Lecture 3: Simulation algorithms

Gaussian Markov random fields

David Bolin
Chalmers University of Technology
January 26, 2015

The sparse precision matrix

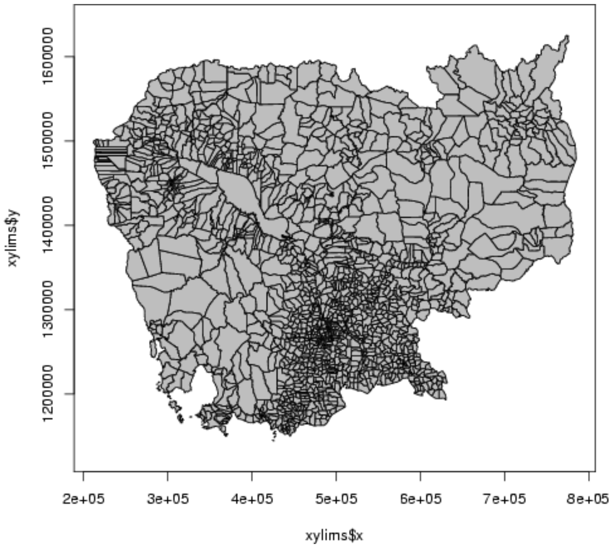
Recall that

$$E(x_i - \mu_i \mid \mathbf{x}_{-i}) = -\frac{1}{Q_{ii}} \sum_{j \sim i} Q_{ij}(x_i - \mu_j), \qquad \mathrm{Prec}(x_i \mid \mathbf{x}_{-i}) = Q_{ii}$$

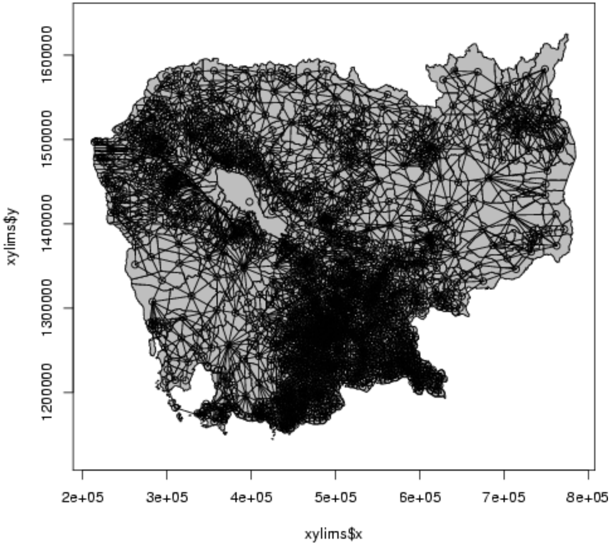
In most cases:

- Total number of neighbours is $\mathcal{O}(n)$.
- Only $\mathcal{O}(n)$ of the n^2 terms in $\mathbf{Q}$ will be non-zero.
- Use this to construct exact simulation algorithms for GMRFs, using numerical algorithms for sparse matrices.

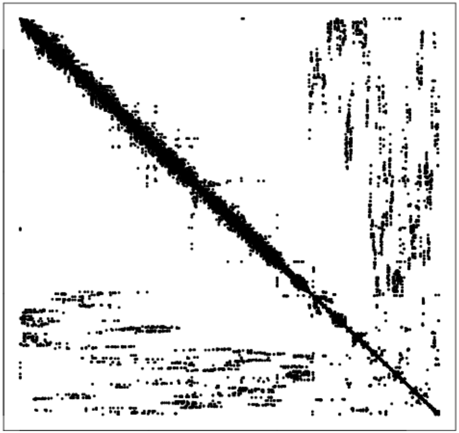
Example of a typical precision matrix



Example of a typical precision matrix



Example of a typical precision matrix



Each node have on the average 7 neighbours.

Simulation algorithms for GMRFs

Can we take advantage of the sparse structure of $\mathbf{Q}$?

- It is faster to factorise a sparse $\mathbf{Q}$ compared to a dense $\mathbf{Q}$.
- The speedup depends on the "pattern" in $\mathbf{Q}$, not only the number of non-zero terms.

Our task

- Formulate all algorithms to use only sparse matrices.
- Unconditional simulation
- Conditional simulation
    - Condition on a subset of variables
    - Condition on linear constraints
    - Condition on linear constraints with normal noise
- Evaluation of the log-density in all cases.

The result

In most cases, the cost is

- $\mathcal{O}(n)$ for temporal GMRFs
- $\mathcal{O}(n^{3/2})$ for spatial GMRFs
- $\mathcal{O}(n^2)$ for spatio-temporal GMRFs

including evaluation of the log-density.

Condition on k linear constraints, add $\mathcal{O}(k^3)$.

These are general algorithms only depending on the graph $\mathcal{G}$ not the numerical values in $\mathbf{Q}$.

The core is numerical algorithms for sparse matrices.

Cholesky factorisation

If $\mathbf{A} > 0$ be a $n \times n$ positive definite matrix, then there exists a unique Cholesky triangle $\mathbf{L}$, such that $\mathbf{L}$ is a lower triangular matrix with positive diagonal elements, and

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

This factorisation is the basis for *solving* systems like

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \text{or} \quad \mathbf{A}\mathbf{X} = \mathbf{B}$$

for k right hand sides, or equivalently, computing

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \quad \text{or} \quad \mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$$

How to compute the Cholesky factorisation

$$Q_{ij} = \sum_{k=1}^{j} L_{ik}L_{jk}, \quad i \geq j.$$

$$v_i = Q_{ij} - \sum_{k=1}^{j-1} L_{ik}L_{jk}, \quad i \geq j,$$

Then

- $L_{jj}^2 = v_j$, and
- $L_{ij}L_{jj} = v_i$ for $i > j$.

If we know $\{v_i\}$ for fixed j, then

$$L_{jj} = \sqrt{v_j} \quad \text{and} \quad L_{ij} = v_i/\sqrt{v_j}, \quad \text{for} \quad i = j+1, \ldots, n.$$

This gives the jth column in $\mathbf{L}$.

Cholesky factorization of $\mathbf{Q} > 0$

Algorithm 1 Computing the Cholesky triangle $\mathbf{L}$ of $\mathbf{Q}$

```
1: for $j = 1$ to $n$ do
2:     $v_{j:n} = Q_{j:n,j}$
3:     for $k = 1$ to $j - 1$ do $v_{j:n} = v_{j:n} - L_{j:n,k}L_{jk}$
4:     $L_{j:n,j} = v_{j:n}/\sqrt{v_j}$
5: end for
6: Return $\mathbf{L}$
```

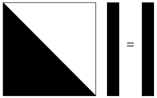
The overall process involves $n^3/3$ flops.

Solving linear equations

Algorithm 2 Solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ where $\mathbf{A} > 0$

```
1: Compute the Cholesky factorisation, $\mathbf{A} = \mathbf{L}\mathbf{L}^T$
2: Solve $\mathbf{L}\mathbf{v} = \mathbf{b}$
3: Solve $\mathbf{L}^T\mathbf{x} = \mathbf{v}$
4: **Return** $\mathbf{x}$
```

Step 2 is called *forward-substitution* and cost $\mathcal{O}(n^2)$ flops.



The solution $\mathbf{v}$ is computed in a forward-loop

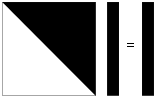
$$v_i = \frac{1}{L_{ii}}(b_i - \sum_{j=1}^{i-1} L_{ij}v_j), \quad i = 1, \ldots, n \tag{1}$$

Solving linear equations

Algorithm 2 Solving $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} > 0$

```
1: Compute the Cholesky factorisation, $\mathbf{A} = \mathbf{LL}^T$
2: Solve $\mathbf{Lv} = \mathbf{b}$
3: Solve $\mathbf{L}^T\mathbf{x} = \mathbf{v}$
4: **Return** $\mathbf{x}$
```

Step 3 is called *back-substitution* and costs $\mathcal{O}(n^2)$ flops.



The solution $\mathbf{x}$ is computed in a backward-loop

$$x_i = \frac{1}{L_{ii}}(v_i - \sum_{j=i+1}^{n} L_{ji}x_j), \quad i = n, \dots, 1 \tag{2}$$

We do not need to compute inverses

To compute $\mathbf{A}^{-1}\mathbf{B}$ where $\mathbf{B}$ is a $n \times k$ matrix, we do this by computing the solution $\mathbf{X}$ of

$$\mathbf{A}\mathbf{X}_j = \mathbf{B}_j$$

for each of the k columns of $\mathbf{X}$.

Algorithm 3 Solving $\mathbf{A}\mathbf{X} = \mathbf{B}$ where $\mathbf{A} > 0$

```
1: Compute the Cholesky factorisation, $\mathbf{A} = \mathbf{L}\mathbf{L}^T$
2: for $j = 1$ to $k$ do
3:     Solve $\mathbf{L}\mathbf{v} = \mathbf{B}_j$
4:     Solve $\mathbf{L}^T\mathbf{X}_j = \mathbf{v}$
5: end for
6: Return $\mathbf{X}$
```

Sample $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{Q}^{-1})$

If $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$ and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, then $\mathbf{x}$ defined by

$$\mathbf{L}^T\mathbf{x} = \mathbf{z}$$

has covariance

$$\mathrm{Cov}(\mathbf{x}) = \mathrm{Cov}(\mathbf{L}^{-T}\mathbf{z}) = (\mathbf{L}\mathbf{L}^T)^{-1} = \mathbf{Q}^{-1}$$

Algorithm 4 Sampling $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{Q}^{-1})$

```
1: Compute the Cholesky factorisation, $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$
2: Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
3: Solve $\mathbf{L}^T\mathbf{v} = \mathbf{z}$
4: Compute $\mathbf{x} = \boldsymbol{\mu} + \mathbf{v}$
5: Return $\mathbf{x}$
```

Evaluating the log-density

For Bayesian computations, we often have to compute the log-density for the normal distribution:

$$\log \pi(\mathbf{x}) = -\frac{n}{2}\log 2\pi + \frac{1}{2}\log\det(\mathbf{Q}) - \frac{1}{2}\underbrace{(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{Q}(\mathbf{x}-\boldsymbol{\mu})}_{=q}$$

If $\mathbf{x}$ is sampled, then $q = \mathbf{z}^T\mathbf{z}$ otherwise, compute this term as

- $\mathbf{u} = \mathbf{x} - \boldsymbol{\mu}$
- $\mathbf{v} = \mathbf{Q}\mathbf{u}$
- $q = \mathbf{u}^T\mathbf{v}$

$\log\det(\mathbf{Q})$ is also easy to evaluate: We get the determinant for free with the Cholesky decomposition:

$$\frac{1}{2}\log\det(\mathbf{Q}) = \frac{1}{2}\log\det(\mathbf{L}\mathbf{L}^\top) = \log\det(\mathbf{L}) = \sum_{i=1}^{n}\log L_{ii}$$

Conditional simulation of a GMRF

Decompose $\mathbf{x}$ as

$$\begin{pmatrix} \mathbf{x}_A \\ \mathbf{x}_B \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_B \end{pmatrix}, \begin{pmatrix} \mathbf{Q}_{AA} & \mathbf{Q}_{AB} \\ \mathbf{Q}_{BA} & \mathbf{Q}_{BB} \end{pmatrix}^{-1} \right)$$

Then

$$\mathbf{x}_A \mid \mathbf{x}_B$$

has canonical parameterisation

$$\mathbf{x}_A - \boldsymbol{\mu}_A \mid \mathbf{x}_B \sim \mathcal{N}_C(-\mathbf{Q}_{AB}(\mathbf{x}_B - \boldsymbol{\mu}_B), \mathbf{Q}_{AA}) \tag{3}$$

Simulate using an algorithm for a canonical parameterisation.

Sample $\mathbf{x} \sim \mathcal{N}_C(\mathbf{b}, \mathbf{Q}^{-1})$

Recall that

$$\text{“}\mathcal{N}_C(\mathbf{b}, \mathbf{Q}) \quad = \quad \mathcal{N}(\mathbf{Q}^{-1}\mathbf{b}, \mathbf{Q}^{-1})\text{”}$$

so we need to compute the mean as well.

Algorithm 5 Sampling $\mathbf{x} \sim \mathcal{N}_C(\mathbf{b}, \mathbf{Q})$

```
1: Compute the Cholesky factorisation, $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$
2: Solve $\mathbf{L}\mathbf{w} = \mathbf{b}$
3: Solve $\mathbf{L}^T\boldsymbol{\mu} = \mathbf{w}$
4: Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Solve $\mathbf{L}^T\mathbf{v} = \mathbf{z}$
6: Compute $\mathbf{x} = \boldsymbol{\mu} + \mathbf{v}$
7: Return $\mathbf{x}$
```

Posterior sampling and kriging

A common scenario is that we have a hierarchical model

$$\begin{aligned} \mathbf{y}|\mathbf{x} &\sim \mathsf{N}(\mathbf{A}\mathbf{x}, \mathbf{Q}_\varepsilon^{-1}) \\ \mathbf{x} &\sim \mathsf{N}(\boldsymbol{\mu}_x, \mathbf{Q}_x^{-1}) \end{aligned}$$

and are interested in sampling $\mathbf{x}|\mathbf{y} \sim \mathsf{N}(\boldsymbol{\mu}_{x|y}, \mathbf{Q}_{x|y}^{-1})$, with

$$\begin{aligned} \boldsymbol{\mu}_{x|y} &= \boldsymbol{\mu}_x + \mathbf{Q}_{x|y}^{-1}\mathbf{A}^\top\mathbf{Q}_\varepsilon(\mathbf{y} - \mathbf{A}\boldsymbol{\mu}_x) \\ \mathbf{Q}_{x|y} &= \mathbf{Q}_x + \mathbf{A}^\top\mathbf{Q}_\varepsilon\mathbf{A} \end{aligned}$$

Direct method for sampling: Use Algorithm 5 for
$\mathbf{x}|\mathbf{y} - \boldsymbol{\mu}_x \sim \mathsf{N}_C(\mathbf{A}^\top\mathbf{Q}_\varepsilon(\mathbf{y} - \mathbf{A}\boldsymbol{\mu}_x), \mathbf{Q}_{x|y})$.

This work well if we only have local observations.

Conditioning by kriging

A popular approach for posterior sampling for covariance-based hierarchical models is the *conditioning by kriging* approach.

The idea behind it is to sample the prior and then correct the sample so that it becomes a sample from the posterior.

The traditional form of the method is

- Sample $\mathbf{x}^* \sim \mathsf{N}(\boldsymbol{\mu}_x, \mathbf{Q}_x^{-1})$
- Sample $\mathbf{y}^* \sim \mathsf{N}(\mathbf{y}, \mathbf{Q}_\varepsilon^{-1})$
- Return $\mathbf{x} = \mathbf{x}^* - \mathbf{Q}_x^{-1}\mathbf{A}^\top(\mathbf{A}\mathbf{Q}_x^{-1}\mathbf{A}^\top + \mathbf{Q}_\varepsilon^{-1})^{-1}(\mathbf{A}\mathbf{x}^* - \mathbf{y}^*)$

The name comes from the fact that the final expression is the covariance-based kriging equation.

Conditioning by kriging II

$(\mathbf{A}\mathbf{Q}_x^{-1}\mathbf{A}^\top + \mathbf{Q}_\varepsilon^{-1})$ is almost always a dense matrix.

- We can only handle a small number of observations.

However, we can modify the expression using the *Woodbury identity* to recover the precision-based kriging expression:

$$\begin{aligned}\mathbf{x} &= \mathbf{x}^* - \mathbf{Q}_x^{-1}\mathbf{A}^\top(\mathbf{A}\mathbf{Q}_x^{-1}\mathbf{A}^\top + \mathbf{Q}_\varepsilon^{-1})^{-1}(\mathbf{A}\mathbf{x}^* - \mathbf{y}^*)\\ &= \mathbf{x}^* + \mathbf{Q}_{x|y}^{-1}\mathbf{A}\mathbf{Q}_\varepsilon(\mathbf{y}^* - \mathbf{A}\mathbf{x}^*)\end{aligned}$$

which is calculated using forward and backward solves.

A quick operation count shows that this almost always is less efficient than the direct method for GMRF models

The exception is when we have non-local observations.

Evaluating the conditional marginal data likelihood

To evaluate $\pi(\mathbf{y}|\boldsymbol{\theta})$, we can use that for any $\mathbf{x}^*$, we have

$$
\begin{aligned}
\pi(\mathbf{y}|\boldsymbol{\theta}) &= \frac{\pi(\boldsymbol{\theta}, \mathbf{y})}{\pi(\boldsymbol{\theta})} \\
&= \left.\frac{\pi(\boldsymbol{\theta}, \mathbf{y})\pi(\mathbf{x}|\boldsymbol{\theta}, \mathbf{y})}{\pi(\boldsymbol{\theta})\pi(\mathbf{x}|\boldsymbol{\theta}, \mathbf{y})}\right|_{\mathbf{x}=\mathbf{x}^*} \\
&= \left.\frac{\pi(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})}{\pi(\boldsymbol{\theta})\pi(\mathbf{x}|\boldsymbol{\theta}, \mathbf{y})}\right|_{\mathbf{x}=\mathbf{x}^*} \\
&= \left.\frac{\pi(\mathbf{X}|\boldsymbol{\theta})\pi(\mathbf{y}|\boldsymbol{\theta}, \mathbf{x})}{\pi(\mathbf{x}|\boldsymbol{\theta}, \mathbf{y})}\right|_{\mathbf{x}=\mathbf{x}^*}
\end{aligned}
$$

In practise, we use $\mathbf{x}^* = \boldsymbol{\mu}_{x|y}$ for numerical stability.

Evaluating the conditional marginal data likelihood

With $\mathbf{x}^* = \boldsymbol{\mu}_{x|y}$, we get

$$\begin{aligned}\log \pi(\mathbf{y}|\boldsymbol{\theta}) = &- \frac{n}{2}\log(2\pi) + \frac{1}{2}\log|\mathbf{Q}_x| + \frac{1}{2}\log|\mathbf{Q}_\varepsilon| - \frac{1}{2}\log|\mathbf{Q}_{x|y}| \\ &- \frac{1}{2}(\boldsymbol{\mu}_{x|y} - \boldsymbol{\mu}_x)^\top Q_x(\boldsymbol{\mu}_{x|y} - \boldsymbol{\mu}_x) \\ &- \frac{1}{2}(\mathbf{y} - \mathbf{A}\boldsymbol{\mu}_{x|y})^\top Q_\varepsilon(\mathbf{y} - \mathbf{A}\boldsymbol{\mu}_{x|y})\end{aligned}$$

Note that we do not need to evaluate $\mathbf{Q}_y$, which is dense.

By comparing terms, we can note that

1. $\log|\mathbf{Q}_y| = \log|\mathbf{Q}_x| + \log|\mathbf{Q}_\varepsilon| - \log|\mathbf{Q}_{x|y}|$
2. $(\mathbf{y} - \mathbf{A}\boldsymbol{\mu}_x)^\top \mathbf{Q}_y(\mathbf{y} - \mathbf{A}\boldsymbol{\mu}_x)$ is given by the sum of the quadratic forms above.

Linear constraints

Posterior sampling can be seen as sampling under linear contraints.

More generally, there are three basic classes of linear contraints that are important

1. Non-interacting hard constraints
   Certain nodes are given explicitly
2. Interacting hard contraints
   Linear combinations of nodes are constrained
3. Soft contraints
   Linear combinations are constrained under uncertainty

Linear constraints

All these cases can be written as

$$\mathbf{A}\mathbf{x} + \boldsymbol{\varepsilon} = \mathbf{e}$$

where

$$\boldsymbol{\varepsilon} = \begin{cases} 0 & \text{for hard contraints} \\ \mathsf{N}(\mathbf{0}, \mathbf{Q}_{\varepsilon}^{-1}) & \text{for soft contraints} \end{cases}$$

Aim: Sample from $\mathbf{x}|\mathbf{e}$ when

$$\mathbf{x} \sim \mathsf{N}(\boldsymbol{\mu}, \mathbf{Q}^{-1})$$
$$\mathbf{e}|\mathbf{x} \sim \mathsf{N}(\mathbf{A}\mathbf{x}, \mathbf{Q}_{\varepsilon}^{-1})$$

Soft constraints

Soft contraints are equivalent to the posterior sampling case.

Thus, in general we use the direct method for sampling.

The exception is if we have non-local observations, when it is better to use conditioning by kriging since the posterior precision is dense.

Example

Observe the sum of x_i with unit variance noise, the posterior precision is

$$\mathbf{Q} + \mathbf{1}\mathbf{1}^T$$

which is a dense matrix.

Algorithm 6 Sampling $\mathbf{x} \mid \mathbf{A}\mathbf{x} = \boldsymbol{\epsilon}$ when $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{e}, \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}})$ and $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{Q}^{-1})$

```
1: Compute the Cholesky factorisation, $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$
2: Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
3: Solve $\mathbf{L}^T\mathbf{v} = \mathbf{z}$
4: Compute $\mathbf{x} = \boldsymbol{\mu} + \mathbf{v}$
5: Compute $\mathbf{V}_{n\times k} = \mathbf{Q}^{-1}\mathbf{A}^T$ using Algorithm 3 using $\mathbf{L}$
6: Compute $\mathbf{W}_{k\times k} = \mathbf{A}\mathbf{V} + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$
7: Compute $\mathbf{U}_{k\times n} = \mathbf{W}^{-1}\mathbf{V}^T$ using Algorithm 3
8: Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{e}, \mathbf{W})$ using the factorisation from step 7
9: Compute $\mathbf{c} = \mathbf{A}\mathbf{x} - \boldsymbol{\epsilon}$
10: Compute $\mathbf{x}^* = \mathbf{x} - \mathbf{U}^T\mathbf{c}$
11: **Return** $\mathbf{x}^*$
```

When $\mathbf{z} = \mathbf{0}$ and $\boldsymbol{\epsilon} = \mathbf{e}$ then $\mathbf{x}^*$ is the conditional mean.

This is computationally feasible if $k \ll n$.

Evaluating the log-density

With only local observations, we evaluate the log-density of the posterior as we evaluated the prior log-denstiy.

For non-local observations, the log-density can be computed via

$$\pi(\mathbf{x} \mid \mathbf{e}) = \frac{\pi(\mathbf{x})\pi(\mathbf{e} \mid \mathbf{x})}{\pi(\mathbf{e})} \tag{4}$$

All required Cholesky factors are available from the simulation algorithm.

$\pi(\mathbf{x})$-term. This is a GMRF.

$\pi(\mathbf{e} \mid \mathbf{x})$-term. $\mathbf{e} \mid \mathbf{x}$ is Gaussian with mean $\mathbf{A}\mathbf{x}$ and covariance $\mathbf{\Sigma}_\epsilon$.

$\pi(\mathbf{e})$-term. $\mathbf{e}$ is Gaussian with mean $\mathbf{A}\boldsymbol{\mu}$ and covariance matrix $\mathbf{A}\mathbf{Q}^{-1}\mathbf{A}^T + \mathbf{\Sigma}_\epsilon$.

Non-interacting hard contraints

This is the simplest case, we split the joint distribution for $\mathbf{x}$ in non-constrained nodes $\mathbf{x}_n$ and constrained nodes $\mathbf{x}_c$:

$$\begin{pmatrix} \mathbf{x}_n \\ \mathbf{x}_c \end{pmatrix} \sim \mathsf{N}\left(\begin{pmatrix} \boldsymbol{\mu}_n \\ \boldsymbol{\mu}_c \end{pmatrix}, \begin{pmatrix} \mathbf{Q}_{nn} & \mathbf{Q}_{nc} \\ \mathbf{Q}_{cn} & \mathbf{Q}_{cc} \end{pmatrix}^{-1} \right).$$

By Theorem 2.5 we have $\mathbf{x}_n | \mathbf{x}_c = \mathbf{e} \sim \mathsf{N}(\boldsymbol{\mu}_{n|c}, \mathbf{Q}_{cc}^{-1})$, where

$$\boldsymbol{\mu}_{n|c} = \boldsymbol{\mu}_n - \mathbf{Q}_{nn}^{-1}\mathbf{Q}_{nc}(\mathbf{e} - \boldsymbol{\mu}_c)$$

Thus, we can sample the constrained distribution using Algorithm 6.

Note that the joint constrained distribution is degenerate:

$$\left(\begin{pmatrix} \mathbf{x}_n \\ \mathbf{x}_c \end{pmatrix} \mid \mathbf{x_c} = \mathbf{e} \right) \sim \mathsf{N}\left(\begin{pmatrix} \boldsymbol{\mu}_{n|c} \\ \mathbf{e} \end{pmatrix}, \begin{pmatrix} \mathbf{Q}_{nn}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \right).$$

Interacting hard contraints $\mathbf{x}|\mathbf{Ax} = \mathbf{e}$

This case occurs quite frequently:

> *A sum-to-zero constraint corresponds to* $k = 1$, $\mathbf{A} = \mathbf{1}^T$ *and* $\mathbf{e} = 0$.

The linear constraint makes the conditional distribution Gaussian

- but it is singular as the rank of the constrained covariance matrix is $n - k$
- more care must be exercised during sampling

We have that

$$\mathsf{E}(\mathbf{x} \mid \mathbf{Ax} = \mathbf{e}) = \boldsymbol{\mu} - \mathbf{A}\mathbf{Q}^{-1}(\mathbf{A}\mathbf{Q}^{-1}\mathbf{A}^T)^{-1}(\mathbf{A}\boldsymbol{\mu} - \mathbf{e}) \quad (5)$$

$$\mathrm{Cov}(\mathbf{x} \mid \mathbf{Ax} = \mathbf{e}) = \mathbf{Q}^{-1} - \mathbf{Q}^{-1}\mathbf{A}^T(\mathbf{A}\mathbf{Q}^{-1}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{Q}^{-1} \quad (6)$$

This is typically a dense-matrix case, which must be solved using general $\mathcal{O}(n^3)$ algorithms.

Conditioning via Kriging

However, this is where we can use conditioning by kriging and *correct* for the constraints, at nearly no costs if $k \ll n$.

With $\mathbf{Q}_{\varepsilon}^{-1} = \mathbf{0}$ in the earlier expression, we get:

Let $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{Q})$, then compute

$$\mathbf{x}^* = \mathbf{x} - \mathbf{Q}^{-1}\mathbf{A}^T(\mathbf{A}\mathbf{Q}^{-1}\mathbf{A}^T)^{-1}(\mathbf{A}\mathbf{x} - \mathbf{e}). \qquad (7)$$

Now $\mathbf{x}^*$ has the correct conditional distribution!

$\mathbf{A}\mathbf{Q}^{-1}\mathbf{A}^T$ is a $k \times k$ matrix, hence its factorisation is fast to compute for small k.

Algorithm 7 Sampling $\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{e}$ when $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{Q}^{-1})$

```
1: Compute the Cholesky factorisation, $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$
2: Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
3: Solve $\mathbf{L}^T\mathbf{v} = \mathbf{z}$
4: Compute $\mathbf{x} = \boldsymbol{\mu} + \mathbf{v}$
5: Compute $\mathbf{V}_{n\times k} = \mathbf{Q}^{-1}\mathbf{A}^T$ with Algorithm 3
6: Compute $\mathbf{W}_{k\times k} = \mathbf{A}\mathbf{V}$
7: Compute $\mathbf{U}_{k\times n} = \mathbf{W}^{-1}\mathbf{V}^T$ using Algorithm 3
8: Compute $\mathbf{c} = \mathbf{A}\mathbf{x} - \mathbf{e}$
9: Compute $\mathbf{x}^* = \mathbf{x} - \mathbf{U}^T\mathbf{c}$
10: **Return** $\mathbf{x}^*$
```

If $\mathbf{z} = \mathbf{0}$ in Algorithm 7, then $\mathbf{x}^*$ is the conditional mean.
Extra cost is only $\mathcal{O}(k^3)$ for large k!

Evaluating the log-density

The log-density can be rapidly computed using the following identity,

$$\pi(\mathbf{x} \mid \mathbf{A}\mathbf{x}) = \frac{\pi(\mathbf{x})\pi(\mathbf{A}\mathbf{x} \mid \mathbf{x})}{\pi(\mathbf{A}\mathbf{x})}, \tag{8}$$

We can compute each term on the rhs easier than the lhs.

$\pi(\mathbf{x})$-term. This is a GMRF and the log-density is easy to compute using $\mathbf{L}$ computed in Algorithm 7 step 1.

Evaluating the log-density

The log-density can be rapidly computed using the following identity,

$$\pi(\mathbf{x} \mid \mathbf{A}\mathbf{x}) = \frac{\pi(\mathbf{x})\pi(\mathbf{A}\mathbf{x} \mid \mathbf{x})}{\pi(\mathbf{A}\mathbf{x})}, \tag{8}$$

We can compute each term on the rhs easier than the lhs.

$\pi(\mathbf{A}\mathbf{x} \mid \mathbf{x})$-term. This is a degenerate density, which is either zero or a constant,

$$\log \pi(\mathbf{A}\mathbf{x} \mid \mathbf{x}) = -\frac{1}{2} \log |\mathbf{A}\mathbf{A}^T| \tag{9}$$

The determinant of a $k \times k$ matrix can be found its Cholesky factorisation.

Evaluating the log-density

The log-density can be rapidly computed using the following identity,

$$\pi(\mathbf{x} \mid \mathbf{Ax}) = \frac{\pi(\mathbf{x})\pi(\mathbf{Ax} \mid \mathbf{x})}{\pi(\mathbf{Ax})}, \tag{9}$$

We can compute each term on the rhs easier than the lhs.

$\pi(\mathbf{Ax})$-term. $\mathbf{Ax}$ is Gaussian with mean $\mathbf{A}\boldsymbol{\mu}$ and covariance matrix $\mathbf{AQ}^{-1}\mathbf{A}^T$ with Cholesky triangle $\widetilde{\mathbf{L}}$ available from Algorithm 7 step 7.

Numerical methods for sparse matrices

We see that computations for GMRFs can be expressed such that the main tasks are

1. compute the Cholesky factorisation of $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$, and
2. solve $\mathbf{L}\mathbf{v} = \mathbf{b}$ and $\mathbf{L}^T\mathbf{x} = \mathbf{z}$.

Now we will see how sparsity can be used to do these tasks efficiently.

The second task is much faster than the first, but sparsity will be of advantage also here.

The goals

The goal with this lecture and the next is to explain

- **why** a sparse $\mathbf{Q}$ allow for fast factorisation
- **how** we can take advantage of it,
- **why** we gain if we permute the vertics before factorising the matrix,
- **how** statisticians can benefit for recent research in this area by the numerical mathematicians.

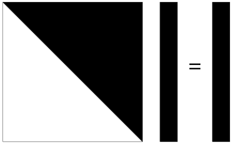
At the end we present a small case study factorising some typical matrices for GMRFs, using a classical and more recent methods for factorising matrices.

Interpretation of $\mathbf{L}$ (I)

Let $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$, then the solution of

$$\mathbf{L}^T\mathbf{x} = \mathbf{z} \quad \text{where} \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

is $\mathcal{N}(\mathbf{0}, \mathbf{Q}^{-1})$ distributed.



Since $\mathbf{L}$ is lower triangular then

$$x_n = \frac{1}{L_{nn}} z_n$$
$$x_{n-1} = \frac{1}{L_{n-1,n-1}} \left(z_{n-1} - L_{n,n-1}x_n\right)$$
$$\dots$$

Interpretation of $\mathbf{L}$ (II)

Theorem

Let $\mathbf{x}$ be a GMRF wrt to the labelled graph $\mathcal{G}$, with mean $\boldsymbol{\mu}$ and precision matrix $\mathbf{Q} > 0$. Let $\mathbf{L}$ be the Cholesky triangle of $\mathbf{Q}$. Then for $i \in \mathcal{V}$,

$$\begin{aligned} \mathsf{E}(x_i \mid \mathbf{x}_{(i+1):n}) &= \mu_i - \frac{1}{L_{ii}} \sum_{j=i+1}^{n} L_{ji}(x_j - \mu_j) \qquad \textit{and} \\ \mathrm{Prec}(x_i \mid \mathbf{x}_{(i+1):n}) &= L_{ii}^2. \end{aligned}$$

Determine the zero-pattern in $\mathbf{L}$ (I)

Theorem

Let $\mathbf{x}$ be a GMRF wrt $\mathcal{G}$, with mean $\boldsymbol{\mu}$ and precision matrix $\mathbf{Q} > 0$. Let $\mathbf{L}$ be the Cholesky triangle of $\mathbf{Q}$ and define for $1 \leq i < j \leq n$ the set

$$F(i,j) = \{i+1, \ldots, j-1, j+1, \ldots, n\},$$

which is the future of i except j. Then

$$x_i \perp x_j \mid \mathbf{x}_{F(i,j)} \iff L_{ji} = 0.$$

If we can verify that L_{ji} is zero, we do not have to compute it when factorising $\mathbf{Q}$

Determine the zero-pattern in $\mathbf{L}$ (II)

The global Markov property provide a simple and sufficient criteria for checking if $L_{ji} = 0$.

Corollary

If $F(i,j)$ separates $i < j$ in $\mathcal{G}$, then $L_{ji} = 0$.

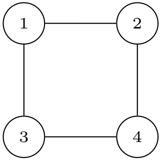
Corollary

If $i \sim j$ then $F(i,j)$ does not separate $i < j$.

The idea is simple

- Use the *global Markov property* to check if $L_{ji} = 0$.
- Compute only the non-zero terms in $\mathbf{L}$, so that $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$.
- Note that the corollary does not use numerical values of $\mathbf{Q}$, thus it is true for any $\mathbf{Q} > 0$ with the same graph.

Example



$$\mathbf{Q} = \begin{pmatrix} \times & \times & \times & \\ \times & \times & & \times \\ \times & & \times & \times \\ & \times & \times & \times \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} \times & & & \\ \times & \times & & \\ \times & ? & \times & \\ ? & \times & \times & \times \end{pmatrix} \qquad \mathbf{L} = \begin{pmatrix} \times & & & \\ \times & \times & & \\ \times & \surd & \times & \\ & \times & \times & \times \end{pmatrix}$$

Example: AR(1)-process

$$x_t \mid \mathbf{x}_{1:(t-1)} \sim \mathcal{N}(\phi x_{t-1}, \sigma^2), \quad t = 1, \ldots, n$$

$$\mathbf{Q} = \begin{pmatrix} \times & \times & & & & & \\ \times & \times & \times & & & & \\ & \times & \times & \times & & & \\ & & \times & \times & \times & & \\ & & & \times & \times & \times & \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{pmatrix} \mathbf{L} = \begin{pmatrix} \times & & & & & & \\ \times & \times & & & & & \\ & \times & \times & & & & \\ & & \times & \times & & & \\ & & & \times & \times & & \\ & & & & \times & \times & \\ & & & & & \times & \times \end{pmatrix}$$

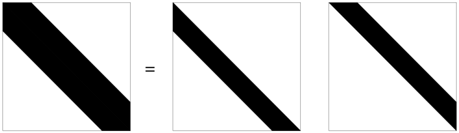
Bandwidth is preserved

Similarly, for an AR(p)-process

- $\mathbf{Q}$ have bandwidth p.
- $\mathbf{L}$ have lower-bandwidth p.

Theorem

Let $\mathbf{Q} > 0$ be a band matrix with bandwidth p and dimension n, then the Cholesky triangle of $\mathbf{Q}$ has (lower) bandwidth p.



...easy to modify existing Cholesky-factorisation code to use only entries where $|i - j| \leq p$.

Avoid computing L_{ij} and reading Q_{ij} for $|i-j| > p$.

Algorithm 8 Band-Cholesky factorization of $\mathbf{Q}$ with bandwidth p

```
 1: for $j = 1$ to $n$ do
 2:     $\lambda = \min\{j+p, n\}$
 3:     $v_{j:\lambda} = Q_{j:\lambda,j}$
 4:     for $k = \max\{1, j-p\}$ to $j-1$ do
 5:         $i = \min\{k+p, n\}$
 6:         $v_{j:i} = v_{j:i} - L_{j:i,k}L_{jk}$
 7:     end for
 8:     $L_{j:\lambda,j} = v_{j:\lambda}/\sqrt{v_j}$
 9: end for
10: Return $\mathbf{L}$
```

Cost is now $n(p^2+3p)$ flops assuming $n \gg p$.