Monte Carlo methods in latent spatial Gaussian models. Accepted by Statistics and Computing, 2003.

[44] D. J. Wilkinson. Parallel Bayesian computation. In E.J. Kontoghiorghes, editor, *Handbook of Parallel Computing and Statistics*, Statistics: Textbooks and Monographs. Marcel Dekker, New York, 2004. To appear.

[45] G. Winkler. *Image analysis, Random Fields and Dynamic Monte Carlo Methods*. Springer Verlag, Berlin Heidelberg, 1995.