Lecture 4: Numerical methods for sparse matrices

Gaussian Markov random fields

David Bolin
Chalmers University of Technology
January 27, 2015

The goals

Yesterday, we showed 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 $\mathbf{L}^T\mathbf{x} = \mathbf{z}$.

We also discussed

- **why** a sparse $\mathbf{Q}$ allow for fast factorisation
- **how** we can take advantage of it,

The goal today is to show

- **why** we gain if we permute the vertics before factorising the matrix,
- **how** we can use sparsity properties to compute marginal variances
- a small case study factorising some typical matrices for GMRFs

How we use the Markov property

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*

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

- If we can verify that L_{ji} is zero, we do not have to compute it when factorising $\mathbf{Q}$
- Use the *global Markov property* to check if $L_{ji} = 0$.
- Compute only the non-zero terms in $\mathbf{L}$, so that $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$.

Band matrices

The profile (or envelope) of a symmetric matrix determines how close its non-zero elements are to the diagonal:

$$\textit{Profile}(\mathbf{Q}) = \sum_{i=1}^{n} (i - \min(\mathrm{ne}(i)))$$

The bandwidth is the largest deviation:

$$\textit{Bandwidth}(\mathbf{Q}) = \max(i - \min(\mathrm{ne}(i)))$$

Recall that the cost for a band cholesky factorisation with bandwidth p is $n(p^2 + 3p)$ flops assuming $n \gg p$.

Reorder to reduce the bandwidth

We rarely have band matrices, but we can permute the vertexes;

> *select one of the $n!$ possible permutations, define the corresponding permutation matrix $\mathbf{P}$, such that $\mathbf{i}^P = \mathbf{P}\mathbf{i}$, where $\mathbf{i} = (1, \ldots, n)^T$, is the new ordering of the vertexes.*

Chose $\mathbf{P}$, if possible, such that

$$\mathbf{Q}^P = \mathbf{P}\mathbf{Q}\mathbf{P}^T \tag{1}$$

is a band-matrix with a small bandwidth.

Solve $\mathbf{Q}\boldsymbol{\mu} = \mathbf{b}$ as follows:

- $\mathbf{b}^P = \mathbf{P}\mathbf{b}$.
- Solve $\mathbf{Q}^P\boldsymbol{\mu}^P = \mathbf{b}^P$
- Map the solution back, $\boldsymbol{\mu} = \mathbf{P}^T\boldsymbol{\mu}^P$.

Some more concepts from graph theory

Graph bandwidth

For a numbering f of the nodes, we define the bandwidth of $\mathcal{G}$ relative to f by

$$\beta_f(\mathcal{G}) = \max\left(|f(v_1) - f(v_2)| : \{v_1, v_2\} \in \mathcal{E}\right)$$

The minimum of $\beta_f(\mathcal{G})$ over all numberings of $\mathcal{G}$ is called the bandwidth of $\mathcal{G}$, denoted by $\beta(\mathcal{G})$.

Level structures

A level structure, $L(\mathcal{G})$ of a graph is a partition of the nodes into levels $L_1, \ldots, L_k$, such that

1. $\mathrm{ne}(L_1) \subset L_1 \cup L_2$
2. $\mathrm{ne}(L_i) \subset L_{i-1} \cup L_i \cup L_{i+1}$ for $1 < i < k$
3. $\mathrm{ne}(L_k) \subset L_{k-1} \cup L_k$

Rooted level structures

Definition

To each $v \in \mathcal{V}$, there is an associated level structure $L_v(\mathcal{G})$, called the level structure rooted at v, defined by

1. $L_1 = \{v\}$
2. for $i > 1$, $L_i = \mathrm{ne}(L_{i-1}) \setminus \{L_1 \cup \cdots \cup L_{i-1}\}$

For a given level structure L,

- $w_i(L) = |L_i|$ is called the width of level i
- $w(L) = \max |L_i|$ is called the width of the level structure.

Let f_L be the ordering obtained by assigning consecutive integers level by level to the nodes in a level structure, we then have

1. $\beta_{f_L} \leq 2w(L) - 1$
2. If L is rooted, we also have $w(L) \leq \beta_{f_L}$

Cuthill-McKee ordering

Finding the ordering that minimizes the bandwidth is NP-hard, so we have to use heuristics.

The simplest bandwidth reduction method is the Cuthill-Mckee algorithm which goes as follows:

The Cuthill-McKee method

Choose a starting node v and generate $L_v(\mathcal{G})$. Assign v the number 1, then for each successive level beginning with L_2:

1. Number the nodes adjacent to the lowest numbered node of the preceding level, in order of increasing degree.
2. Number the remaining vertices adjacent to the next lowest numbered vertex of the preceding level.
3. Continue the process until all vertices of the current level are numbered, then begin again on the next level.

Bandwidth reduction reorderings (II)

A good choice of the starting node is vital for the method.

One method is to try all nodes of low degree, compute the ordering, and select the ordering with the lowest resulting bandwidth.

Another common way is to choose v as a *pseudoperpheral node*, which is found as follows:

1. Select an arbitrary starting node
2. Repeat: Select the most distant node to the previously selected node (found using a breath-first search).
3. Stop when the distance is not increased.

The profile may be further reduced by reversing the ordering. It can be shown that this cannot increase the profile (and it has no effect on the bandwidth)

`symrcm` in Matlab implements the reversed Cuthill-McKee algorithm, with a pseudoperpheral node as starting value.

Bandwidth reduction reorderings (III)

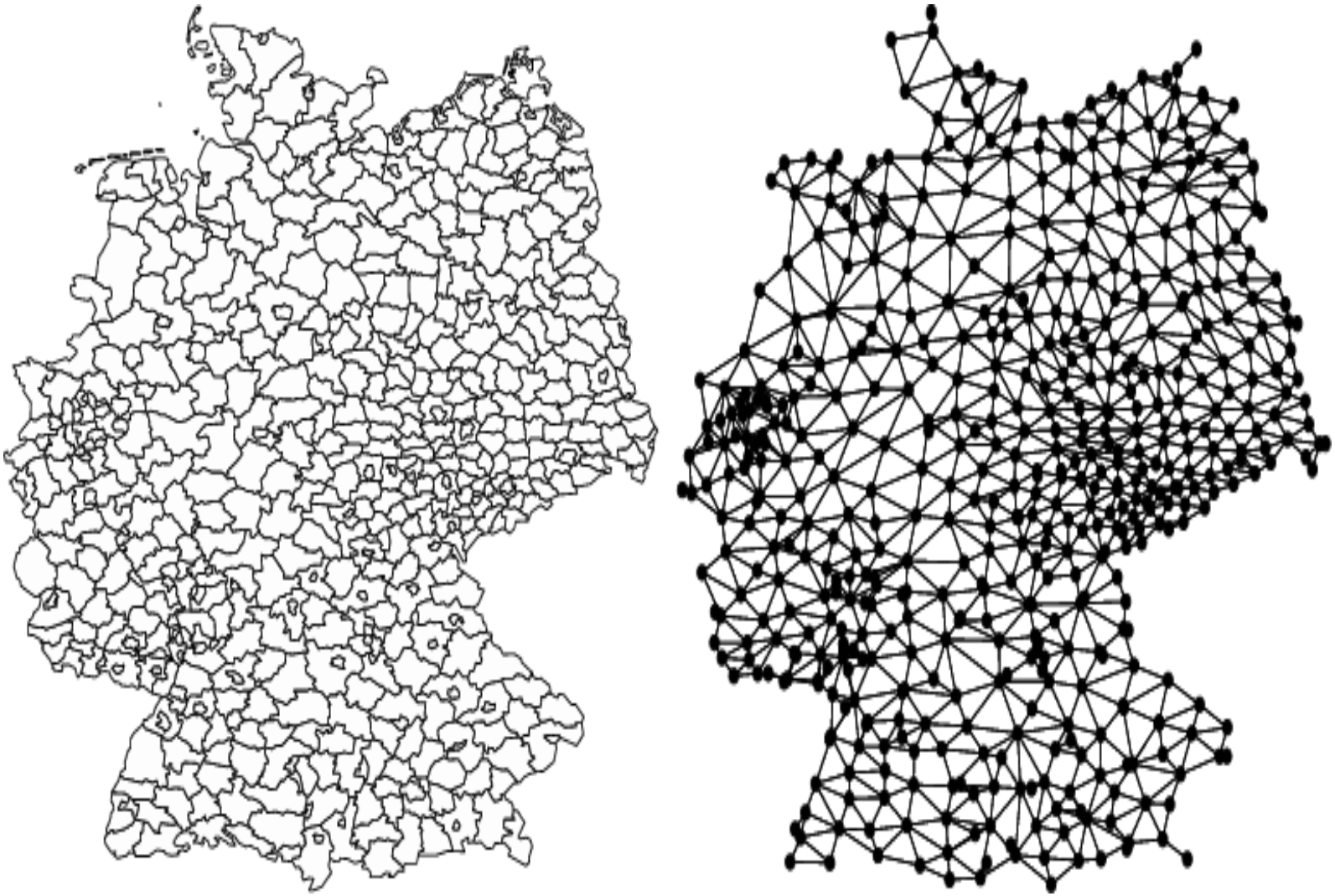
The Cuthill-McKee algorithm is simple, but has several disadvantages.

One important disadvantage is that because L is rooted, we have $w(L) \leq \beta_{F_L}$, even though $\beta(\mathcal{G})$ can be much smaller than the width of any rooted level structure.

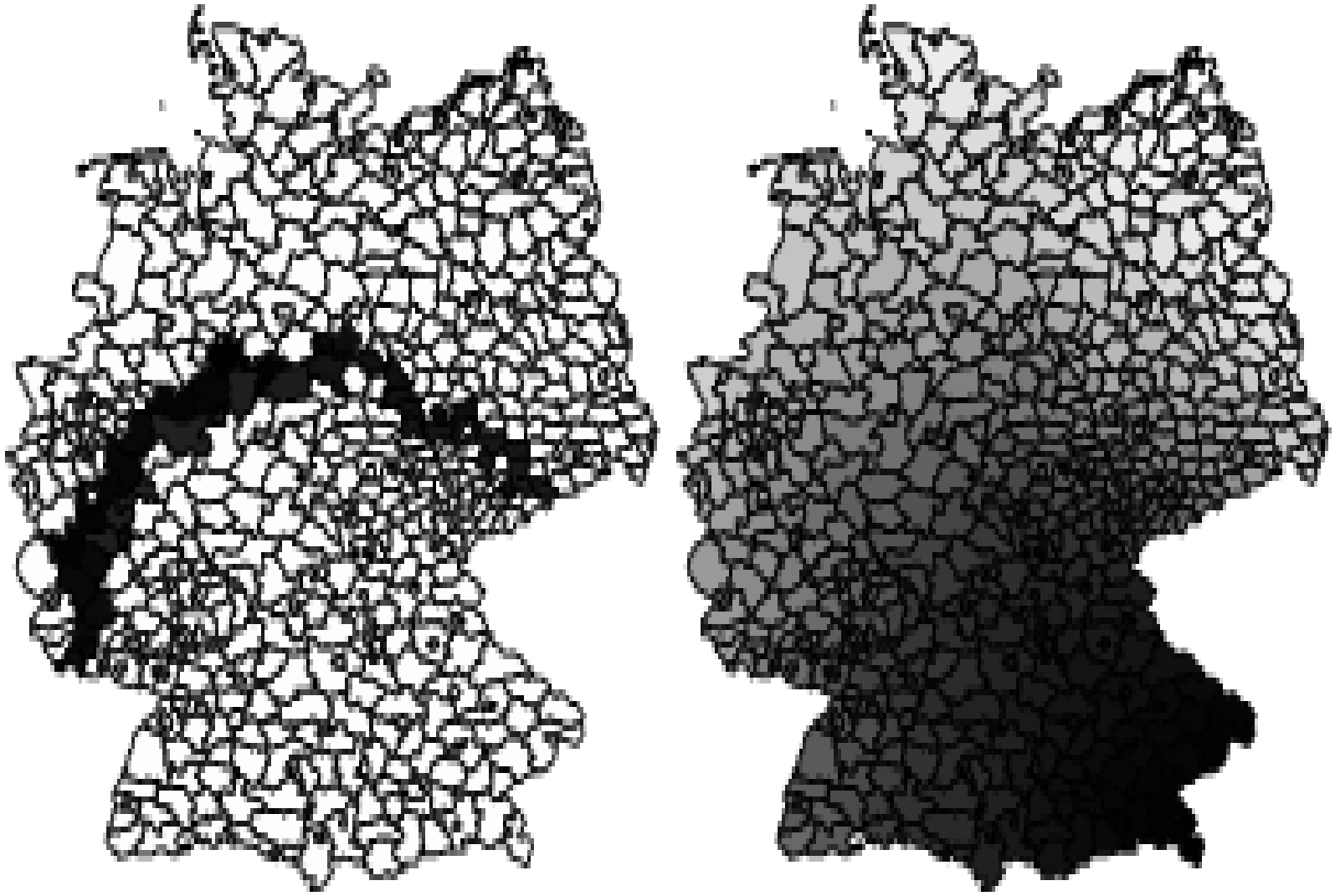
The Gibbs-Poole-Stockmeyer and Gibbs-King algorithm is

1. an alternative, often superior, profile reduction method
2. based on finding more general level structures chosen to minimize the level width.
3. Slightly more complex, see references on the homepage for details.

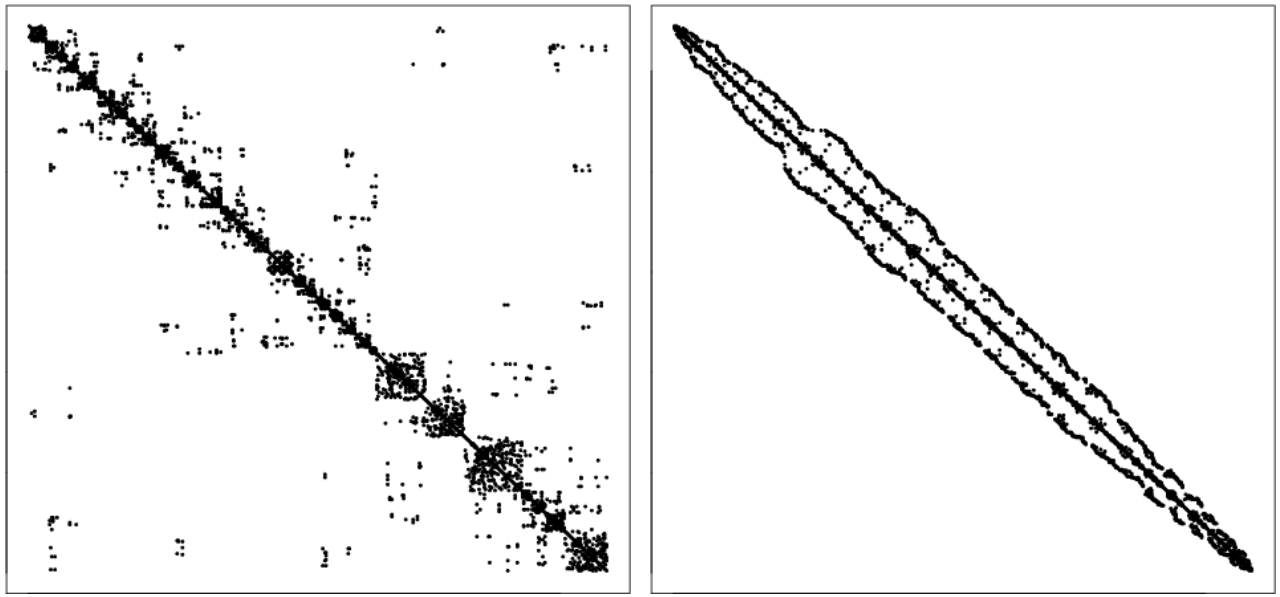
Example (I)



Example (II)



Example (III)



- Obtained using the GPSKCA algorithm
- The bandwidth is in this case 43, which is about $\mathcal{O}(\sqrt{n})$, so this cost is $\mathcal{O}(n^2)$ for this graph
- The fill-in ratio is 5.3, so we could do even better!

Fill-in reducing orderings

The purpose of a reordering is to reduce the fill-ins: $n_L - n_Q$

- n_L is the number of non-zero elements in $\mathbf{L}$
- n_Q is the number of non-zero elements in the lower triangular part of $\mathbf{Q}$

n_L/n_Q is the fill-in ratio

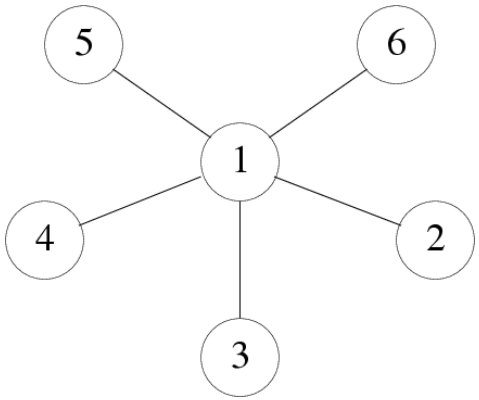
The bandwidth reduction methods use a simple heuristics, but there may be other better methods.

Finding the optimal ordering is NP-hard, so a sub-optimal ordering will have to do

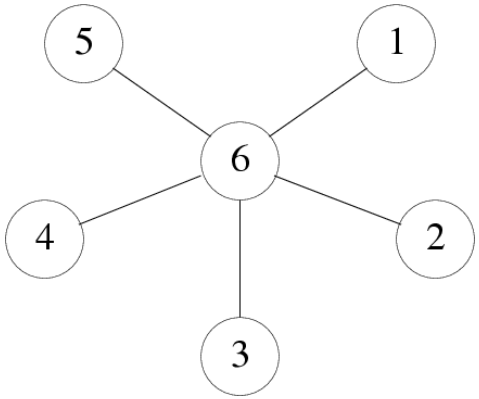
The common fill-reducing orderings can be divided into three main categories

1. bandwidth (or profile) reduction
2. Minimum degree
3. nested dissection

More optimal reordering schemes



$$\begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ \times & & \times & & & \\ \times & & & \times & & \\ \times & & & & \times & \\ \times & & & & & \times \end{pmatrix} \quad \begin{pmatrix} \times & & & & & \\ \times & \times & & & & \\ \times & \surd & \times & & & \\ \times & \surd & \surd & \times & & \\ \times & \surd & \surd & \surd & \times & \\ \times & \surd & \surd & \surd & \surd & \times \end{pmatrix}$$



$$\begin{pmatrix} \times & & & & & \times \\ & \times & & & & \times \\ & & \times & & & \times \\ & & & \times & & \times \\ & & & & \times & \times \\ \times & \times & \times & \times & \times & \times \end{pmatrix} \quad \begin{pmatrix} \times & & & & & \\ & \times & & & & \\ & & \times & & & \\ & & & \times & & \\ & & & & \times & \\ \times & \times & \times & \times & \times & \times \end{pmatrix}$$

Nested dissection reordering (I)

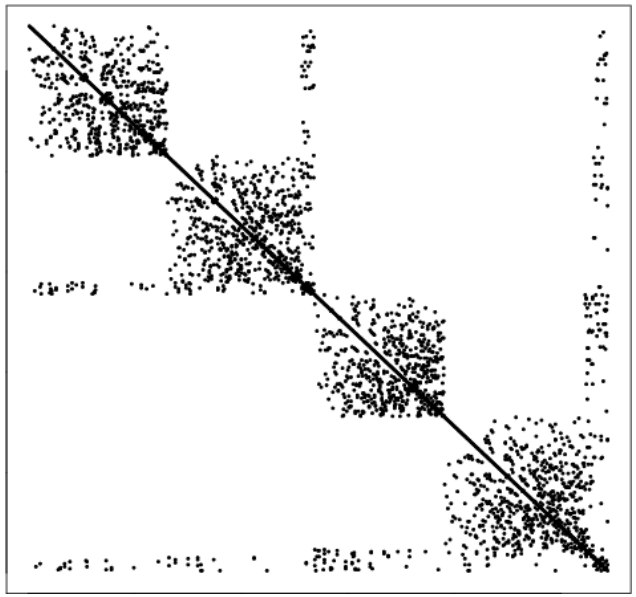
The idea generalise as follows.

- Select a (small) set of nodes whose removal divides the graph into two disconnected subgraphs of almost equal size.
- Order the nodes chosen *after* ordering all the nodes in both subgraphs.
- Apply this procedure recursively to the nodes in each subgraph.

Costs in the spatial case

- Factorisation $\mathcal{O}(n^{3/2})$
- Fill-in $\mathcal{O}(n \log n)$
- Optimal in the order sense.

Nested dissection reordering (II)



Minimum degree orderings

Consider a right-looking sparse Cholesky factorisation:

1: **for** $k = 1$ **to** n **do**
2: $\quad L_{k,k} = \sqrt{Q_{k,k}}$
3: $\quad \mathbf{L}_{k+1:n,k} = \mathbf{Q}_{k+1:n,k} / L_{k,k}$
4: $\quad \mathbf{Q}_{k+1:n,k+1:n} = \mathbf{Q}_{k+1:n,k+1:n} + \mathbf{L}_{k+1:n,k} \mathbf{L}_{k+1:n,k}^{\top}$
5: **end for**

- Let $\mathbf{Q}^{[k]}$ denote the submatrix $\mathbf{Q}_{k:n,k:n}$ at the beginning of iteration k.
- Let $d_i, i > k$ be the number of neighbors (the degree) to node i in in the graph of $\mathbf{Q}^{[k]}$.
- Rather than choosing node k in the k:th step, the minimum degree selects the node with the smallest degree

Minimum degree orderings (II)

- When a node is selected, the degree of all nodes neighboring that node must be recomputed. This is the costly part.
- The approximate minimum degree algorithm (`AMD` in Matlab) uses an approximation of the degree to reduce the computational cost.
- Further cost reductions can be obtained by
  1. The use of *supernodes*
     - Obtained by merging indistinguishable nodes
     - when one of them has minimal degree, the other also has minimal degree
     - Eliminating both causes no more edges in the graph than eliminating just one
  2. *mass elimination*, where at step k if a node i only has one edge to k, it can be eliminated directly
- See references on homepage for details

Statisticians and numerical methods for sparse matrices

A Key lesson

If you have a sparse matrix you should Always reorder your nodes

So what ordering should we use?

- Minimum degree and its variants are most common and have good performance on the widest range of matrices
- Good nested dissection methods tend to be superior to minimum degree orderings for large problems that come from 2D and 3D spatial discretizations (such as FEM methods)

Implementing these methods require substantial knowledge of numerical methods, data structures, and high-performance computing

There exists many excellent sparse solvers which we can use!

Computing marginal variances for GMRFs

We often need the marginal variances of a GMRF

- For example when computing kriging standard errors

Let

$$\mathbf{Q} = \mathbf{V}\mathbf{D}\mathbf{V}^T$$

- where $\mathbf{D}$ is a diagonal matrix, and
- $\mathbf{V}$ is a lower triangular matrix with ones on the diagonal.

The matrix identity

$$\mathbf{\Sigma} = \mathbf{D}^{-1}\mathbf{V}^{-1} + (\mathbf{I} - \mathbf{V}^T)\mathbf{\Sigma}$$

define *recursions* which can be used to compute

- $\mathrm{Var}(x_i)$ and $\mathrm{Cov}(x_i, x_j)$ for $i \sim j$

essentially without cost when the Cholesky triangle $\mathbf{L}$ is known.

Statistical derivation

Recall for a zero mean GMRF that

$$x_i \mid x_{i+1}, \ldots, x_n \sim \mathcal{N}(-\frac{1}{L_{ii}} \sum_{k=i+1}^{n} L_{ki} x_k, 1/L_{ii}^2), \quad i = n, \ldots, 1.$$

provides a sequential representation of the GMRF

Multiply by x_j, $j \geq i$, and taking expectation yields

$$\Sigma_{ij} = \delta_{ij}/L_{ii}^2 - \frac{1}{L_{ii}} \sum_{k \in \mathcal{I}(i)}^{n} L_{ki} \Sigma_{kj}, \quad j \geq i, \; i = n, \ldots, 1,$$

where $\mathcal{I}(i)$ as those k where L_{ki} is non-zero,

$$\mathcal{I}(i) = \{k > i \; : \; L_{ki} \neq 0\}$$

and δ_{ij} is one if $i = j$ and zero otherwise.

The method

General algorithm

for $i = n, \ldots, 1$

for decreasing j *in* $\mathcal{I}(i)$

Compute $\Sigma_{i,j}$ *from*

$$\Sigma_{ij} = \delta_{ij}/L_{ii}^2 - \frac{1}{L_{ii}} \sum_{k \in \mathcal{I}(i)}^{n} L_{ki}\Sigma_{kj}$$

Equivalent to Kalman-recursions for smoothing.

Computational costs:

Time	$\mathcal{O}(n)$
Spatial	$\mathcal{O}(n \log(n)^2)$
Spatio-temporal	$\mathcal{O}(n^{5/3})$.

Which elements do we need to compute?

- Assume we want to compute all marginal variances.
- To do so, we need to compute Σ_{ij} (or Σ_{ji}) for all ij in some set $\mathcal{S}$.
- If the recursions can be solved by only computing Σ_{ij} for all $ij \in \mathcal{S}$ we say that the recursions are *solvable* using $\mathcal{S}$.

From

$$\Sigma_{ij} = \delta_{ij}/L_{ii}^2 - \frac{1}{L_{ii}} \sum_{k \in \mathcal{I}(i)}^{n} L_{ki}\Sigma_{kj}, \quad j \geq i,\ i = n, \ldots, 1, \tag{2}$$

it is evident that $\mathcal{S}$ must satisfy

$$ij \in \mathcal{S} \text{ and } k \in \mathcal{I}(i) \implies kj \in \mathcal{S} \tag{3}$$

We also need that $ii \in \mathcal{S}$ for $i = 1, \ldots, n$.

The result

- $\mathcal{S} = \mathcal{V} \times \mathcal{V}$ is a valid set, but we want $|\mathcal{S}|$ to be minimal to avoid unnecessary computations.
- Such a minimal set depends however on the numerical values in $\mathbf{L}$, or $\mathbf{Q}$ implicitly.
- Denote by $\mathcal{S}(\mathbf{Q})$ a minimal set.

Theorem

The union of $\mathcal{S}(\mathbf{Q})$ for all $\mathbf{Q} > 0$ with fixed graph $\mathcal{G}$, is a subset of

$$\mathcal{S}^* = \{ij \in \mathcal{V} \times \mathcal{V} \ : \ j \geq i, \ i \text{ and } j \text{ are not separated by } F(i,j)\}$$

and $\mathcal{S}^*$ is solvable.

Interpretation of $\mathcal{S}^*$

- $\mathcal{S}^*$ is the set of all possible non-zero elements in $\mathbf{L}$ based on $\mathcal{G}$ only.
- This is the set of L_{ji}'s that are computed when computing $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$.
- Since $L_{ji} \neq 0$ in general when $i \sim j$, then we compute also $\mathrm{Cov}(x_i, x_j)$ for $i \sim j$.
- Some of the L_{ij}'s might turn out to be zero depending on the conditional independence properties of the marginal density for $\mathbf{x}_{i:n}$ for $i = n, \ldots, 1$. This might cause a slight problem in practice (implementation dependent).

A numerical case-study of typical GMRFs

Divide GMRF models into three categories.

1. GMRF models in time or on a line; auto-regressive models and models for smooth functions.
   Neighbours to x_t are then those $\{x_s\}$ such that $|s-t| \leq p$.
2. Spatial GMRF models; regular lattice, or irregular lattice induced by a tessellation or regions of land.
   Neighbours to x_i (spatial index i), are those j spatially "close" to i, where "close" is defined from its context.
3. Spatio-temporal GMRF models. Often an extension of spatial models to also include dynamic changes.

Global nodes

Also include some "global" nodes:

Let $\mathbf{x}$ be a GMRF with a common mean μ

$$\mathbf{x}|\mu \sim \mathcal{N}(\mu\mathbf{1}, \mathbf{Q}^{-1}) \tag{4}$$

Assume $\mu \sim \mathcal{N}(0, \sigma^2)$

...then $(\mathbf{x}, \mu)$ is also a GMRF where the node μ is neighbour with all x_i's.

The methods investigated

Two different algorithms

1. The band-Cholesky factorisation (BCF).
   Here we use the LAPACK-routines `DPBTRF` and `DTBSV`, for the factorisation and the forward/back-substitution, respectively, and the Gibbs-Poole-Stockmeyer algorithm for bandwidth reduction.
2. The Multifrontal Supernodal Cholesky factorisation (MSCF) implementation in the library `TAUCS` using the nested dissection reordering from the library `METIS`.

These methods are available in the C library `GMRFLib`...

which forms the basis for the `INLA` library...

which forms the basis for the `R-INLA` package...

which we will look at later.

The tasks of interest

The tasks we want to investigate are

1. factorising $\mathbf{Q}$ into $\mathbf{L}\mathbf{L}^T$, and
2. solving $\mathbf{L}\mathbf{L}^T\boldsymbol{\mu} = \mathbf{b}$.

Producing a random sample from the GMRF, is half the cost of solving the linear system in step 2.

All tests reported here, we conducted on a $1\,200MHz$ laptop with $512Mb$ memory running Linux.

New machines are nearly $3 - 10$ times as fast...

GMRF models in time

Let $\mathbf{Q}$ be a band-matrix with bandwidth p and dimension n.
For such a problem, using BCF will be (theoretically) optimal, as the fillin will be be zero.

	$n = 10^3$		$n = 10^4$		$n = 10^5$	
CPU-time	$p = 5$	$p = 25$	$p = 5$	$p = 25$	$p = 5$	$p = 25$
Factorise	0.0005	0.0019	0.0044	0.0271	0.0443	0.2705
Solve	0.0000	0.0004	0.0031	0.0109	0.0509	0.1052

"Long and thin" are fast!
The MSCF is less optimal for band-matrices: fillin and more complicated data-structures.

... with global nodes

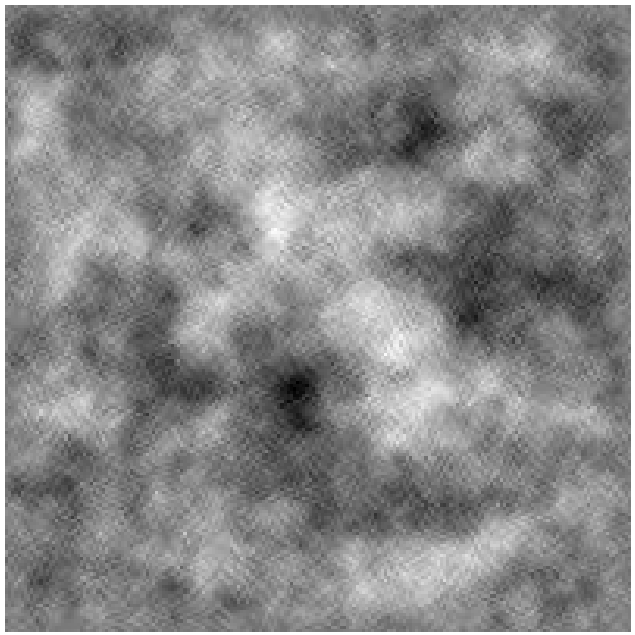
Add 10 global nodes:

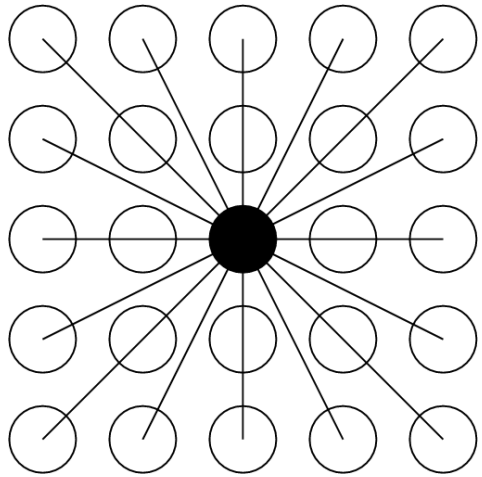
	$n = 10^3$		$n = 10^4$		$n = 10^5$	
CPU-time	$p = 5$	$p = 25$	$p = 5$	$p = 25$	$p = 5$	$p = 25$
Factorise	0.0119	0.0335	0.1394	0.4085	1.6396	4.1679
Solve	0.0007	0.0035	0.0138	0.0306	0.1541	0.3078

- The fillin $\approx pn$.
- The nested dissection ordering give good results in all cases considered so far.

Spatial GMRF models

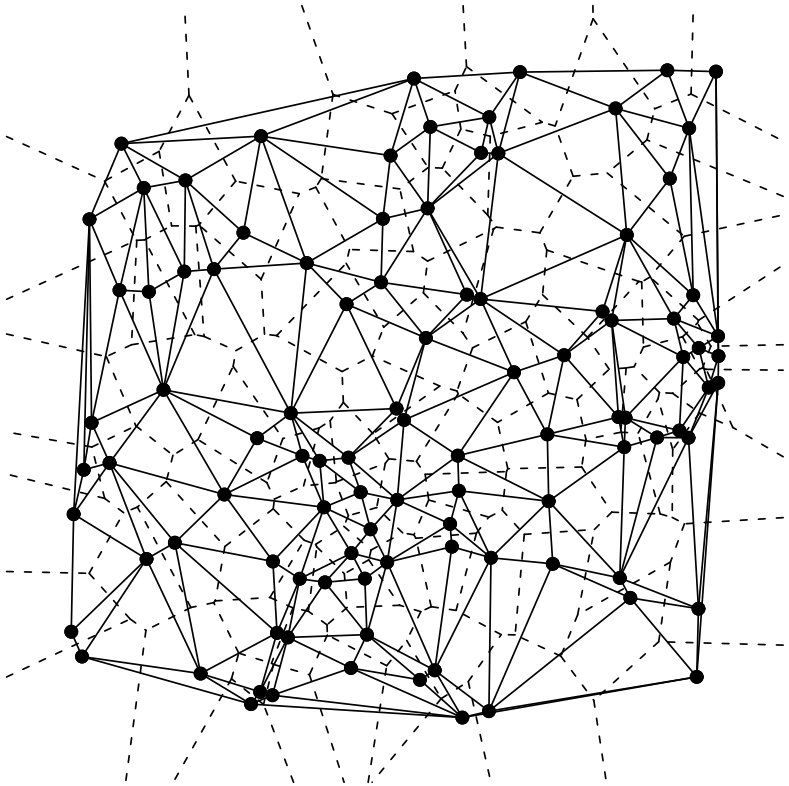
Regular lattice





Spatial GMRF models

Irregular lattice



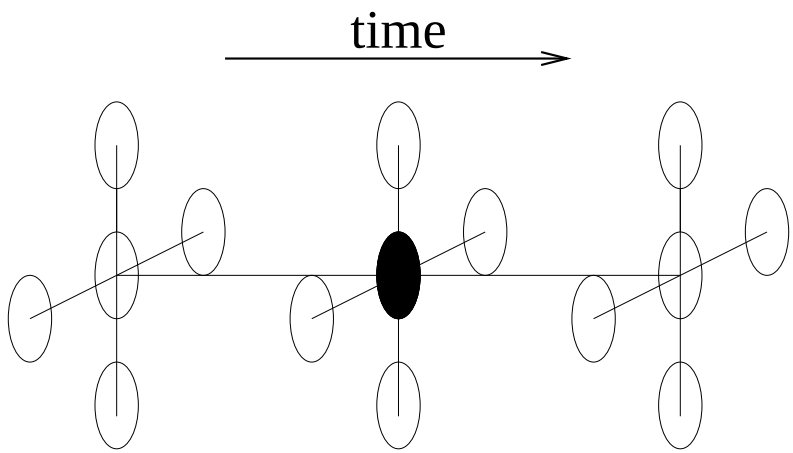
Results for spatial GMRF models

		$n = 100^2$		$n = 150^2$		$n = 200^2$	
CPU-time	Method	3×3	5×5	3×3	5×5	3×3	5×5
Factorise	BCF	0.51	1.02	2.60	4.93	13.30	38.12
	MSCF	0.17	0.62	0.55	1.92	1.91	4.90
Solve	BCF	0.03	0.05	0.10	0.16	0.24	0.43
	MSCF	0.01	0.04	0.04	0.11	0.08	0.21

- For the largest lattice the MSCF really outperform the BCF.
- The reason is the $\mathcal{O}(n^{3/2})$ cost for MSCF compared to $\mathcal{O}(n^2)$ for the BCF.

Spatio-temporal GMRF models

Spatio-temporal GMRF models is often an extension of spatial GMRF models to account to time variation.



Use this model and the graph of Germany, for $T = 10$ and $T = 100$.

Results for spatio-temporal GMRF models

			10 global nodes	
CPU-time	$T = 10$	$T = 100$	$T = 10$	$T = 100$
Factorise	0.25	39.96	0.31	39.22
Solve	0.02	0.42	0.02	0.42

- The results shows a quite heavy dependency on T.
- Spatio-Temporal GMRFs are more demanding than spatial ones, $\mathcal{O}(n^2)$ for a $n^{1/3} \times n^{1/3} \times n^{1/3}$-cube.

Next week

- Next week we will leave the computations and instead look at intrinsic GMRFs, which are very popular to use a priors
- We will also have the first lab/project.
- We will have the lecture on Monday and the lab on Tuesday (and not the other way around as first stated in the course plan)