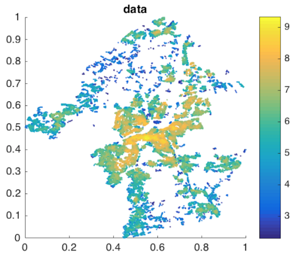
Lecture 5: Gaussian Markov random fields

Spatial Statistics and Image Analysis

David Bolin
University of Gothenburg

Gothenburg
April 8, 2019

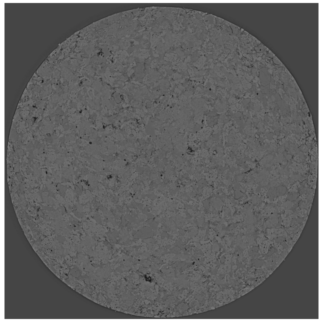
Example project 1



Data from the Spatial Morphology Group at Chalmers

- For city planning it is important to know how the structure of the city affects things such as population density, and housing prices.
- The aim is to derive a spatial model for predicting population density or housing prices using various explanatory variables.

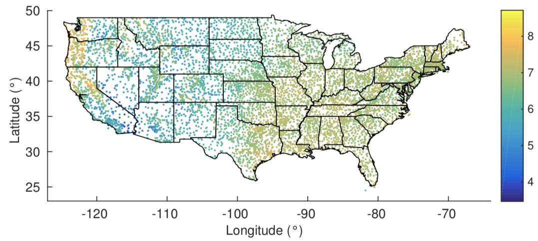
Example project 2



Data from AstraZeneca

- For the production of medical tablets, it is important to know how the manufacturing process affects the composition.
- To do this, one first needs to be able to identify the different components in the tablet based on micro-CT images.
- The goal of this project is to design a method for image segmentation of such images.

Example project 3



- Download a dataset of temperature or precipitation measurements.
- Develop geostatistical and machine learning methods for predicting temperature or precipitation.
- Compare the models using cross-validation.

Gaussian random fields

So far, we have looked at models

$$Y_i = \mathbf{B}(\mathbf{s}_i)\boldsymbol{\beta} + X(\mathbf{s}_i) + \varepsilon_i, \quad i = 1, \ldots, N$$

where $\varepsilon_i \sim \mathsf{N}(0, \sigma_e^2)$ and $X(\mathbf{s})$ is a Gaussian random field.

- The data vector $\mathbf{Y} = (Y_1, \ldots, Y_N)^T$ has distribution $\mathsf{N}(\mathbf{B}\boldsymbol{\beta}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_X + \sigma_e^2\mathbf{I}$.
- log-likelihood: $\ell(\mathbf{Y}; \boldsymbol{\beta}, \boldsymbol{\theta}) = \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{Y} - \mathbf{B}\boldsymbol{\beta})^T\boldsymbol{\Sigma}^{-1}(\mathbf{Y} - \mathbf{B}\boldsymbol{\beta})$.
- Kriging: $\mathsf{E}(\mathbf{Y}_0|\mathbf{Y}, \boldsymbol{\beta}, \boldsymbol{\theta}) = \mathbf{B}(\mathbf{s}_0)\boldsymbol{\beta} + \mathbf{r}\boldsymbol{\Sigma}^{-1}(\mathbf{Y} - \mathbf{B}\boldsymbol{\beta})$, where $\mathbf{r}_i = \mathsf{C}(Y_0, Y_i)$.
- Sampling: $\mathbf{Y}_s = \mathbf{B}\boldsymbol{\beta} + \mathbf{R}^T\mathbf{e}$, where $\mathbf{e} \sim \mathsf{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{R}^T\mathbf{R} = \boldsymbol{\Sigma}$ is the Cholesky factorization.

Computation times for a GMRF

Assume that $\mathbf{x}$ is an image of size $N = n \times n$, chosen as a GMRF specified using the stencil

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Let us now sample $\mathbf{x}$ and measure

1. The RAM memory required.
2. The computation time for performing the necessary steps.

The following table gives some results for the sampling on a standard laptop.

	time (s)	Memory (MB)
n = 50	0.012	0.21
n = 100	0.054	0.83
n = 150	0.177	1.88

Implementation aspects

Consider the problem of sampling. Two important aspects are

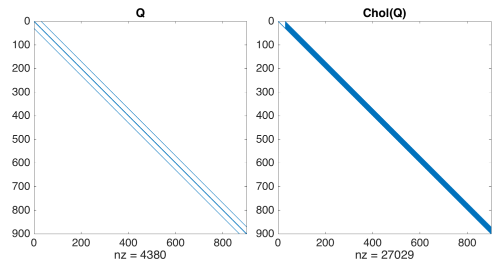
1. The RAM memory required: This is dominated by the memory required to store $\boldsymbol{\Sigma}$, which has $\mathcal{O}(N^2)$ unique elements.
2. The computation time for performing the necessary steps: Compute $\boldsymbol{\Sigma}$, compute the Cholesky factorization $\boldsymbol{\Sigma} = \mathbf{R}^T\mathbf{R}$, solve $\mathbf{x} = \mathbf{R}^T\mathbf{e}$ with $\mathbf{e} \sim \mathsf{N}(\mathbf{0}, \mathbf{I})$. This requires $\mathcal{O}(N^3)$ flops.

Assume that $\mathbf{x}$ is an image of size $N = n \times n$. The following table gives some results for the sampling on a standard laptop.

	time (s)	Memory (MB)
n = 50	1.1	47.7
n = 100	23.4	762.9
n = 150	272.5	3862.4

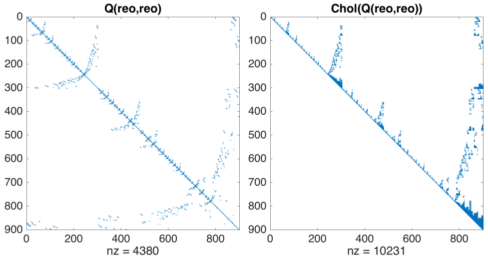
An image of size 150×150 is not a very large image!

Sparsity of $\mathbf{Q}$ and $\mathbf{R}$



- The crucial aspect of computations with GMRFs is that the Cholesky factor $\mathbf{R}$ is sparse.
- However, it is often less sparse than the precision matrix $\mathbf{Q}$. The additional non-zero nodes is usually called fill-in.
- We can reduce the fill-in by reordering the nodes.

Sparsity using reorderings



- Finding the optimal reordering is an NP-hard problem, but there are many fast methods for finding good reorderings.
- The approximate minimum degree (AMD) reordering is generally a good option.
- The images above are obtained with `reo = amd(Q)` in Matlab.
- If you use reorderings, remember to also reorder the observations, covariates, etc. using the same reordering.