Bayesian computation using INLA

5: Spatial Markovian models—The SPDE approach

Outline

Spatial interpolants

$$(\text{observed value})_i = (\text{true value at location } i) + (\text{error})_i$$

- We treat the *observed* covariates as being measured with error
- The errors are usually assumed to be independent and identically distributed (i.i.d.)
  - Usually, we take them to be Gaussian
  - If we think there may be outliers, we might use something else (e.g. a Student-T distribution)
  - The only change in `R-INLA`is in the `family` argument in the INLA call

So how does that help us fill in the field?

$$(\text{observed value})_i = (\text{true value at location } i) + (\text{error})_i$$

or

$$y_i = x(s_i) + \epsilon_i$$

We need priors!

- We have chosen the error distribution to be $\epsilon_i \sim N(0, \sigma^2)$
  - A *zero mean* means that there is no systemic measurement error
  - A *common variance* means that everything was measured the same way
- Now we need a prior on the truth...

Gaussian random fields

If we have a process that is occurring everywhere in space, it is natural to try to model it using some sort of function.

- This is hard!
- We typically make our lives easier by making everything Gaussian.
- What makes a function Gaussian?

Gaussian random fields

If we are trying to model $x(s)$ what sort of things do we need?

- We don't ever observe a function *everywhere*.
- If $\boldsymbol{x}$ is a vector of observations of $x(s)$ at different locations, we want this to be normally distributed:

$$\boldsymbol{x} = (x(s_1), \ldots, x(s_p))^T \sim N(\mathbf{0}, \boldsymbol{\Sigma}_{x(s_1),\ldots,x(s_p)})$$

- This is actually quite tricky: the covariance matrix $\boldsymbol{\Sigma}$ will need to depend on the set of observation sites and always has to be positive definite.
- It turns out you can actually do this by setting $\Sigma_{ij} = c(\boldsymbol{s}_i, \boldsymbol{s}_j)$ for some *covariance function* $c(\cdot, \cdot)$.
- Not every function will ensure that $\boldsymbol{\Sigma}$ is positive definite!

Gaussian random fields

If we are trying to model $x(s)$ what sort of things do we need?

- We don't ever observe a function *everywhere*.
- If $\boldsymbol{x}$ is a vector of observations of $x(s)$ at different locations, we want this to be normally distributed:

$$\boldsymbol{x} = (x(s_1), \ldots, x(s_p))^T \sim N(\mathbf{0}, \boldsymbol{\Sigma}_{x(s_1),\ldots,x(s_p)})$$

- This is actually quite tricky: the covariance matrix $\boldsymbol{\Sigma}$ will need to depend on the set of observation sites and always has to be positive definite.
- It turns out you can actually do this by setting $\Sigma_{ij} = c(\boldsymbol{s}_i, \boldsymbol{s}_j)$ for some *covariance function* $c(\cdot, \cdot)$.
- Not every function will ensure that $\boldsymbol{\Sigma}$ is positive definite!

Gaussian random fields

If we are trying to model $x(s)$ what sort of things do we need?

- We don't ever observe a function *everywhere*.
- If $\boldsymbol{x}$ is a vector of observations of $x(s)$ at different locations, we want this to be normally distributed:

$$\boldsymbol{x} = (x(s_1), \ldots, x(s_p))^T \sim N(\mathbf{0}, \boldsymbol{\Sigma}_{x(s_1),\ldots,x(s_p)})$$

- This is actually quite tricky: the covariance matrix $\boldsymbol{\Sigma}$ will need to depend on the set of observation sites and always has to be positive definite.
- It turns out you can actually do this by setting $\Sigma_{ij} = c(\boldsymbol{s}_i, \boldsymbol{s}_j)$ for some *covariance function* $c(\cdot, \cdot)$.
- Not every function will ensure that $\boldsymbol{\Sigma}$ is positive definite!

Gaussian random fields

If we are trying to model $x(s)$ what sort of things do we need?

- We don't ever observe a function *everywhere*.
- If $\boldsymbol{x}$ is a vector of observations of $x(s)$ at different locations, we want this to be normally distributed:

$$\boldsymbol{x} = (x(s_1), \ldots, x(s_p))^T \sim N(\mathbf{0}, \boldsymbol{\Sigma}_{x(s_1),\ldots,x(s_p)})$$

- This is actually quite tricky: the covariance matrix $\boldsymbol{\Sigma}$ will need to depend on the set of observation sites and always has to be positive definite.
- It turns out you can actually do this by setting $\Sigma_{ij} = c(\boldsymbol{s}_i, \boldsymbol{s}_j)$ for some *covariance function* $c(\cdot, \cdot)$.
- Not every function will ensure that $\boldsymbol{\Sigma}$ is positive definite!

Gaussian random fields

If we are trying to model $x(s)$ what sort of things do we need?

- We don't ever observe a function *everywhere*.
- If $\boldsymbol{x}$ is a vector of observations of $x(s)$ at different locations, we want this to be normally distributed:

$$\boldsymbol{x} = (x(s_1), \ldots, x(s_p))^T \sim N(\mathbf{0}, \boldsymbol{\Sigma}_{x(s_1),\ldots,x(s_p)})$$

- This is actually quite tricky: the covariance matrix $\boldsymbol{\Sigma}$ will need to depend on the set of observation sites and always has to be positive definite.
- It turns out you can actually do this by setting $\Sigma_{ij} = c(\boldsymbol{s}_i, \boldsymbol{s}_j)$ for some *covariance function* $c(\cdot, \cdot)$.
- Not every function will ensure that $\boldsymbol{\Sigma}$ is positive definite!

A good "first model"

Stationary random fields

A GRF is stationary if:

- has mean zero.
- the covariance between two points depends only on the distance and direction between those points.

It is isotropic if the covariance only depends on the *distance between the points*.

- Zero mean $\longrightarrow$ remove the mean
- Stationarity is a *mathematical assumption* and may have no bearing on reality
- But it makes lots of things easier.

A good "first model"

Stationary random fields

A GRF is stationary if:

- has mean zero.
- the covariance between two points depends only on the distance and direction between those points.

It is isotropic if the covariance only depends on the *distance between the points*.

- Zero mean $\longrightarrow$ remove the mean
- Stationarity is a *mathematical assumption* and may have no bearing on reality
- But it makes lots of things easier.

A good "first model"

Stationary random fields

A GRF is stationary if:

- has mean zero.
- the covariance between two points depends only on the distance and direction between those points.

It is isotropic if the covariance only depends on the *distance between the points*.

- Zero mean $\longrightarrow$ remove the mean
- Stationarity is a *mathematical assumption* and may have no bearing on reality
- But it makes lots of things easier.

A good "first model"

Stationary random fields

A GRF is stationary if:

- has mean zero.
- the covariance between two points depends only on the distance and direction between those points.

It is isotropic if the covariance only depends on the *distance between the points*.

- Zero mean $\longrightarrow$ remove the mean
- Stationarity is a *mathematical assumption* and may have no bearing on reality
- But it makes lots of things easier.

The three typical parameters for a GRF

- The variance (or precision) parameter:
  - This controls how wildly the function can deviate from its mean
- The range parameter
  - This controls the range over which the correlation between $x(\boldsymbol{s})$ and $x(\boldsymbol{s}+\boldsymbol{h})$ is essentially zero
  - Often the "range" parameter is some transformation of this distance
- The smoothness parameter
  - Controls how differentiable the field is.
  - This essentially controls how similar nearby points are
  - Often not jointly identifiable with the range

For isotropic random fields, these parameters are constant.

The three typical parameters for a GRF

- The variance (or precision) parameter:
  - This controls how wildly the function can deviate from its mean
- The range parameter
  - This controls the range over which the correlation between $x(\boldsymbol{s})$ and $x(\boldsymbol{s} + \boldsymbol{h})$ is essentially zero
  - Often the "range" parameter is some transformation of this distance
- The smoothness parameter
  - Controls how differentiable the field is.
  - This essentially controls how similar nearby points are
  - Often not jointly identifiable with the range

For isotropic random fields, these parameters are constant.

The three typical parameters for a GRF

- The variance (or precision) parameter:
  - This controls how wildly the function can deviate from its mean
- The range parameter
  - This controls the range over which the correlation between $x(\boldsymbol{s})$ and $x(\boldsymbol{s}+\boldsymbol{h})$ is essentially zero
  - Often the "range" parameter is some transformation of this distance
- The smoothness parameter
  - Controls how differentiable the field is.
  - This essentially controls how similar nearby points are
  - Often not jointly identifiable with the range

For isotropic random fields, these parameters are constant.

Gaussian random fields

Defn: Gaussian random fields

A Gaussian random field $x(\boldsymbol{s})$ is defined by a mean function $\boldsymbol{\mu}(\boldsymbol{s})$ and a covariance function $c(\boldsymbol{s}_1, \boldsymbol{s}_2)$. It has the property that, for every finite collection of points $\{s_1, \ldots, s_p\}$,

$$\boldsymbol{x} \equiv (x(s_1), \ldots, x(s_p))^T \sim N(\boldsymbol{0}, \boldsymbol{\Sigma}),$$

where $\Sigma_{ij} = c(s_i, s_j)$.

- $\boldsymbol{\Sigma}$ will almost never be sparse.
- It is typically very hard to find families of parameterised covariance functions.
- It isn't straightforward to make this work for multivariate, spatiotemporal, or processes on non-flat spaces.

How big is the problem?

Let's do a quick operation count!

- Parameter estimation: Requires the field at N data points: Must factor an $N \times N$ matrix
- Spatial prediction (Kriging): Requires the field at m points, probably densely through the domain: Must factor an $m \times m$ matrix.
- Joint parameter and spatial estimation: Needs it at both. Must factor a $(N + m) \times (N + m)$.

Working with dense matrices

- Storage:
  - $\mathcal{O}(N^2)$
  - 2500 points for 20 years requires $\sim$ 20 Gbytes
- Computation:
  - Each sample, solve, or determinant costs $\mathcal{O}(N^3)$.
  - We always need quite a few of these (likelihood, VB, INLA)
  - If we use MCMC we need a gargantuan number!
    - Remember $1,000,000$ samples give $\sim$ 3 decimal places of accuracy.

Clearly this won't work if N is large.

Working with dense matrices

- Storage:
  - $\mathcal{O}(N^2)$
  - 2500 points for 20 years requires $\sim$ 20 Gbytes
- Computation:
  - Each sample, solve, or determinant costs $\mathcal{O}(N^3)$.
  - We always need quite a few of these (likelihood, VB, INLA)
  - If we use MCMC we need a gargantuan number!
    - Remember $1,000,000$ samples give $\sim$ 3 decimal places of accuracy.

Clearly this won't work if N is large.

Working with dense matrices

- Storage:
  - $\mathcal{O}(N^2)$
  - 2500 points for 20 years requires $\sim$ 20 Gbytes
- Computation:
  - Each sample, solve, or determinant costs $\mathcal{O}(N^3)$.
  - We always need quite a few of these (likelihood, VB, INLA)
  - If we use MCMC we need a gargantuan number!
    - Remember $1,000,000$ samples give $\sim$ 3 decimal places of accuracy.

Clearly this won't work if N is large.

What if our matrices magically became sparse?

Sparse covariance matrices can be formed using covariance tapering or compactly supported covariance functions.

- Storage:
  - $\mathcal{O}(N)$
  - 2500 points for 20 years requires $\sim$ 400 Kilobytes
- Computation:
  - Each sample, solve, or determinant costs $\sim \mathcal{O}(N^{3/2})$.

Question: Can we find random field that 'magically' give us sparse matrices?

What if our matrices magically became sparse?

Sparse covariance matrices can be formed using covariance tapering or compactly supported covariance functions.

- Storage:
  - $\mathcal{O}(N)$
  - 2500 points for 20 years requires $\sim$ 400 Kilobytes
- Computation:
  - Each sample, solve, or determinant costs $\sim \mathcal{O}(N^{3/2})$.

Question: Can we find random field that 'magically' give us sparse matrices?

Outline

The Goldilocks principle

So modelling directly with GRFs is too hard, while GMRFs can be difficult when they're not on a lattice.

Is there something "just right"?

It is instructive to consider another "awkward" method that aims to fix the computational problems of GRFs while keeping a continuous specification (hence avoiding the lattice problem).

Combining this method with GMRFs will lead to (finally!) a stable, flexible, computationally feasible method.

Reducing the dimension

Most of the methods aimed at reducing the "big N problem" in spatial statistics is based on some sort of low-dimensional approximation:

$$x(s) \approx \sum_{i=1}^{n} w_i \phi_i(s),$$

where $\boldsymbol{w}$ is jointly Gaussian and $\phi_i(s)$ are a set of known deterministic functions.

If $\boldsymbol{w} \sim N(\mathbf{0}, \boldsymbol{\Sigma})$, then the covariance function of $x(\boldsymbol{s})$ is

$$c(\boldsymbol{s}_1, \boldsymbol{s}_2) = \boldsymbol{\Phi}(\boldsymbol{s}_1)^T \boldsymbol{\Sigma} \boldsymbol{\Phi}(\boldsymbol{s}_2),$$

where $\boldsymbol{\Phi}(\boldsymbol{s})$ is a vector with the ϕ_i functions evaluated at point $\boldsymbol{s}$.

Kernel methods

We will take a close look at Kernel methods and, in particular, when they fail.

- They are popular!
- They are easy to analyse!
- They are prototypical of this low-dimensional (please don't say "low-rank"!!) approach.

Kernel representations

Most GRFs can be represented as

$$x(s) = \int_{\mathbb{R}^2} k(s,t)\, dW(t),$$

where $W(t)$ is white noise, and $k(s,t)$ is a deterministic "kernel" function.

- It is often suggested that we model $k(\cdot,\cdot)$ directly.
- We can approximate the integral by a sum (Higdon, '98)

$$x(s) \approx \sum_{i=1}^{n} k(x,t_i)\xi_i,$$

where ξ_i are i.i.d. normals.

- This does not work well. (S, Lindgren, Rue, '10, Bolin and Lindgren '10)

Kernel representations

Most GRFs can be represented as

$$x(s) = \int_{\mathbb{R}^2} k(s,t)\, dW(t),$$

where $W(t)$ is white noise, and $k(s,t)$ is a deterministic "kernel" function.

- It is often suggested that we model $k(\cdot,\cdot)$ directly.
- We can approximate the integral by a sum (Higdon, '98)

$$x(s) \approx \sum_{i=1}^{n} k(x,t_i)\xi_i,$$

  where ξ_i are i.i.d. normals.
- This does not work well. (S, Lindgren, Rue, '10, Bolin and Lindgren '10)

Kernel representations

Most GRFs can be represented as

$$x(s) = \int_{\mathbb{R}^2} k(s,t)\, dW(t),$$

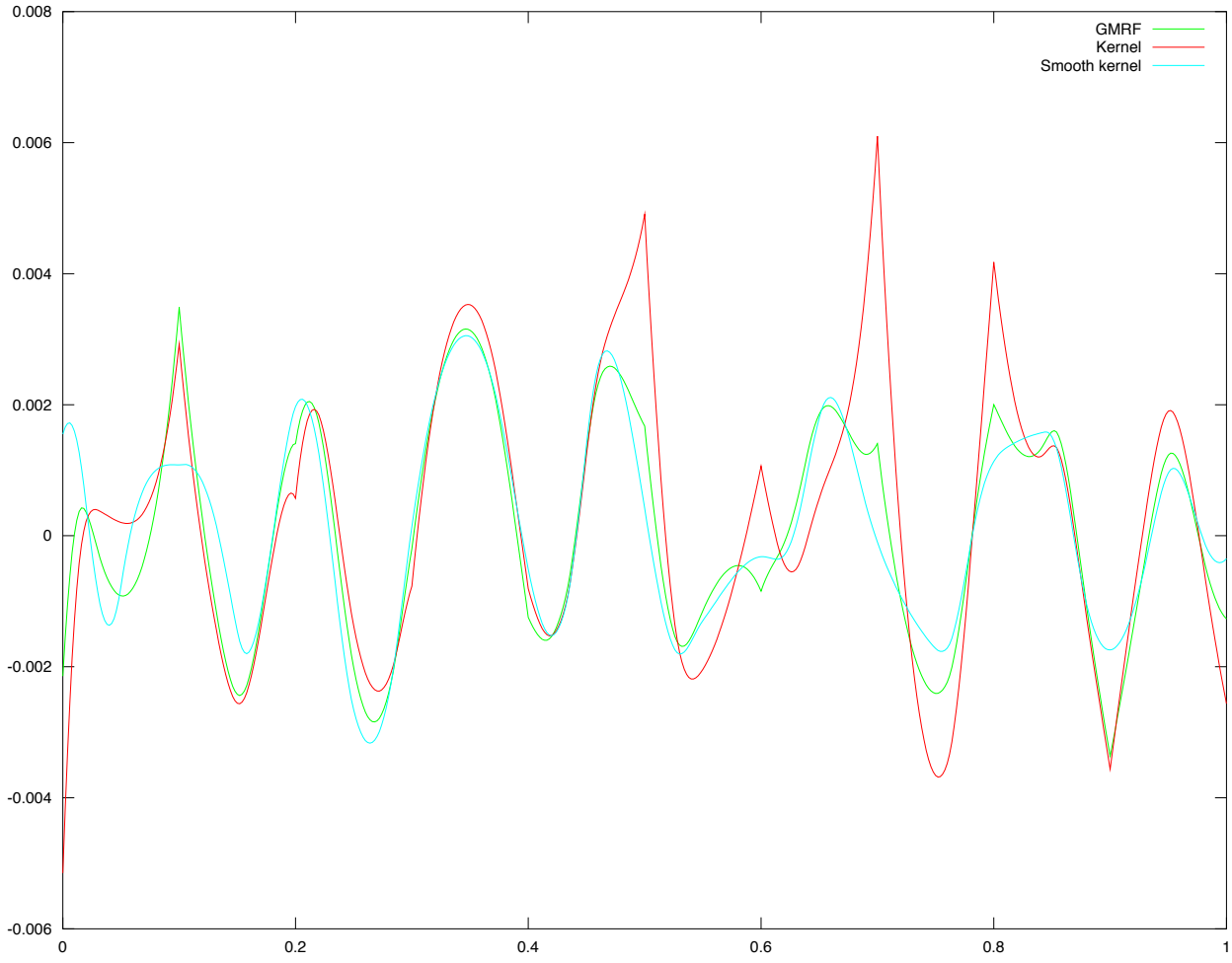
where $W(t)$ is white noise, and $k(s,t)$ is a deterministic "kernel" function.

- It is often suggested that we model $k(\cdot,\cdot)$ directly.
- We can approximate the integral by a sum (Higdon, '98)

$$x(s) \approx \sum_{i=1}^{n} k(x,t_i)\xi_i,$$

  where ξ_i are i.i.d. normals.
- This does not work well. (S, Lindgren, Rue, '10, Bolin and Lindgren '10)

So what happens?



Approximation properties

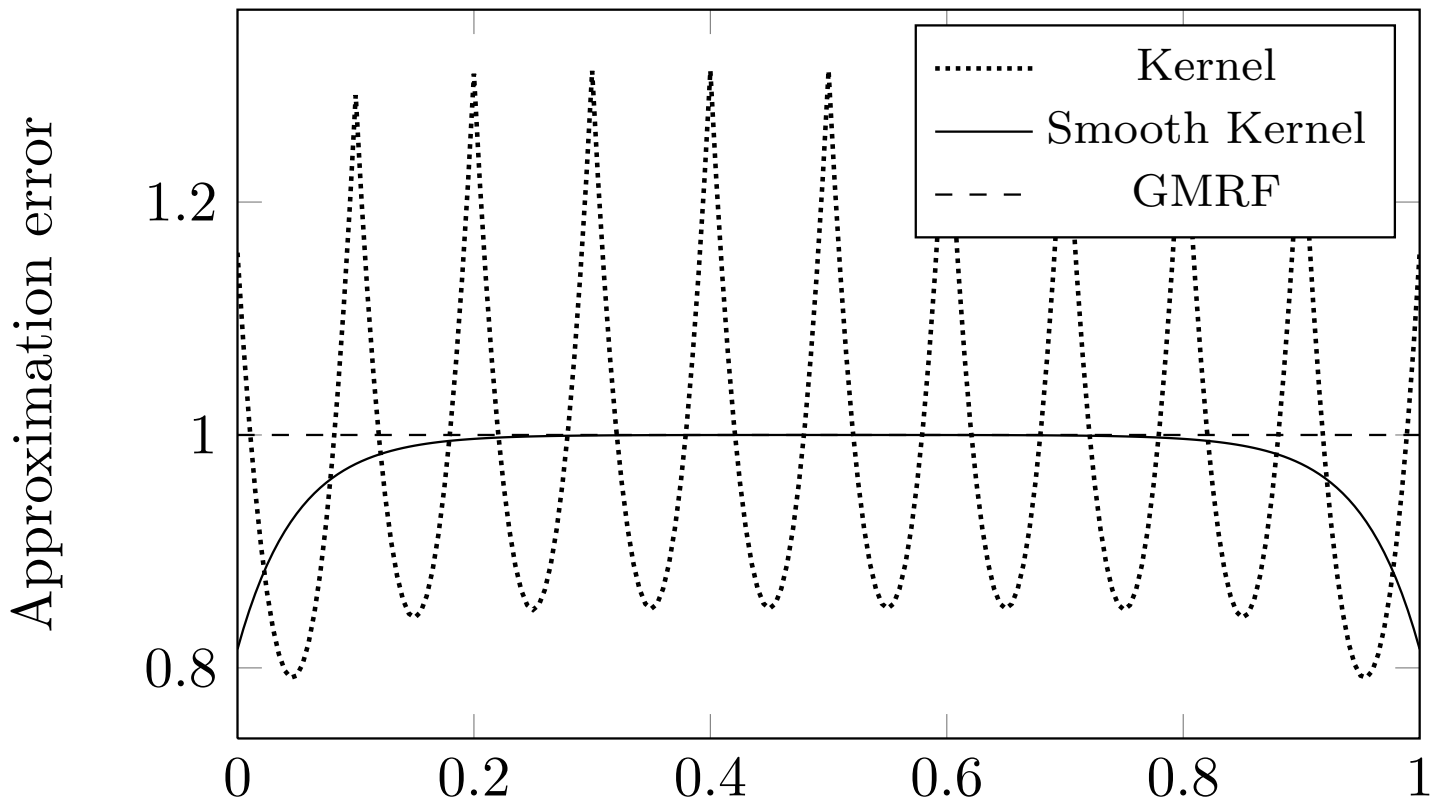
Realisations of Gaussian Random Fields are *functions*.

Appropriate question

How well can realisations of $x(\cdot)$ be approximated by functions of the form $\sum_{i=1}^{n} w_i \phi_i(s)$.

- This is *not* an asymptotic question! n never goes to infinity.
- Without considering these questions, you *cannot* know how a method will work!

Best Kernel approximation to a constant



Why did kernel methods perform badly?

- Kernel methods performed badly because there weren't enough points.
- Kernel methods performed badly because the range was smaller than the grid spacing.
- Kernel methods performed badly because the basis functions depend on the parameter being inferred!

This is a common problem and leads to "spotty" spatial predictions and bad uncertainty estimates.

Why did kernel methods perform badly?

- Kernel methods performed badly because there weren't enough points.
- Kernel methods performed badly because the range was smaller than the grid spacing.
- Kernel methods performed badly because the basis functions depend on the parameter being inferred!

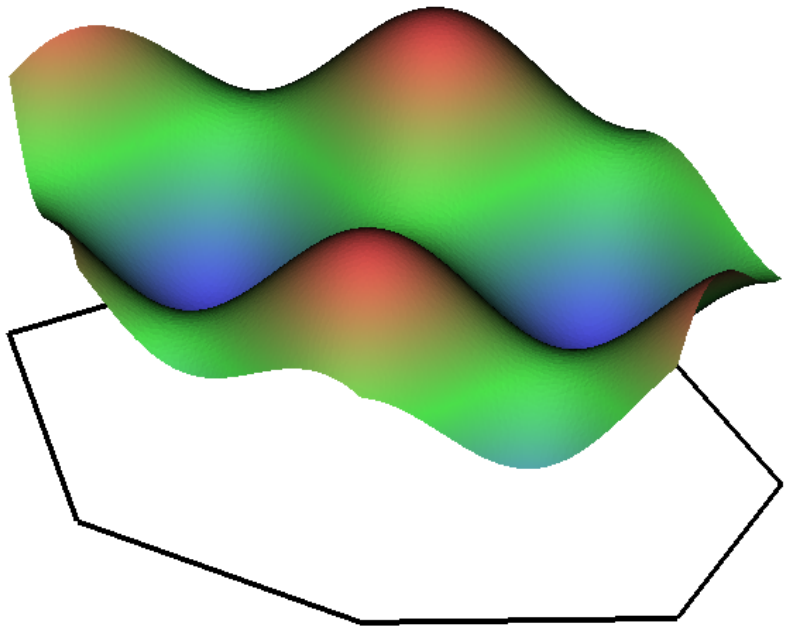
This is a common problem and leads to "spotty" spatial predictions and bad uncertainty estimates.

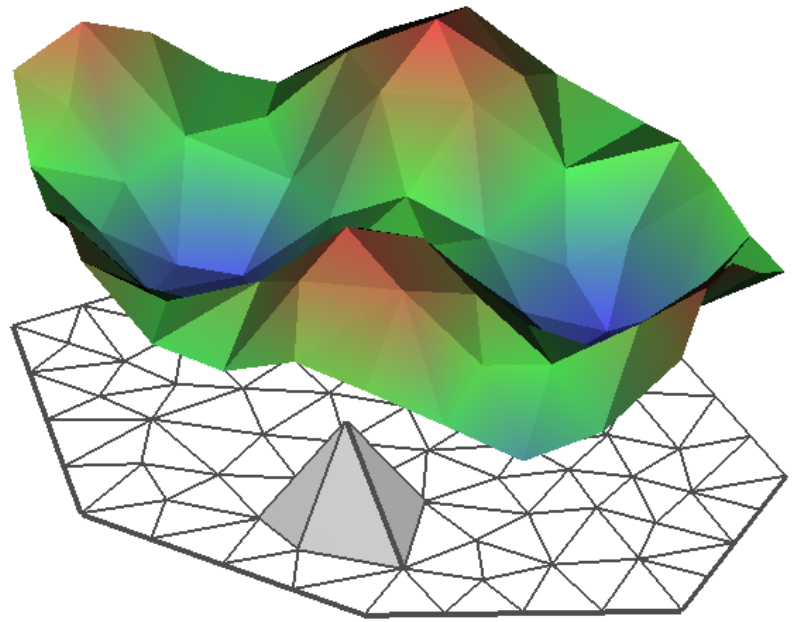
Why did kernel methods perform badly?

- Kernel methods performed badly because there weren't enough points.
- Kernel methods performed badly because the range was smaller than the grid spacing.
- Kernel methods performed badly because the basis functions depend on the parameter being inferred!

This is a common problem and leads to "spotty" spatial predictions and bad uncertainty estimates.

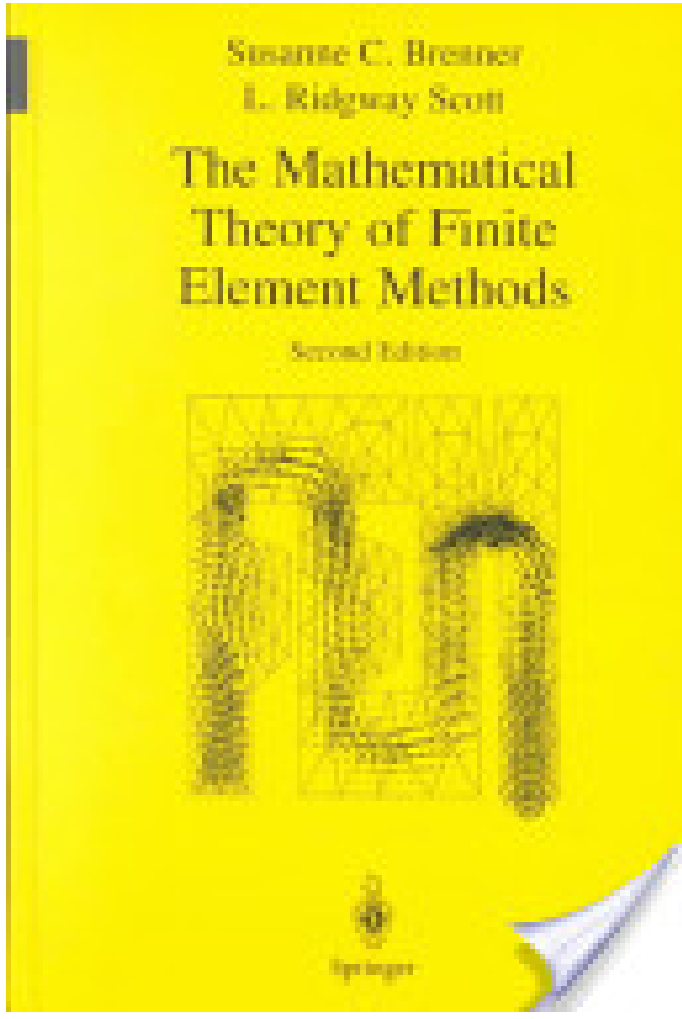
Piecewise linear approximation of surfaces





NB: The basis functions are only non-zero on a small part of the domain.

Known approximation properties



How can we use these functions?

There is no obvious way to use piecewise linear functions...

Outline

SPDE models

In this section we're going to look at a class of flexible, computationally efficient spatial models.

- These models will give you a lot of the flexibility of GRFs without the pain
- They have the good computational properties of GMRFs, but with more flexibility
- They are defined continuously like Kernel methods, but are stable
- Don't focus too much on the theory (unless you want to!)

SPDE models

In this section we're going to look at a class of flexible, computationally efficient spatial models.

- These models will give you a lot of the flexibility of GRFs without the pain
- They have the good computational properties of GMRFs, but with more flexibility
- They are defined continuously like Kernel methods, but are stable
- Don't focus too much on the theory (unless you want to!)

SPDE models

In this section we're going to look at a class of flexible, computationally efficient spatial models.

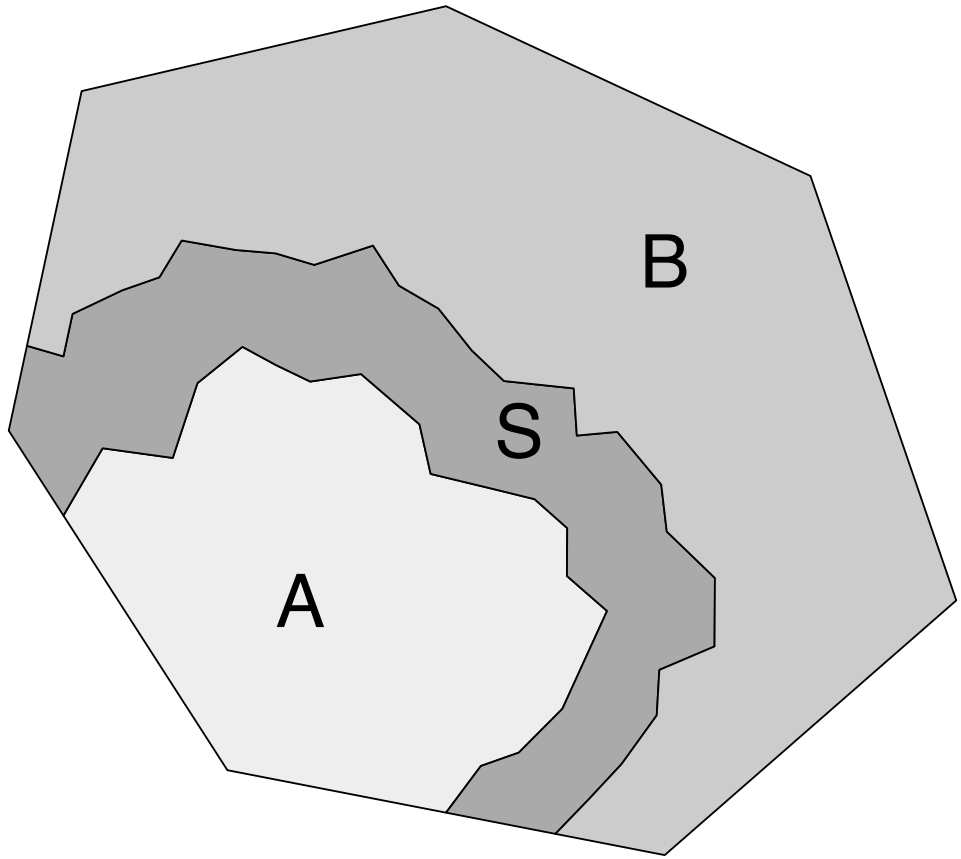
- These models will give you a lot of the flexibility of GRFs without the pain
- They have the good computational properties of GMRFs, but with more flexibility
- They are defined continuously like Kernel methods, but are stable
- Don't focus too much on the theory (unless you want to!)

SPDE models

In this section we're going to look at a class of flexible, computationally efficient spatial models.

- These models will give you a lot of the flexibility of GRFs without the pain
- They have the good computational properties of GMRFs, but with more flexibility
- They are defined continuously like Kernel methods, but are stable
- Don't focus too much on the theory (unless you want to!)

The secret is in the Markov property



How does this translate to maths?

General Result

The power spectrum of a stationary Markovian Gaussian random field has the form $R(\boldsymbol{k}) = 1/p(\boldsymbol{k})$, where $p(\boldsymbol{k})$ is a positive, symmetric polynomial.

Oh dear!

Can we salvage something from this?

Sometimes it's useful to be an engineer!

An engineering calculation

Let L be a differential operator. Then the solution to

$$Lx(s) = W(\cdot)$$

is a Gaussian random field and it has the Markov property.

- "Prove" it using Fourier transforms.
- The derivatives (local) produce the Markov property (local)
- Now we're solving (partial) differential equations: *standard!*

Can we salvage something from this?

Sometimes it's useful to be an engineer!

An engineering calculation

Let L be a differential operator. Then the solution to

$$Lx(s) = W(\cdot)$$

is a Gaussian random field and it has the Markov property.

- "Prove" it using Fourier transforms.
- The derivatives (local) produce the Markov property (local)
- Now we're solving (partial) differential equations: *standard!*

Can we salvage something from this?

Sometimes it's useful to be an engineer!

An engineering calculation

Let L be a differential operator. Then the solution to

$$Lx(s) = W(\cdot)$$

is a Gaussian random field and it has the Markov property.

- "Prove" it using Fourier transforms.
- The derivatives (local) produce the Markov property (local)
- Now we're solving (partial) differential equations: *standard!*

What does this remind us of?

- Recall our SAR(1) model (slightly re-written)

$$4x_i - (x_n + x_s + x_e + x_w) \sim N(0, \sigma^2).$$

- Also remember that

$$-\frac{d^2x}{ds^2} \approx \frac{-x(s+h) + 2x(s) - x(s-h)}{h^2}$$

- So if we scale our SAR(1) model and let the lattice spacing $h \to 0$, we get

$$-\Delta x(s) \equiv -\left(\frac{d^2x}{ds_1^2} + \frac{d^2x}{ds_2^2}\right) \stackrel{d}{=} W(s)$$

In the context of GMRFs

In this context, this was first noted by Whittle in the 50s (!!) who noted that Matérn fields, which have covariance function of the form

$$c(x, y) \propto (\kappa \|x - y\|)^{\nu} K_{\nu} (\kappa \|x - y\|),$$

are the stationary solutions to the SPDE

$$(\kappa^2 - \Delta)^{\frac{\nu + d/2}{2}} x(s) = W(s),$$

where

- $\Delta = \sum_{i=1}^{d} \frac{\partial^2}{\partial s_i^2}$ is the Laplacian
- $W(s)$ is spatial white noise.
- The parameter ν controls the smoothness.
- The parameter κ controls the range.

Practical interpretation of the parameters

We have

$$(\kappa^2 - \Delta)^{\frac{\alpha}{2}}(\tau x(s)) = W(s),$$

where $\alpha = \nu + d/2$ is an integer.

- κ^2 is a range parameter. The approximate range is

$$\texttt{range} \approx \frac{\sqrt{8\nu}}{\kappa}.$$

- The variance of the model is

$$\sigma^2 = \frac{\Gamma(\nu)}{\gamma(\nu + d/2)(4\pi)^{d/2}\kappa^{2\nu}\tau^2}.$$

So which models do we get?

So, according to the Whittle characterisation of the Matérn covariance functions, we get a Markovian random field when $\alpha = \nu + d/2$ is an integer. When d is odd, Matérn models with $\nu \in 1/2\mathbb{N}$.

- This include the Thin Plate Spline model ($\nu = 1$)
- And the exponential covariance ($\nu = 1/2$).

So which models do we get?

When d is even, we get Matérn models with $\nu \in \mathbb{N}$

- This include the Thin Plate Spline model ($\nu = 1$)
- But not the exponential covariance ($\nu = 1/2$).

So which models do we get?

When d is even, we get Matérn models with $\nu \in \mathbb{N}$

- This include the Thin Plate Spline model ($\nu = 1$)
- But not the exponential covariance ($\nu = 1/2$).

Let's simplify things: set $\nu + d/2 = 2$

The SPDE becomes

$$(\kappa^2 - \Delta)x(s) = W(s),$$

which only involves second derivatives, which are nice.

- Connection to thin plate splines!

Approximating the SPDE

We are looking for the piecewise linear random field

$$x_n(s) = \sum_{i=1}^{n} w_i \phi_i(s)$$

for piecewise linear functions $\phi_i(s)$ that *best* approximates the solution to $(\kappa^2 - \Delta)x(s) = W(s)$.

Step 1: The 'weak' solution

White noise is weird, but if we integrate it, it becomes nicer. So we require that for every suitable function $\phi(s)$,

$$\int_\Omega \psi(s)(\kappa^2 - \Delta)x(s)\, ds \stackrel{D}{=} \int_\Omega \psi(s)\, dW(s).$$

- White noise integrals aren't scary!

$$\int_\Omega \psi(s)\, dW(s) \sim N(0, \int_\Omega \psi(s)^2\, ds)$$

Step 2: Plug in the basis functions

Replace $x(s)$ with the basis function expansions and chose $\phi(s)$ to be the set of basis functions

We get the system of linear equations

$$\int_\Omega \phi_j(s)(\kappa^2 - \Delta)\left(\sum_i w_i \phi_i(s)\right) ds \stackrel{D}{=} \int_\Omega \phi_j(s)\, dW(s)$$

This is good— LHS has things we can compute, RHS has integrals of white noise.

Step 2: Plug in the basis functions

Replace $x(s)$ with the basis function expansions and chose $\phi(s)$ to be the set of basis functions

We get the system of linear equations

$$\int_\Omega \phi_j(s)(\kappa^2 - \Delta)\left(\sum_i w_i \phi_i(s)\right) ds \stackrel{D}{=} \int_\Omega \phi_j(s)\, dW(s)$$

This is good— LHS has things we can compute, RHS has integrals of white noise.

What comes out?

We get two matrices:

- $\mathbf{C}_{ij} = \int_\Omega \phi_i(s)\, ds$ (the constant terms)
- $\mathbf{K}_{ij} = \int_\Omega \nabla\phi_i(s) \cdot \nabla\phi_j(s)\, ds$ (the Laplacian term)

The (scary) SPDE becomes the (normal) equation

$$(\kappa^2\mathbf{C} + \mathbf{K})\mathbf{w} \sim \mathcal{N}(0, \mathbf{C})$$

and therefore $\mathbf{w}$ is a GMRF with precision matrix

$$\mathbf{Q} = (\kappa^2\mathbf{C} + \mathbf{K})^T \mathbf{C}^{-1} (\kappa^2\mathbf{C} + \mathbf{K}).$$

What comes out?

We get two matrices:

- $\mathbf{C}_{ij} = \int_\Omega \phi_i(s)\, ds$ (the constant terms)
- $\mathbf{K}_{ij} = \int_\Omega \nabla\phi_i(s) \cdot \nabla\phi_j(s)\, ds$ (the Laplacian term)

The (scary) SPDE becomes the (normal) equation

$$(\kappa^2\mathbf{C} + \mathbf{K})\mathbf{w} \sim \mathcal{N}(0, \mathbf{C})$$

and therefore $\mathbf{w}$ is a GMRF with precision matrix

$$\mathbf{Q} = \left(\kappa^2\mathbf{C} + \mathbf{K}\right)^T \mathbf{C}^{-1} \left(\kappa^2\mathbf{C} + \mathbf{K}\right).$$

Notes

- This works for any Matérn field where $\alpha = \nu - d/2$ is an integer.
- More importantly, this works for any SPDE $Lx = W$.
- More importantly, if we $Q = L^*L$, this method can be applied directly to the precision operator Q.
- We can approximate non-Markov fields by Markovian fields by approximating $1/R(\boldsymbol{k})$ by a polynomial.

Notes

- This works for any Matérn field where $\alpha = \nu - d/2$ is an integer.
- More importantly, this works for any SPDE $Lx = W$.
- More importantly, if we $Q = L^*L$, this method can be applied directly to the precision operator Q.
- We can approximate non-Markov fields by Markovian fields by approximating $1/R(\boldsymbol{k})$ by a polynomial.

Notes

- This works for any Matérn field where $\alpha = \nu - d/2$ is an integer.
- More importantly, this works for any SPDE $Lx = W$.
- More importantly, if we $Q = L^*L$, this method can be applied directly to the precision operator Q.
- We can approximate non-Markov fields by Markovian fields by approximating $1/R(\boldsymbol{k})$ by a polynomial.

Notes

- This works for any Matérn field where $\alpha = \nu - d/2$ is an integer.
- More importantly, this works for any SPDE $Lx = W$.
- More importantly, if we $Q = L^*L$, this method can be applied directly to the precision operator Q.
- We can approximate non-Markov fields by Markovian fields by approximating $1/R(\boldsymbol{k})$ by a polynomial.

Outline

Leukaemia survival

Leukaemia survival data (Henderson et al, 2002, JASA), 1043 cases.

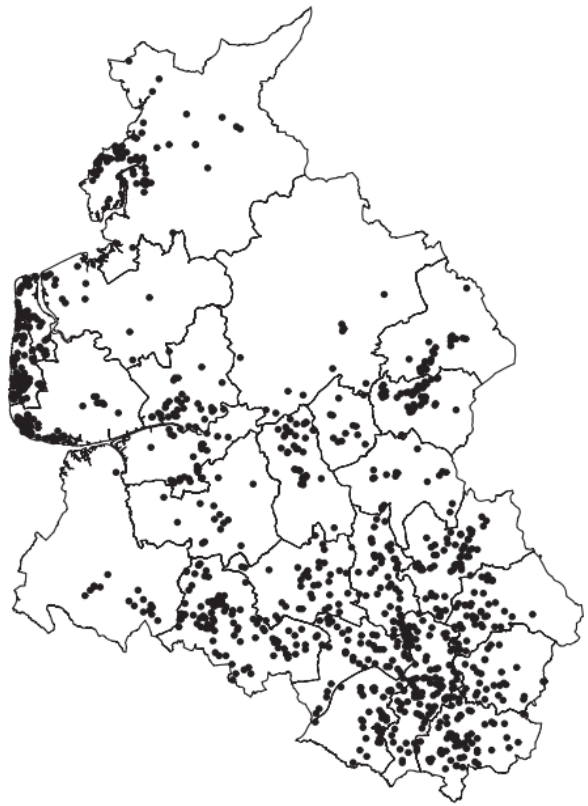


Fig. 1. Leukaemia survival data: districts of Northwest England and locations of the observations.

Survival models

Survival models are different for many models in statistics. There are two types of observations: an event (death) or we stop measuring (censoring).

Rather than directly modelling the *hazard* (instantaneous risk)

$$\begin{aligned} h(y)\ dy &= \text{Prob}(y \leq Y < y + \ dy | Y > y) \\ h(y) &= \frac{f(t)}{S(t)} \end{aligned}$$

Cox proportional hazards model

Write the hazard function for each patient as:

$$h(y_i|w_i, \boldsymbol{x}_i) = h_0(y_i)\ w_i\ \exp(\boldsymbol{c}_i^T \boldsymbol{\beta}) \exp(x(s_i));\ \ i = 1, \ldots, 1043$$

where

$h_0(\cdot)$	is the baseline hazard function
w_i	is the log-Normal frailty effect associated with patient i
$\boldsymbol{c}_i$	is the vector of observed covariates for patient i
$\boldsymbol{\beta}$	is a vector of unknown parameters
$x(s_i)$	is the value of the spatial effect $x(s)$ for patient i.

Spatial survival: example

$$\log(\text{hazard}) = \log(\text{baseline})$$
$$+ f(\text{age})$$
$$+ f(\text{white blood cell count})$$
$$+ f(\text{deprivation index})$$
$$+ f(\text{spatial})$$
$$+ \text{sex}$$

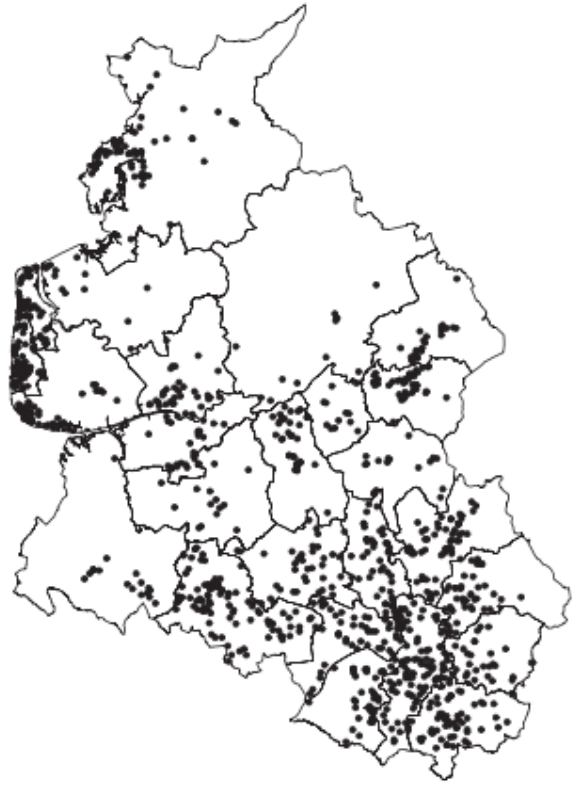
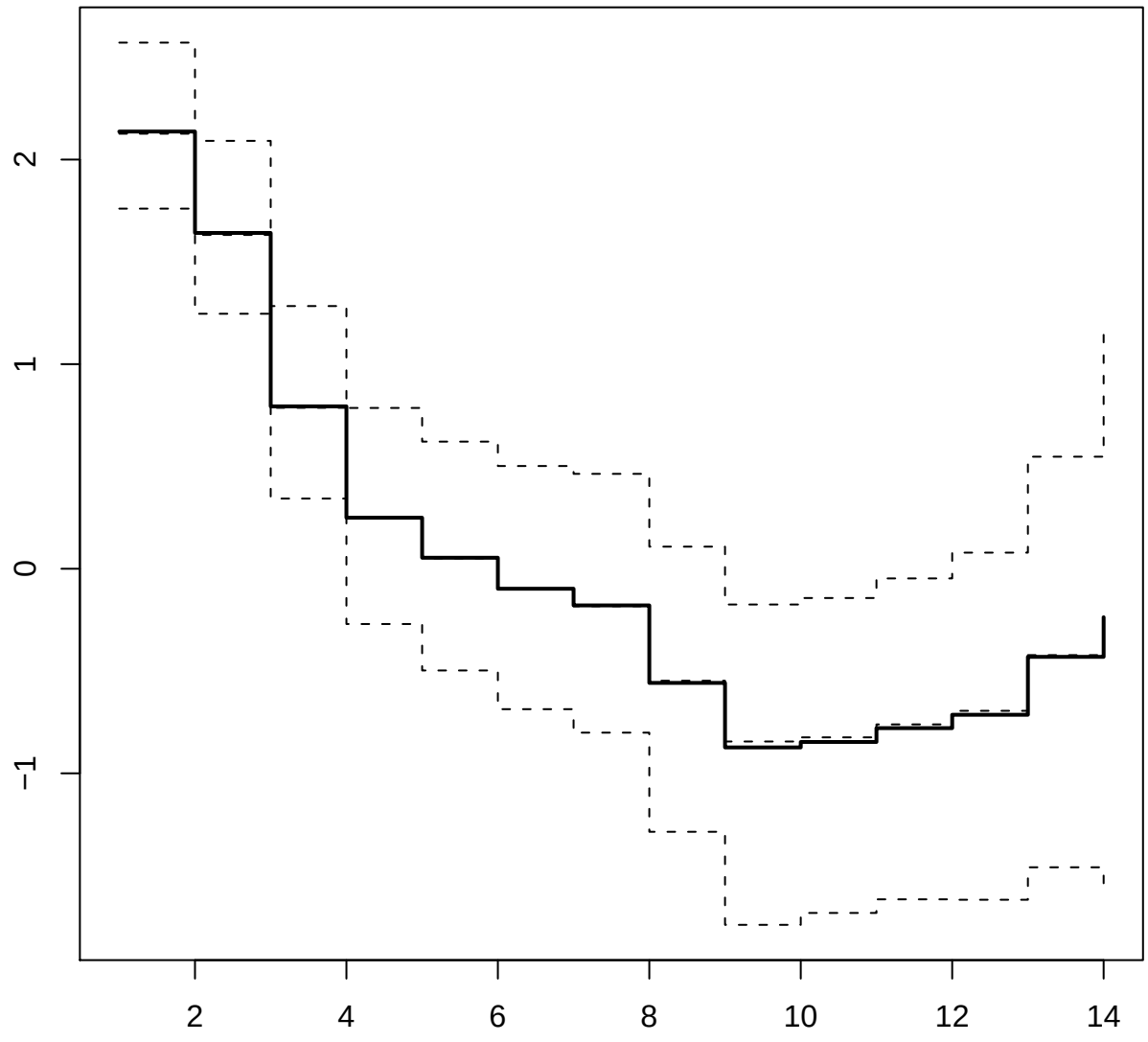


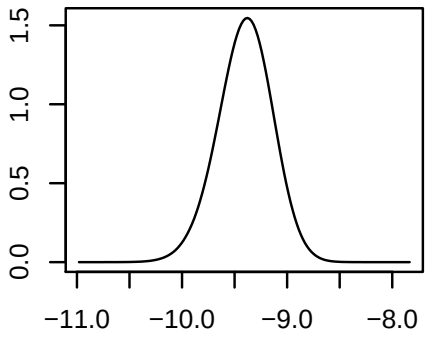
Fig. 1. Leukaemia survival data: districts of Northwest England and locations of the obse

R-code (regions)

```
data(Leuk)
g = system.file("demodata/Leuk.graph", package="INLA")

formula = inla.surv(Leuk$time, Leuk$cens) ~ sex + age +
    f(inla.group(wbc), model="rw1")+
    f(inla.group(tpi), model="rw2")+
    f(district, model="besag", graph = g)

result = inla(formula, family="coxph", data=Leuk)

source(system.file("demodata/Leuk-map.R", package="INLA"))
Leuk.map(result$summary.random$district$mean)
plot(result)
```

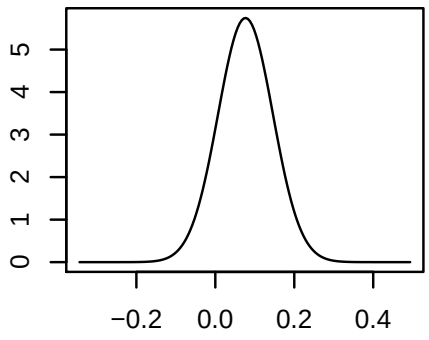

baseline.hazard
2
1
0
−1
2
4
6
8
10
12
14
PostMean 0.025% 0.5% 0.975%

PostDens [(Intercept)]



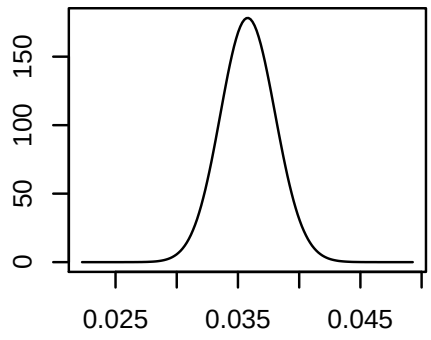
Mean = −9.401 SD = 0.261

PostDens [sex]

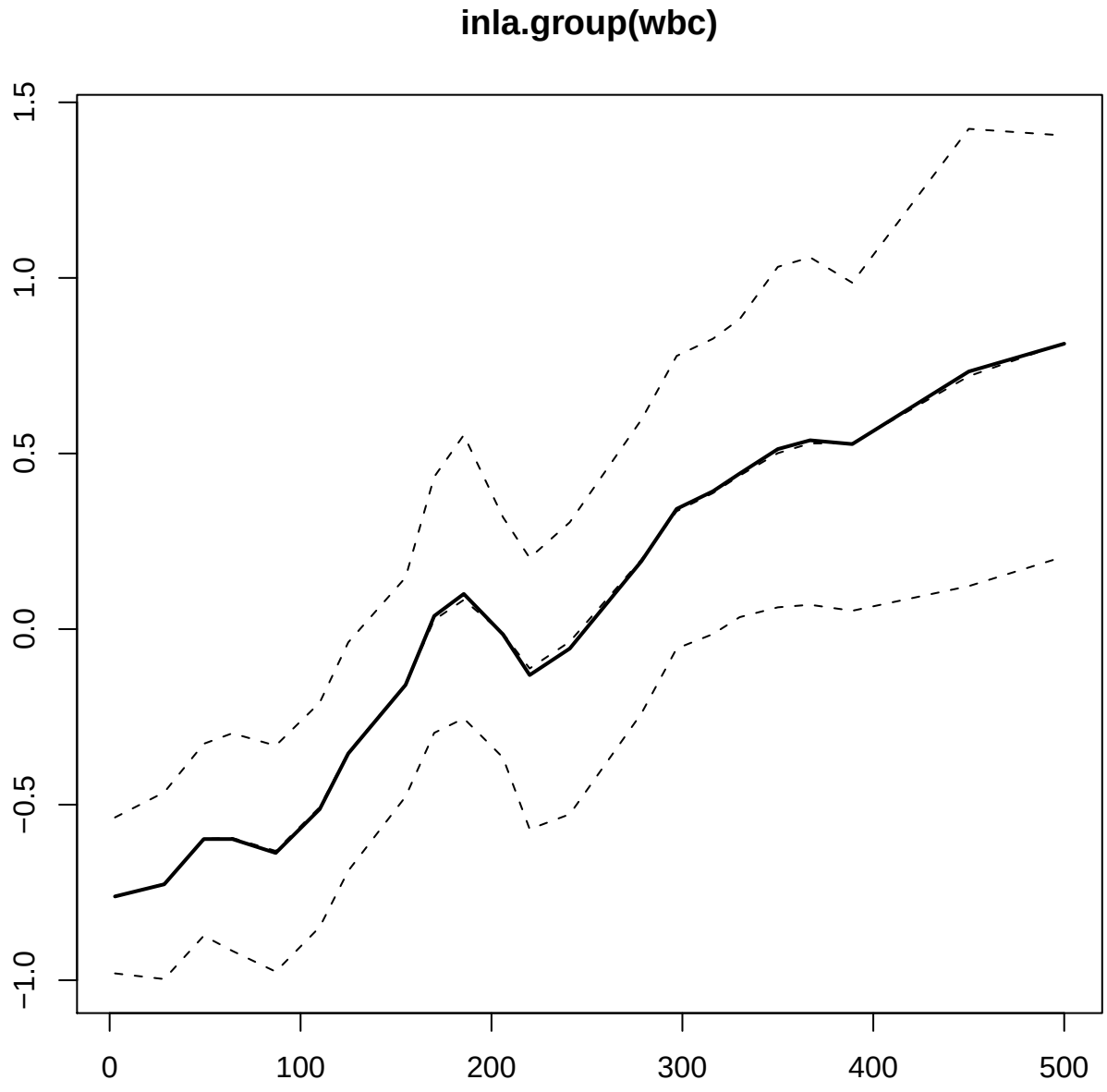


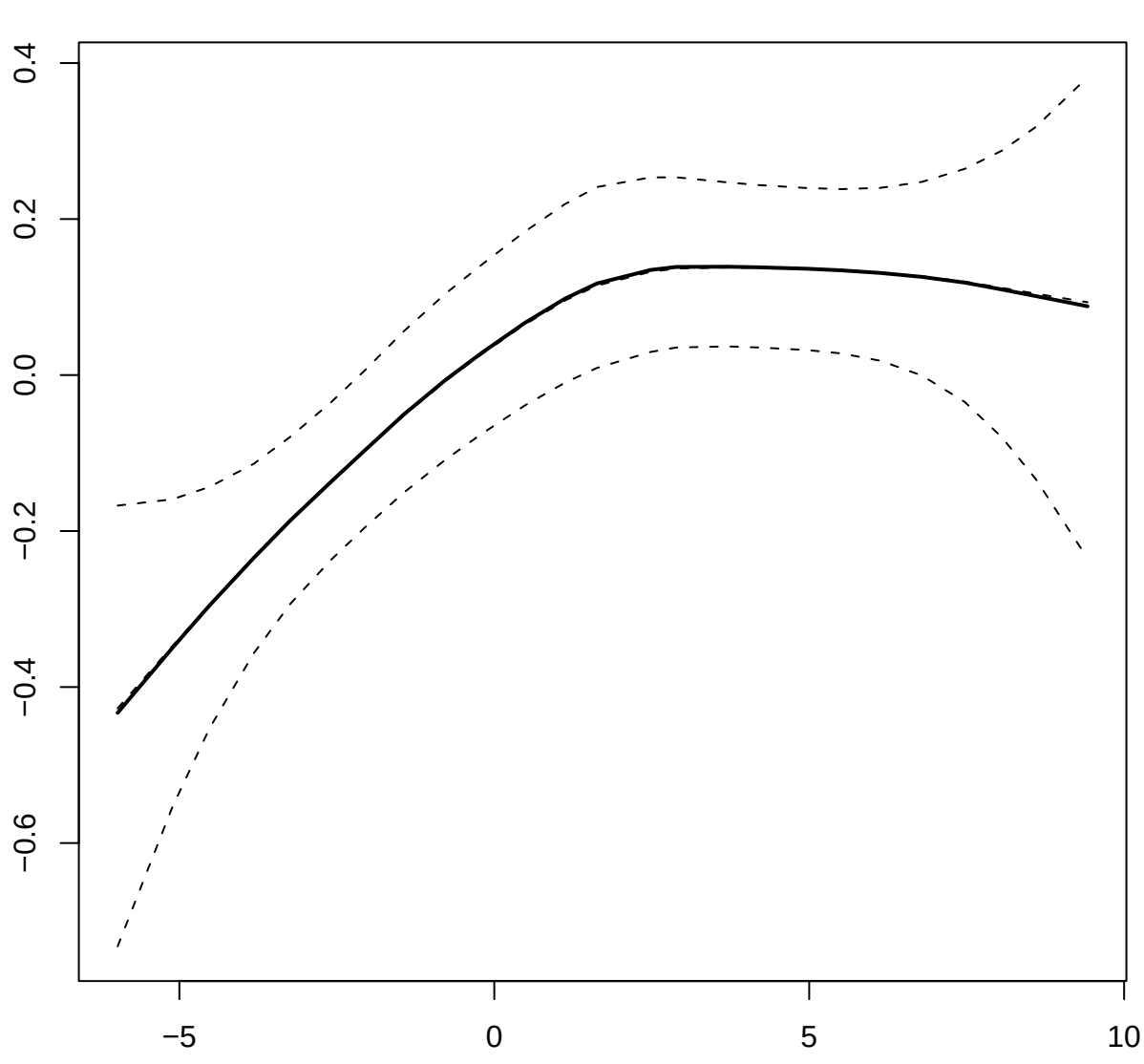
Mean = 0.077 SD = 0.069

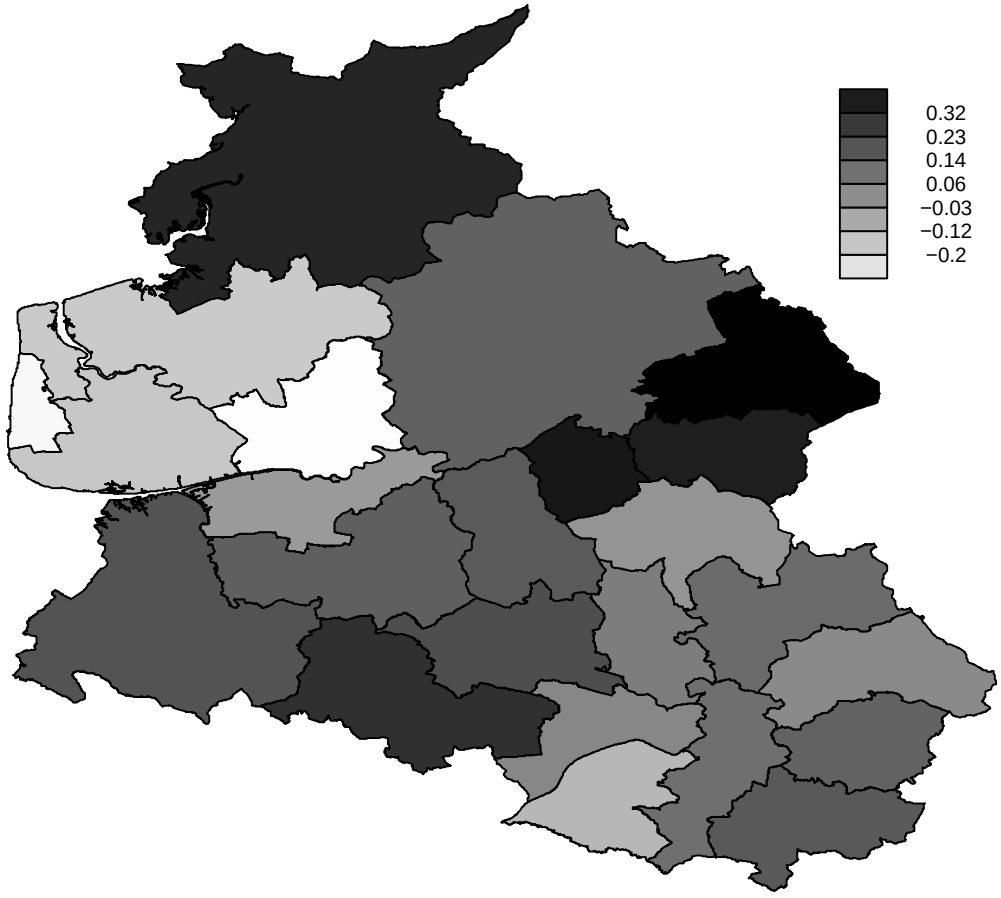
PostDens [age]



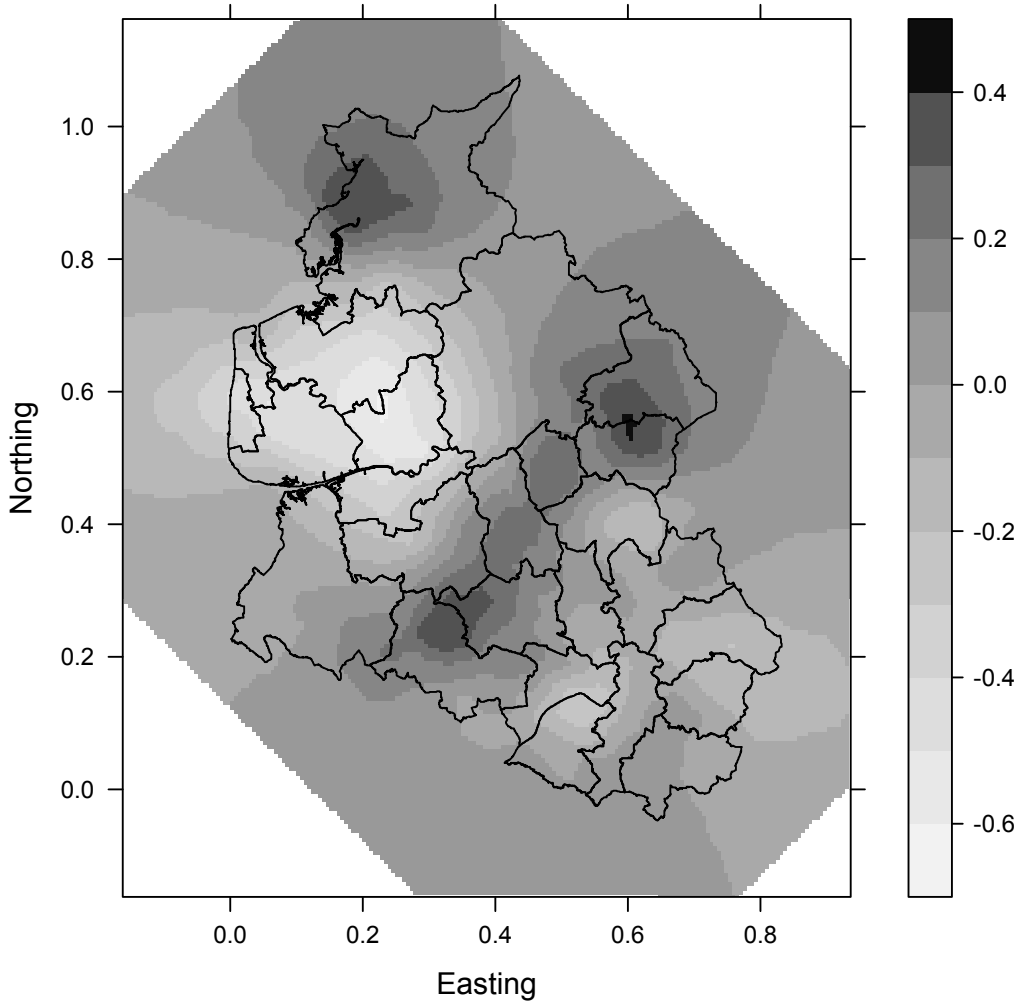
Mean = 0.036 SD = 0.002


inla.group(wbc)
1.5
1.0
0.5
0.0
−0.5
−1.0
0
100
200
300
400
500
PostMean 0.025% 0.5% 0.975%


inla.group(tpi)
0.4
0.2
0.0
−0.2
−0.4
−0.6
−5
0
5
10
PostMean 0.025% 0.5% 0.975%


0.32
0.23
0.14
0.06
−0.03
−0.12
−0.2

Continuous spatial effect



SPDE models

We call spatial Markov models defined on a mesh *SPDE models*.

SPDE* models have 3 parts

- A mesh
- A range parameter κ
- A precision parameter τ

SPDE=Stochastic Partial Differential Equation

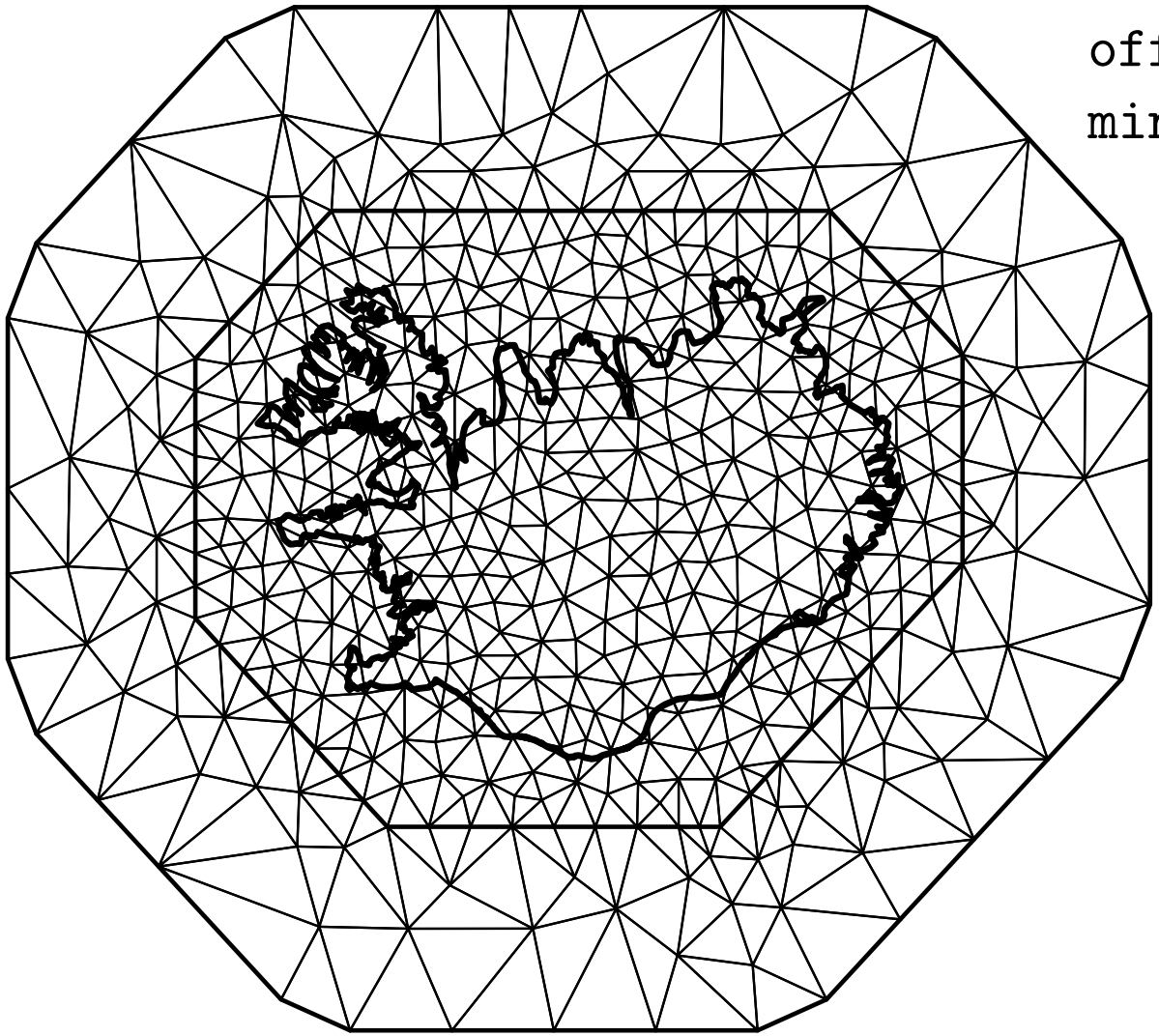
The mesh

Meshes can be created using two different functions:

- `inla.mesh.create`: The workhorse function. An interface to the meshing code written by Finn Lindgren.
- `inla.mesh.2d`: A slightly more user friendly interface for creating practical meshes (we will focus on this one).

Typical use

```
mesh <- inla.mesh.create.helper(points.domain=iceland_ISN,
                                max.edge=c(40,800),
                                offset=c(50,150),
                                min.angle=25)
```



inla.mesh.create.helper

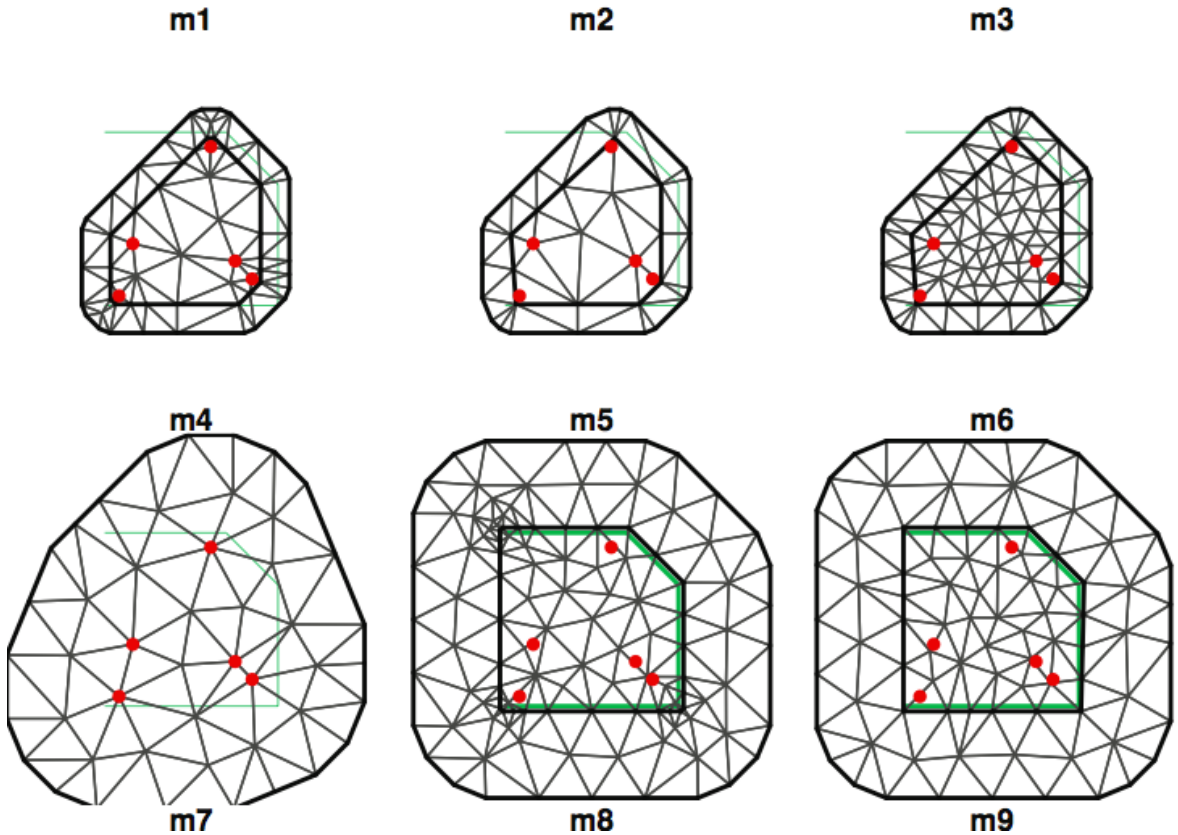
```
inla.mesh.create.helper(loc = NULL,
                        loc = NULL,
                    loc.domain = NULL,
                    offset = NULL,
                    n = NULL,
                      boundary = NULL,
                    interior = NULL,
                    max.edge,
                    min.angle = NULL,
                    cutoff = 0,
                    plot.delay = NULL)
```

This function contains a mesh with two regions: the interior mesh, which is where the action happens; and the exterior mesh, which is designed to alleviate the boundary effects.

Arguments

- `loc`: Points to be included as vertices in the triangulation.
- `loc.domain`: Points not in the mesh, but that are used to define the internal mesh section (taken as the convex hull of these points).
- `offset=c(a,b)`: Distance from the points to the inner (outer) boundary. Negative numbers = relative distance.
- `boundary`: Prescribed boundary. (inla.mesh.segment type)
- `max.edge = c(a,b)`: Maximum triangle edge length in the inner (outer) segment.
- `min.angle = c(a,b)`: Minimum angle for the inner and outer segments (bigger angles are better, but harder to make)
- `cutoff`: Minimum distance between two distinct points.

Good and bad meshes



Between the mesh and the data

- So a good mesh probably doesn't have vertices at the data locations
- This means we need to have a way to get between values of the field at the vertices and the value of the field at the data points
- The trick is that the SPDE model is *linear* on the triangles, so the value of the field at any point is a weighted sum of the vertices of the triangle the point is in.
- In maths speak, we are observing $\boldsymbol{Ax}$ rather than $\boldsymbol{x}$
- We call $\boldsymbol{A}$ the "A-matrix" or the "observation matrix"

Making observation matrices in INLA

When the observations don't occur at mesh points, we need some way to map between the latent field and the observation process.

- `inla.spde.make.A` constructs the matrix $A_{ij} = \phi_j(s_i)$ that maps a field defined on the mesh to the observation locations s_i.
- The function will also automatically deal with space-time models and replicates.
- A related function (inla.mesh.projector) builds an A-matrix for projecting onto a lattice. This is useful for plotting.

The inla.spde.make.A call

```
inla.spde.make.A(mesh = NULL,
                 loc = NULL,
                 index = NULL,
                 group = NULL,
                 repl = 1L,
                 n.mesh = NULL,
                 n.group = max(group),
                 n.repl = max(repl),
                 group.mesh = NULL,
                 group.method = c("nearest", "S0", "S1"),
                 weights = NULL)
```

- The first two arguments are needed.
- group is needed to build space-time models
- The other arguments are fairly advanced!

Other mesh commands

- `inla.mesh.segment`: Constructs an object that can be given to `inla.mesh.create` as a boundary or interior segment
- `inla.mesh.boundary`: Extracts a boundary segment from a mesh.
- `inla.mesh.project` and `inla.mesh.projector`: Projects results from a mesh to a lattice. Useful for plotting.
- `inla.mesh.basis`: Constructs a B-spline basis of a given degree on a mesh.
- `inla.mesh.query`: Extracts information about the topology of the mesh (advanced!)

Constructing SPDE models

For historical reasons there are two different SPDE classes (`spde1` and `spde2`)

- `spde1` is the "classic" SPDE model!
- The `spde2` class is more flexible and defines non-stationarity in a more natural way.
- The primary difference between the two models is in the prior specification.
- At some point there will probably be an `spde3` class: We are interested in backwards-compatability!
- For "stationary" models, these are fairly much the same (up to prior specification)

The spde1 call

```
inla.spde.create(mesh,
                 model = c("matern", "imatern","matern.osc"),
                 param = NULL)
```

- "imatern" is the intrinsic model ($\kappa^2 = 0$).
- "matern.osc" is an oscillating Matérn model.
- param is a list that contains alpha (1 or 2) and stuff about non-stationarity.

Why does spde2 exist?

The problem with the spde1 comes when specifying non-stationarity.

- Suppose we want to model $\tau(s) = \sum_{i=1}^{k} \theta_i^\tau b_i(s)$ for some basis functions $\{b_i(s)\}$. (Similar for $\kappa^2(s)$)
- The spde1 model put i.i.d. log-normal priors on the θ_i.
- This is not a good idea: what if we want a smooth effect—should have a spline prior...
- We also penalise the (log) variance directly:

$$\log(\sigma^2) = \text{const.} - 2\log(\kappa) - 2\log(\tau)$$

- spde2 fixes this by putting a multivariate normal prior on

$$\log(\boldsymbol{\tau}) = \boldsymbol{B}^\tau \boldsymbol{\theta}, \qquad \log(\boldsymbol{\kappa}^2) = \boldsymbol{B}^\kappa \boldsymbol{\theta}$$

with the same $\boldsymbol{\theta} \sim N(\boldsymbol{\mu}, \boldsymbol{Q}^{-1})$.

Why does spde2 exist?

The problem with the spde1 comes when specifying non-stationarity.

- Suppose we want to model $\tau(s) = \sum_{i=1}^{k} \theta_i^\tau b_i(s)$ for some basis functions $\{b_i(s)\}$. (Similar for $\kappa^2(s)$)
- The spde1 model put i.i.d. log-normal priors on the θ_i.
- This is not a good idea: what if we want a smooth effect—should have a spline prior...
- We also penalise the (log) variance directly:

$$\log(\sigma^2) = \text{const.} - 2\log(\kappa) - 2\log(\tau)$$

- spde2 fixes this by putting a multivariate normal prior on

$$\log(\boldsymbol{\tau}) = \boldsymbol{B}^\tau \boldsymbol{\theta}, \qquad \log(\boldsymbol{\kappa}^2) = \boldsymbol{B}^\kappa \boldsymbol{\theta}$$

with the same $\boldsymbol{\theta} \sim N(\boldsymbol{\mu}, \boldsymbol{Q}^{-1})$.

Why does spde2 exist?

The problem with the spde1 comes when specifying non-stationarity.

- Suppose we want to model $\tau(s) = \sum_{i=1}^{k} \theta_i^\tau b_i(s)$ for some basis functions $\{b_i(s)\}$. (Similar for $\kappa^2(s)$)
- The spde1 model put i.i.d. log-normal priors on the θ_i.
- This is not a good idea: what if we want a smooth effect—should have a spline prior...
- We also penalise the (log) variance directly:

$$\log(\sigma^2) = \text{const.} - 2\log(\kappa) - 2\log(\tau)$$

- spde2 fixes this by putting a multivariate normal prior on

$$\log(\boldsymbol{\tau}) = \boldsymbol{B}^\tau \boldsymbol{\theta}, \qquad \log(\boldsymbol{\kappa}^2) = \boldsymbol{B}^\kappa \boldsymbol{\theta}$$

with the same $\boldsymbol{\theta} \sim N(\boldsymbol{\mu}, \boldsymbol{Q}^{-1})$.

Why does spde2 exist?

The problem with the spde1 comes when specifying non-stationarity.

- Suppose we want to model $\tau(s) = \sum_{i=1}^{k} \theta_i^\tau b_i(s)$ for some basis functions $\{b_i(s)\}$. (Similar for $\kappa^2(s)$)
- The spde1 model put i.i.d. log-normal priors on the θ_i.
- This is not a good idea: what if we want a smooth effect—should have a spline prior...
- We also penalise the (log) variance directly:

$$\log(\sigma^2) = \text{const.} - 2\log(\kappa) - 2\log(\tau)$$

- spde2 fixes this by putting a multivariate normal prior on

$$\log(\boldsymbol{\tau}) = \boldsymbol{B}^\tau \boldsymbol{\theta}, \qquad \log(\boldsymbol{\kappa}^2) = \boldsymbol{B}^\kappa \boldsymbol{\theta}$$

with the same $\boldsymbol{\theta} \sim N(\boldsymbol{\mu}, \boldsymbol{Q}^{-1})$.

Why does spde2 exist?

The problem with the spde1 comes when specifying non-stationarity.

- Suppose we want to model $\tau(s) = \sum_{i=1}^{k} \theta_i^\tau b_i(s)$ for some basis functions $\{b_i(s)\}$. (Similar for $\kappa^2(s)$)
- The spde1 model put i.i.d. log-normal priors on the θ_i.
- This is not a good idea: what if we want a smooth effect—should have a spline prior...
- We also penalise the (log) variance directly:

$$\log(\sigma^2) = \text{const.} - 2\log(\kappa) - 2\log(\tau)$$

- spde2 fixes this by putting a multivariate normal prior on

$$\log(\boldsymbol{\tau}) = \boldsymbol{B}^\tau \boldsymbol{\theta}, \qquad \log(\boldsymbol{\kappa}^2) = \boldsymbol{B}^\kappa \boldsymbol{\theta}$$

with the same $\boldsymbol{\theta} \sim N(\boldsymbol{\mu}, \boldsymbol{Q}^{-1})$.

The spde2 call

```
inla.spde2.matern(mesh,
                  alpha = 2,
                  B.tau = matrix(c(0,1,0),1,3),
                  B.kappa = matrix(c(0,0,1),1,3),
                  prior.variance.nominal = 1,
                  prior.range.nominal = NULL,
                  prior.tau = NULL,
                  prior.kappa = NULL,
                  theta.prior.mean = NULL,
                  theta.prior.prec = NULL,
                fractional.method = c("parsimonious", "null"))
```

Arguments

- `mesh`: An inla.mesh object. (Necessary)
- `alpha` =2: The smoothness. Exact fields if it's an integer, approximate fields for non-integer α
- `B.tau`: The matrix $\boldsymbol{B}^\tau$ use to define non-stationary $\tau(s)$
- `B.kappa`: As above, but for $\kappa^2(s)$
- `prior.variance.nominal`, `prior.range.nominal`: Helps the automatic prior know the scale of the variance and the range
- `prior.tau`, `prior.kappa`: Prior specification for τ and κ^2. (not often used)
- `theta.prior.mean`, `theta.prior.prec`: Mean vector and precision matrix for $\boldsymbol{\theta}$ prior.
- `fractional.method`: Method for constructing fractional α approximation

Arguments

- `mesh`: An inla.mesh object. (Necessary)
- `alpha` =2: The smoothness. Exact fields if it's an integer, approximate fields for non-integer α
- `B.tau`: The matrix $\boldsymbol{B}^{\tau}$ use to define non-stationary $\tau(s)$
- `B.kappa`: As above, but for $\kappa^2(s)$
- `prior.variance.nominal`, `prior.range.nominal`: Helps the automatic prior know the scale of the variance and the range
- `prior.tau`, `prior.kappa`: Prior specification for τ and κ^2. (not often used)
- `theta.prior.mean`, `theta.prior.prec`: Mean vector and precision matrix for $\boldsymbol{\theta}$ prior.
- `fractional.method`: Method for constructing fractional α approximation

Arguments

- `mesh`: An inla.mesh object. (Necessary)
- `alpha` =2: The smoothness. Exact fields if it's an integer, approximate fields for non-integer α
- `B.tau`: The matrix $\boldsymbol{B}^{\tau}$ use to define non-stationary $\tau(s)$
- `B.kappa`: As above, but for $\kappa^2(s)$
- `prior.variance.nominal`, `prior.range.nominal`: Helps the automatic prior know the scale of the variance and the range
- `prior.tau`, `prior.kappa`: Prior specification for τ and κ^2. (not often used)
- `theta.prior.mean`, `theta.prior.prec`: Mean vector and precision matrix for $\boldsymbol{\theta}$ prior.
- `fractional.method`: Method for constructing fractional α approximation

Arguments

- `mesh`: An inla.mesh object. (Necessary)
- `alpha` =2: The smoothness. Exact fields if it's an integer, approximate fields for non-integer α
- `B.tau`: The matrix $\boldsymbol{B}^\tau$ use to define non-stationary $\tau(s)$
- `B.kappa`: As above, but for $\kappa^2(s)$
- `prior.variance.nominal`, `prior.range.nominal`: Helps the automatic prior know the scale of the variance and the range
- `prior.tau`, `prior.kappa`: Prior specification for τ and κ^2. (not often used)
- `theta.prior.mean`, `theta.prior.prec`: Mean vector and precision matrix for $\boldsymbol{\theta}$ prior.
- `fractional.method`: Method for constructing fractional α approximation

Arguments

- `mesh`: An inla.mesh object. (Necessary)
- `alpha` =2: The smoothness. Exact fields if it's an integer, approximate fields for non-integer α
- `B.tau`: The matrix $\boldsymbol{B}^{\tau}$ use to define non-stationary $\tau(s)$
- `B.kappa`: As above, but for $\kappa^2(s)$
- `prior.variance.nominal`, `prior.range.nominal`: Helps the automatic prior know the scale of the variance and the range
- `prior.tau`, `prior.kappa`: Prior specification for τ and κ^2. (not often used)
- `theta.prior.mean`, `theta.prior.prec`: Mean vector and precision matrix for $\boldsymbol{\theta}$ prior.
- `fractional.method`: Method for constructing fractional α approximation

Arguments

- `mesh`: An inla.mesh object. (Necessary)
- `alpha` =2: The smoothness. Exact fields if it's an integer, approximate fields for non-integer α
- `B.tau`: The matrix $\boldsymbol{B}^\tau$ use to define non-stationary $\tau(s)$
- `B.kappa`: As above, but for $\kappa^2(s)$
- `prior.variance.nominal`, `prior.range.nominal`: Helps the automatic prior know the scale of the variance and the range
- `prior.tau`, `prior.kappa`: Prior specification for τ and κ^2. (not often used)
- `theta.prior.mean`, `theta.prior.prec`: Mean vector and precision matrix for $\boldsymbol{\theta}$ prior.
- `fractional.method`: Method for constructing fractional α approximation

Arguments

- `mesh`: An inla.mesh object. (Necessary)
- `alpha` =2: The smoothness. Exact fields if it's an integer, approximate fields for non-integer α
- `B.tau`: The matrix $\boldsymbol{B}^{\tau}$ use to define non-stationary $\tau(s)$
- `B.kappa`: As above, but for $\kappa^2(s)$
- `prior.variance.nominal`, `prior.range.nominal`: Helps the automatic prior know the scale of the variance and the range
- `prior.tau`, `prior.kappa`: Prior specification for τ and κ^2. (not often used)
- `theta.prior.mean`, `theta.prior.prec`: Mean vector and precision matrix for $\boldsymbol{\theta}$ prior.
- `fractional.method`: Method for constructing fractional α approximation

Arguments

- `mesh`: An inla.mesh object. (Necessary)
- `alpha` =2: The smoothness. Exact fields if it's an integer, approximate fields for non-integer α
- `B.tau`: The matrix $\boldsymbol{B}^{\tau}$ use to define non-stationary $\tau(s)$
- `B.kappa`: As above, but for $\kappa^2(s)$
- `prior.variance.nominal`, `prior.range.nominal`: Helps the automatic prior know the scale of the variance and the range
- `prior.tau`, `prior.kappa`: Prior specification for τ and κ^2. (not often used)
- `theta.prior.mean`, `theta.prior.prec`: Mean vector and precision matrix for $\boldsymbol{\theta}$ prior.
- `fractional.method`: Method for constructing fractional α approximation

A few more useful commands

- `inla.spde.precision(spde,tau=...,kappa2=...)`—computes precision matrix. Less straightforward for `spde2` models
- `inla.qsample(n,Q,...)`—Computes a sample and various other quantities needed for MCMC for precision matrix `Q`
- `inla.qreordering`—Computes a fill-in reducing reordering.
- `inla.qsovle`—Solve a linear system
- `inla.qinv(Q)`—Calculates the elements of the inverse corresponding to the non-zero elements of $\boldsymbol{Q}$. Needed for computing derivatives of Gaussian likelihoods.

A few more useful commands

- `inla.spde.precision(spde,tau=...,kappa2=...)`—computes precision matrix. Less straightforward for `spde2` models
- `inla.qsample(n,Q,...)`—Computes a sample and various other quantities needed for MCMC for precision matrix `Q`
- `inla.qreordering`—Computes a fill-in reducing reordering.
- `inla.qsovle`—Solve a linear system
- `inla.qinv(Q)`—Calculates the elements of the inverse corresponding to the non-zero elements of $\boldsymbol{Q}$. Needed for computing derivatives of Gaussian likelihoods.

A few more useful commands

- `inla.spde.precision(spde,tau=...,kappa2=...)`—computes precision matrix. Less straightforward for `spde2` models
- `inla.qsample(n,Q,...)`—Computes a sample and various other quantities needed for MCMC for precision matrix `Q`
- `inla.qreordering`—Computes a fill-in reducing reordering.
- `inla.qsovle`—Solve a linear system
- `inla.qinv(Q)`—Calculates the elements of the inverse corresponding to the non-zero elements of $\boldsymbol{Q}$. Needed for computing derivatives of Gaussian likelihoods.

A few more useful commands

- `inla.spde.precision(spde,tau=...,kappa2=...)`—computes precision matrix. Less straightforward for `spde2` models
- `inla.qsample(n,Q,...)`—Computes a sample and various other quantities needed for MCMC for precision matrix `Q`
- `inla.qreordering`—Computes a fill-in reducing reordering.
- `inla.qsovle`—Solve a linear system
- `inla.qinv(Q)`—Calculates the elements of the inverse corresponding to the non-zero elements of $\boldsymbol{Q}$. Needed for computing derivatives of Gaussian likelihoods.

A few more useful commands

- `inla.spde.precision(spde,tau=...,kappa2=...)`—computes precision matrix. Less straightforward for `spde2` models
- `inla.qsample(n,Q,...)`—Computes a sample and various other quantities needed for MCMC for precision matrix `Q`
- `inla.qreordering`—Computes a fill-in reducing reordering.
- `inla.qsovle`—Solve a linear system
- `inla.qinv(Q)`—Calculates the elements of the inverse corresponding to the non-zero elements of $\boldsymbol{Q}$. Needed for computing derivatives of Gaussian likelihoods.

Outline

Useful features

- replicate and group
- more than one "family"
- copy
- linear combinations
- $\boldsymbol{A}$ matrix in the linear predictor
- values
- remote computing

Feature: replicate

"replicate" generates iid replicates from the same f()-model with the same hyperparameters.

If $\boldsymbol{x} \mid \boldsymbol{\theta} \sim$ AR(1), then nrep=3, makes

$$\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$$

with mutually independent $\boldsymbol{x}_i$'s from AR(1) with the same $\boldsymbol{\theta}$
Arguments

```
f(..., replicate = r [, nrep = nr ])
```

where replicate are integers $1, 2, \ldots,$ etc

Example: replicate

NAs in INLA

What do NAs do?

- In the covariates, an NA is treated as a zero.
- In the random effect, NAs indicate that the effect does not contribute to the likelihood
- In the data, an NA indicates a location for prediction.

Feature: copy

This feature fixes a limitation in the formula-formulation of the model

The model

```
formula = y ~ f(i, ...) + ...
```

Only allow ONE element from each sub-model, to contribute to the linear predictor for each observation.

Sometimes/Often this is not sufficient.

Feature: copy

This feature fixes a limitation in the formula-formulation of the model

The model

```
formula = y ~ f(i, ...) + ...
```

Only allow ONE element from each sub-model, to contribute to the linear predictor for each observation.

Sometimes/Often this is not sufficient.

Feature: copy

This feature fixes a limitation in the formula-formulation of the model

The model

```
formula = y ~ f(i, ...) + ...
```

Only allow ONE element from each sub-model, to contribute to the linear predictor for each observation.

Sometimes/Often this is not sufficient.

Feature: copy

Suppose

$$\eta_i = u_i + u_{i+1} + ...$$

Then we can code this as

```
formula = y ~ f(i, model="iid") +
              f(i.plus, copy="i") + ...
```

- The copy-feature, creates internally an additional sub-model which is ϵ-close to the target
- Many copies allowed, and copies of copies

Feature: copy

Suppose

$$\eta_i = u_i + \beta u_{i+1} + ...$$

Then we can code this as

```
formula = y ~ f(i, model="iid") +
              f(i.plus, copy="i",
                hyper = list(
                   beta = list(fixed = FALSE))) + ...
```

Feature: copy

Suppose that

$$\eta_i = a_i + b_i z_i + ...$$

where

$$(a_i, b_i) \overset{\text{iid}}{\sim} \mathcal{N}_2(\mathbf{0}, \mathbf{\Sigma})$$

Multiple likelihoods

In many situations, you need to combine data from different sources and need to be able to handle multiple likelihoods.

Examples:

- Joint modelling of longitudinal and event time data (Guo and Carlin, 2004)
- Preferential sampling (Diggle et al, 2010)
- "Marked" point processes
- Animal breeding modelling with multiple traits
- Combining data from multiple experiments

Multiple likelihoods

In many situations, you need to combine data from different sources and need to be able to handle multiple likelihoods.

Examples:

- Joint modelling of longitudinal and event time data (Guo and Carlin, 2004)
- Preferential sampling (Diggle et al, 2010)
- "Marked" point processes
- Animal breeding modelling with multiple traits
- Combining data from multiple experiments

How to do this in INLA

- Make response `y` a *matrix* rather than a vector.

```
> Y = matrix(NA, N, 2)
> Y[1:n, 1] = y[1:n]
> Y[1:n + n, 2] = y[(n + 1):(2 * n)]
```

- `NA`s are used to select components in the formula

```
> cov1 = c(cov, rep(NA,n))
```

Outline

Back to covariates

Consider a model with a linear predictor that looks like

$$\eta_i = \ldots + \beta c(s_i) + \ldots$$

where c is an unknown spatial covariate.

- c_i is unknown, but we have some measurements $\{c'_j\}$ at points $\{s'_j\}$
- We can model the true covariate field as above

$$c'_j | c(\cdot) = c(s'_j) + \epsilon_j$$
$$c(\cdot) \sim \text{SPDE model}$$

Joint modelling of the covariate

We can then fit these models *at the same time*!

Likelihood:

$$y_i \mid \eta_i \sim \texttt{Any INLA likelihood with latent field } \eta$$
$$c'_j | c(\cdot) \sim N(\xi_j, \tau_c^{-1})$$

Latent field:

$$\eta_i = \ldots + \beta c(s_i) + \ldots$$
$$\xi_j = c(s'_j)$$

- We have *two likelihoods* (data and covariate)
- We use the covariate field $c(s)$ twice $\longrightarrow$ *copy*

Setting up the likelihood

We begin by putting the observations and the observed covariates *together as data*

```
> Y = matrix(NA, N, 2)
> Y[1:n, 1] = y
> Y[(n+1):(2*n), 2] = obs_covariate
```

Setting up the formula

We need to set up the formula carefully to separate out the two things. The trick is NAs in indices

```
> covariate_first_lik = c(1:spde$n.spde,
                                    rep(NA, spde$n.spde))
> covariate_second_lik = c(rep(NA, spde$n.spde),
                                          1:spde$n.spde)
```

The formula is then

```
> formula = Y ~ ...+ f(covariate_first_lik, model=spde)
      + f(covariate_second_like, copy=covariate_first_lik) +
```

The INLA call

Finally, we need to make an `inla` call for this model.

```
> result = inla(formula, family = c("_____", "gaussian"),
      data = list(Y=Y,
      covariate_first_lik=covariate_first_lik,
      covariate_second_lik=covariate_second_lik),
      verbose=TRUE)
```

where `_____` is the data likelihood.

This model in practice

- Joint modelling the covariate adds 3 hyperparameters (range, precision, noise precision)
- This can be done any type of data (eg point patterns)
- If there is *misalignment*, it can get tricky
- In this case, you need A-matrices

Organising data, latent fields and $\boldsymbol{A}$ matrices

Real life is hard!

- In complicated models, we will have multiple sources of data occurring in different places with different likelihood.
- The latent field may also be composed of sections defined at different resolutions (grid for a spatial covariate, mesh for random field, etc).
- So we need a function that takes these components and chains them together in a way that makes sense.
- (You can "roll your own" here, but I *really* don't recommend it!)

We are rescued by `inla.stack`!

The inla.stack call

```
stack = inla.stack(data = list(...),
                   A = list(...),
                   effects = list(...),
                   tag = NULL, ...)
```

- The trick here is lists!
- The first element of the `effects` list is mapped to the first element of the `data` list by the first element for the `A` list.
- Slightly more tricky when there are replicates and grouping (time!)
- The functions `inla.stack.data(stack)` and `inla.stack.A(stack)` are used to extract the data.frame and the A-matrix for use in the `inla` call.