Continuous Domain Spatial Models in R-INLA

Finn Lindgren, January 16, 2013
f.lindgren@bath.ac.uk

This is an updated version of a text originally published in *The ISBA Bulletin* Vol.19 No.4, December 2012
The differences concern minor code changes and corrections

Traditionally, Markov models in image analysis and spatial statistics have been largely confined to discrete spatial domains, such as lattices and regional adjacency graphs. However, as discussed in [3], one can express a large class of random field models as solutions to continuous domain stochastic partial differential equations (SPDEs), and write down explicit links between the parameters of each SPDE and the elements of precision matrices for weights in a discrete basis function representation. As shown by Whittle in 1963, such models include those with Matérn covariance functions, which are ubiquitous in traditional spatial statistics, but in contrast to covariance based models it is far easier to introduce non-stationarity into the SPDE models. This is because the differential operators act locally, similarly to local increments in Gibbs-specifications of Markov models, and only mild regularity conditions are required. The practical significance of this is that we can merge the classical Gaussian random fields with methods based on the Markov property, providing continuous domain models that are computationally efficient, and where the parameters can be specified locally without having to worry about positive definiteness of covariance functions.

In this brief note, I present the basic ingredients of the link between continuous domains and Markov models, and show how to perform Bayesian inference for the simplest of these models, using the `R-INLA` software package (`http://www.r-inla.org`). Special emphasis is placed on the abstractions necessary to simplify the practical bookkeeping for the user of the software.

Think continuous

When building and using hierarchical models with latent random fields it is important to remember that the latent fields often represent real-world phenomena that exist independently of whether they are observed in a given location or not. Thus, we are not building models solely for *discretely observed data*, but for approximations of *entire processes* defined on continuous domains. For a spatial field $x(\mathbf{s})$, while the data likelihood typically depends only on the values at a finite set of locations, $\{\mathbf{s}_1, \ldots, \mathbf{s}_m\}$, the model itself defines the joint behaviour for all locations, typically $\mathbf{s} \in \mathbb{R}^2$ or $\mathbf{s} \in \mathbb{S}^2$ (a sphere/globe). In the case of lattice data, the discretisation typically happens in the observation stage, such as integration over grid boxes (e.g. photon collection in a camera sensor). Often, this is approximated by pointwise evaluation, but there is nothing apart from computational challenges preventing other observation models.

As discussed in the introduction, an alternative to traditional covariance based modelling is to use SPDEs, but carry out the practical computations using Gaussian Markov random field (GMRF) representations. This is done by approximating the full set of spatial random functions with weighted sums of simple basis functions, which allows us to hold on to the continuous interpretation of space, while the

computational algorithms only see discrete structures with Markov properties. Beyond the main paper [3], this is further discussed in [5] and [6].

The simplest model for $x(\mathbf{s})$ currently implemented in `R-INLA` is the SPDE/GMRF version of the stationary Matérn family,

$$(\kappa^2 - \Delta)^{\alpha/2}(\tau x(\mathbf{s})) = \mathcal{E}(\mathbf{s}), \quad \mathbf{s} \in \mathbb{R}^2,$$

where Δ is the Laplacian, κ is the spatial scale parameter, α controls the smoothness of the realisations, and τ controls the variance. The right-hand side of the equation, $\mathcal{E}(\mathbf{s})$ is spatial white noise process. The link to the Matérn smoothness ν and variance σ^2 is $\nu = \alpha - d/2$ and $\sigma^2 = \Gamma(\nu)(\Gamma(\alpha)(4\pi)^{d/2}\kappa^{2\nu}\tau^2)^{-1}$, where d is the spatial dimension. From this we can identify the exponential covariance with $\nu = 1/2$ and $\alpha = 3/2$, and note that fields with $\alpha \leq 1$ give $\nu \leq 0$ and that such fields have no point-wise interpretation (but do have well-defined integration properties). From spectral theory one can show that integer values for α gives continuous domain Markov fields, and these are the easiest for which to provide discrete basis representations. In `R-INLA`, the default value is $\alpha = 2$, but $0 \leq \alpha < 2$ are also available, though not as extensively tested (for the non-integer α values the approximation method introduced in the authors' discussion response in [3] is used).

The models discussed in [3] and implemented in `R-INLA` are built on a basis representation

$$x(\mathbf{s}) = \sum_{i=1}^{n} \psi_i(\mathbf{s}) x_i,$$

where the joint distribution of $\mathbf{x} = \{x_1, \ldots, x_n\}$ is chosen so that the distribution of the functions $x(\mathbf{s})$ approximates the distribution of solutions to the SPDE. To obtain a Markov structure, and to preserve it when conditioning on observations, we use basis function with local support. The construction is done by projecting the SPDE onto the basis representation in what is essentially a Finite Element method. To allow easy and explicit evaluation, we use piece-wise linear basis functions defined by a triangulation of the domain of interest. This yields a diagonal matrix $\mathbf{C}$ and a sparse matrix $\mathbf{G}$ such that the appropriate precision matrix for the weights is given by

$$\mathbf{Q} = \tau^2(\kappa^4\mathbf{C} + 2\kappa^2\mathbf{G} + \mathbf{G}\mathbf{C}^{-1}\mathbf{G})$$

for the default case $\alpha = 2$, so that the elements of $\mathbf{Q}$ have explicit expressions as functions of κ and τ. The default internal representation of the parameters in the model interface we will use here is $\log(\tau) = \theta_1$ and $\log(\kappa) = \theta_2$, where θ_1 and θ_2 have a joint normal prior distribution (by default independent).

There is a vast range of possible extensions to the simple model described here, including non-stationary versions (see [3] and [1] for examples). For currently implemented models, the recent paper [2] is a case-study using a straightforward space-time version including the full `R` code. A non-stationary extension by defining spatially varying models for $\kappa(\mathbf{s})$ and $\tau(\mathbf{s})$ is also implemented, and will be mentioned briefly later on.

While the full theory behind fractional stochastic partial differential equations is challenging at best, the major challenge when designing a general software package for practical use of these models is rather that of bookkeeping. To solve this, a bit of abstraction is needed to avoid cluttering the interface with details of internal storage. Thus, instead of visibly keeping track of mappings between triangle mesh node indices and data locations, one can use

sparse matrices to encode these relationships, and provide wrapper functions to manipulate these matrices and associated index and covariate vectors.

The first step is to create the triangulated mesh on top of which the SPDE/GMRF representation is to be built. The example here illustrates a common usage case, which is to have semi-randomly scattered observation locations in a region of space such that there is no physical boundary, just an limited observation region. When dealing with only covariances between data points, this distinction is often unimportant, but here it becomes a possibly vital part of the model, since the SPDE will exhibit boundary effects. In the `R-INLA` implementation, Neumann boundaries are used, which increases the variance near the boundary. If we intend to model a stationary field across the entire domain of observations, we must therefore extend the model domain far enough so that the boundary effects don't influence the observations. However, note that the reverse is also true: if there *is* a physical boundary, the boundary effects may actually be desirable. The helper function `inla.mesh.create.helper()` allows us to create a mesh with small triangles in the domain of interest, and use larger triangles in the extension used to avoid boundary effects. This minimises the extra computational work needed due to the extension.

```
m = 100
points = matrix(runif(m*2),m,2)
mesh = inla.mesh.create.helper(
  points=points,
  cutoff=0.05,
  offset=c(0.1,0.4),
  max.edge=c(0.05,0.5) )
plot(mesh)
points(points[,1],points[,2])
```

The `cutoff` parameter is used to avoid building many small triangles around clustered input locations, `offset` specifies the size of the inner and outer extensions around the data locations, and `max.edge` specifies the maximum allowed triangle edge lengths in the inner domain and in the outer extension. The overall effect of the triangulation construction is that, if desired, one can have smaller triangles, and hence higher accuracy of the field representation, where the observation locations are dense, larger triangles where data is more sparse (and hence provides less detailed information), and large triangles where there is no data and spending computational resources would be wasteful. However, note that there is neither any guarantee nor any requirement that the observation locations are included as nodes in the mesh. If one so desires, the mesh can be designed from different principles, such as lattice points with no relation to the precise measurement locations. This emphasises the decoupling between the continuous domain of the field model and the discrete data locations.

Defining an SPDE model object can now be as simple as

```
spde=inla.spde2.matern(mesh,alpha=2)
```

but in practice we need to also specify the prior distribution for the parameters, and/or modify the parameterisation to suit the specific situation. This is true in particular when the models are used as simple smoothers, as there is then rarely enough information in the likelihood to fully identify the parameters, giving more importance to the prior distributions.

The empirically derived expression $\sqrt{8\nu}/\kappa$ can be used as a measure of the spatial range of the model, which allows us to construct a model with known range and variance ($= 1$) for $(\theta_1, \theta_2) = (0, 0)$, via

```
sigma0 = 1 ## field std.dev.
range0 = 0.2
kappa0 = sqrt(8)/range0
tau0 = 1/(sqrt(4*pi)*kappa0*sigma0)
```

```
spde=inla.spde2.matern(mesh,
  B.tau=cbind(log(tau0),1,0),
  B.kappa=cbind(log(kappa0),0,1),
  theta.prior.mean=c(0,0),
  theta.prior.prec=1)
```

Here, `B.tau` and `B.kappa` can be generalised into matrices where the first column specifies a spatially varying offset for $\log(\tau)$ and $\log(\kappa)$, and the other columns specify basis functions, together defining a non-stationary SPDE model with parameters given by the log-linear models

$$\log(\tau(\mathbf{s})) = b_0^\tau(\mathbf{s}) + \sum_{k=1}^{p} b_k^\tau(\mathbf{s})\theta_k$$
$$\log(\kappa(\mathbf{s})) = b_0^\kappa(\mathbf{s}) + \sum_{k=1}^{p} b_k^\kappa(\mathbf{s})\theta_k$$

where $B_{ik}^{\cdot} = b_k^{\cdot}(\mathbf{s}_i)$, with the columns indexed from 0 and $\mathbf{s}_i$ are the locations of the mesh nodes. Setting sensible priors for θ in these models in general is beyond the scope of this note, and the default priors in the package are likely to change in the near future as we gain more experience with their behaviour, in particular for non-stationary models. The model defined here will give an approximate prior mean variance of $\sigma_0^2 = 1$ for for the field, and approximate prior mean range 0.2 with a prior variance for the range distribution to reach the size of the domain, but not very far beyond.

Models with range larger than the domain size are usually indistinguishable from intrinsic random fields, which can be modelled by fixing κ to zero (or rather some small positive value) with `B.tau=cbind(log(tau0),1)` and `B.kappa=cbind(log(small),0)`. Note that the sum-to-zero constraints often used for lattice based intrinsic Markov models is inappropriate due to the irregular mesh structure, and a *weighted* sum-to-zero constraint is needed to reproduce such models. A future version of the software will include an automatic option to construct such appropriate constraints.

Helper functions in the package can produce precision matrices for given parameter values, as well as samples, so we can generate a synthetic sample:

```
Q=inla.spde2.precision(
    spde,theta=c(0,0))
x=as.vector(inla.qsample(n=1,Q))
proj = inla.mesh.projector(mesh)
image(inla.mesh.project(
        proj,field=x))
```

Here, the `inla.mesh.project/or()` functions are used to map between the basis function weights for the mesh nodes and a lattice format more suited to the standard plotting routines, by default a 100×100 lattice covering the mesh domain.

Bayesian inference

We will now look at a simple example of how to use the SPDE models in latent Gaussian models when doing direct Bayesian inference based on integrated nested Laplace approximations as introduced in [4].

Let's consider a simple Gaussian linear model involving two independent realisations (replicates) of a latent spatial field $x(\mathbf{s})$, observed at the same m locations, $\{\mathbf{s}_1, \ldots, \mathbf{s}_m\}$, for each replicate. For each $i = 1, \ldots, m$,

$$y_i = \beta_0 + c_i\beta_c + x_1(\mathbf{s}_i) + e_i,$$
$$y_{i+m} = \beta_0 + c_{i+m}\beta_c + x_2(\mathbf{s}_i) + e_{i+m},$$

where c_i is an observation-specific covariate, e_i is measurement noise, and $x_1(\cdot)$ and $x_2(\cdot)$ are the two field replicates. Note that the offset, β_0, can be interpreted as a spatial covariate effect, constant over the domain.

We use the basis function representation of $x(\cdot)$ to define a sparse matrix of weights $\mathbf{A}$ such that $x(\mathbf{s}_i) = \sum_j A_{ij} x_j$, where $\{x_j\}$ is the joint set of weights for the two replicate fields. If we only had one replicate, we would have $A_{ij} = \psi_j(\mathbf{s}_i)$. The matrix can be generated by `inla.spde.make.A()`, which locates the points in the mesh and organises the evaluated values of the basis functions for the two replicates:

```
A=inla.spde.make.A(
  mesh,
  loc=points,
  index=rep(1:m,times=2),
  repl=rep(1:2,each=m) )
```

For each observation, `index` gives the corresponding index into the matrix of measurement locations, and `repl` determines the corresponding replicate index. In case of missing observations, one can either keep this `A`-matrix while setting the corresponding elements of the data vector `y` to `NA`, or omit the corresponding elements from `y` as well as from the `index` and `repl` parameters above. Also note that the row-sums of $\mathbf{A}$ are 1, since the piece-wise linear basis functions we use sum to 1 at each location.

Rewriting the observation model on vector form gives

$$\begin{aligned}\mathbf{y} &= \mathbf{1}\beta_0 + \mathbf{c}\beta_c + \mathbf{A}\mathbf{x} + \mathbf{e} \\ &= \mathbf{A}(\mathbf{x} + \mathbf{1}\beta_0) + \mathbf{c}\beta_c + \mathbf{e}\end{aligned}$$

Using the helper functions, we can generate a synthetic sample from our model for the latent fields and observations:

```
Q=inla.spde.precision(
    spde,theta=c(0,0))
x=as.vector(inla.qsample(n=2,Q))
covariate = rnorm(m*2)
y = 5 + covariate*2 +
    as.vector(A %*% x) +
    rnorm(m*2)*0.01
```

The `formula` in `inla()` defines a linear predictor η as the sum of all effects, and an `NA` in a covariate or index vector is interpreted as *no effect*. To accommodate predictors that involve more than one element of a random effect, one can specify a sparse matrix of weights defining an arbitrary linear combination of the elements of η, giving a new predictor vector η^*. The linear predictor output from `inla()` then contains the joint vector (η^*, η). To implement our model, we separate the spatial effects from the covariate by defining

$$\eta_e = \begin{bmatrix} \mathbf{x} + \mathbf{1}\beta_0 \\ \mathbf{c}\beta_c \end{bmatrix},$$

and construct the predictor as

$$\begin{aligned}\eta^*_e &= \mathbf{A}(\mathbf{x} + \mathbf{1}\beta_0) + \mathbf{c}\beta_c \\ &= \begin{bmatrix}\mathbf{A} & \mathbf{I}\end{bmatrix} \eta_e = \mathbf{A}_e \eta_e\end{aligned}$$

so that now $E(\mathbf{y}|\eta_e) = \eta^*_e$. The bookkeeping required to describe this to `inla()` involves concatenating matrices and adding `NA` elements to the covariates and index vectors:

$$\begin{aligned}\mathbf{A}_e &= \begin{bmatrix}\mathbf{A} & \mathbf{I}_{2m}\end{bmatrix} \\ \texttt{field0} &= (1, \ldots, n, 1, \ldots, n) \\ \texttt{field} &= (\texttt{field0}, \texttt{NA}, \ldots, \texttt{NA}) \\ \texttt{offset} &= (1, \ldots, 1, \texttt{NA}, \ldots, \texttt{NA}) \\ \texttt{cov} &= (\texttt{NA}, \ldots, \texttt{NA}, c_1, \ldots, c_{2m})\end{aligned}$$

Doing this by hand with `cBind()`, `c()`, and `rep()` quickly becomes tedious and error-prone, so one can instead use the helper function `inla.stack()`, which takes blocks of data, weight matrices, and effects and joins them, adding `NA` where needed. Identity matrices and constant covariates can be abbreviated to scalars, with a complaint being issued if the input is inconsistent or ambiguous.

We also need to keep track of the two field replicates, and use `inla.spde.make.index()`, which gives a list of index vectors for indexing the full mesh and its replicates (it can also be used for indexing Kronecker

product `group` models, e.g. in simple multivariate and spatio-temporal models). The code

```
mesh.index=inla.spde.make.index(
  name="field",
  n.mesh=mesh$n, n.repl=2)
```

generates a list `mesh.index` with three index vectors,

$$\begin{aligned}\texttt{field} &= (1,\dots,n,1,\dots,n),\\ \texttt{field.repl} &= (1,\dots,1,2,\dots,2),\\ \texttt{field.group} &= (1,\dots,1,1,\dots,1).\end{aligned}$$

The predictor information for the observed data can now be collected, using

```
st.est=inla.stack(
  data=list(y=y),
  A=list(A,1),
  effects=list(
    c(mesh.index,list(offset=1)),
    list(cov=covariate)),
  tag="est")
```

Each "`A`" matrix must have an associated list of "`effects`", in this case `A`:(`field`, `field.repl`, `field.group`, `offset`) and `1`:(`cov`). The `data` list may contain anything associated with the "left hand side" of the model, such as exposure `E` for Poisson likelihoods. By default, duplicates in the effects are identified and replaced by single copies (`compress=TRUE`), and effects that do not affect η^* are removed completely (`remove.unused=TRUE`), so that each column of the resulting `A` matrix has a least one non-zero element.

If we want to obtain the posterior prediction of the combined spatial effects at the mesh nodes, $x(\mathbf{s}_i) + \beta_0$, we can define

$$\begin{aligned}\eta_p &= \mathbf{x} + \mathbf{1}\beta_0\\ \eta_p^* &= \mathbf{I}\eta_p = \mathbf{A}_p\eta_p\end{aligned}$$

and construct the corresponding information stack with

```
st.pred=inla.stack(
  data=list(y=NA),
  A=list(1),
  effects=list(
    c(mesh.index,list(offset=1))),
  tag="pred")
```

We can now join the estimation and prediction stack into a single stack,

```
stack = inla.stack(st.est,st.pred)
```

This stacks the information, and simplifies the result by removing duplicated effects:

$$\begin{aligned}\eta_s^* &= \begin{bmatrix}\mathbf{A}_e & \mathbf{0}\\ \mathbf{0} & \mathbf{A}_p\end{bmatrix}\begin{bmatrix}\eta_e\\ \eta_p\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{A} & \mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{x}+\mathbf{1}\beta_0\\ \mathbf{c}\beta_c\\ \mathbf{x}+\mathbf{1}\beta_0\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{A} & \mathbf{I}\\ \mathbf{I} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{x}+\mathbf{1}\beta_0\\ \mathbf{c}\beta_c\end{bmatrix}\\ &= \mathbf{A}_s\eta_s\end{aligned}$$

In this simple example, the second block row of $\mathbf{A}_s$ (generating $\mathbf{x} + \mathbf{1}\beta_0$) isn't strictly needed, since the same information would be available in η_s itself if we specified `remove.unused=FALSE` when constructing `stack.pred` and `stack`, but in general such special cases can be hard to keep track of.

We are now ready to do the actual estimation. Note that we must explicitly remove the default intercept from the η-model, since that would otherwise be applied twice in the construction of η^*, and the constant covariate `offset` is used instead:

```
formula =
  y ~ -1 + offset + cov +
      f(field, model=spde,
        replicate=field.repl)
inla.result =
  inla(formula,
    data=inla.stack.data(stack),
    family="normal",
    control.predictor=
      list(A=inla.stack.A(stack),
           compute=TRUE))
```

The function `inla.stack.data()` produces the list of variables needed to evaluate the formula (use `inla.stack.data(stack)$E` to extract the Poisson exposures mentioned earlier) and `inla.stack.A()` extracts the $\mathbf{A}_s$ matrix.

Since the SPDE-related contents of `inla.result` can be hard to interpret, the helper function `inla.spde2.result()` can be used to extract the relevant bits and transform them into more user-friendly information, such as posteriors for range and variance instead of raw distributions for θ:

```
result = inla.spde2.result(
  inla.result, "field", spde)
plot(result[
  "marginals.range.nominal"][[1]])
```

The posterior means and standard deviations for the latent fields can be extracted and plotted as follows, where `inla.stack.index()` provides the necessary mappings between the `inla()` output and the original data stack specifications:

```
index=inla.stack.index(
  stack,"pred")$data
image(inla.mesh.project(proj,
 inla.result[[
  "summary.linear.predictor"
  ]]$mean[
  index[mesh.index$field.repl==1]]))
```

As a final note, the `R-INLA` package is in constant development, with new models added as they are needed and developed. The current work for the SPDE models is focusing on completing the documentation, as well as tying together the last bits of the interface for one-dimensional models and simple space-time models. Most of the code for this is already in place, but still in a slightly esoteric form. Also in the pipeline is a separate package for computing level excursion sets with joint excursion probabilities, as well as contour uncertainty regions, mainly developed by David Bolin.

Acknowledgements

I want to thank my collaborators David Bolin, Michela Cameletti, Peter Guttorp, Janine Illian, Johan Lindström, Daniel Simpson, and last but not least Håvard Rue, who is also the main developer of `R-INLA`. I'm also grateful to the ISBA Bulletin editors for providing this opportunity to discuss the SPDE/GMRF part of the `R-INLA` software, and giving me a chance to climb onto a few, though arguably diminutive, soap-boxes regarding spatial modelling philosophy.

The R-code from this note is available at `http://people.bath.ac.uk/fl353/isba/isbaspde.R`, including code for generating more appealing images than the `image()` function provides.

References

[1] D. Bolin and F. Lindgren, Spatial models generated by nested stochastic partial differential equations, with an application to global ozone mapping, *Annals of Applied Statistics*, 5(2011), pp. 523–550.

[2] M. Cameletti, F. Lindgren, D. Simpson, H. Rue, Spatio-temporal modeling of particulate matter concentration through the SPDE approach, *AStA Advances in Statistical Analysis*, 2012, in press. Case study code available on `http://www.r-inla.org`.

[3] F. Lindgren, H. Rue, J. Lindström, An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach (with discussion), *Journal of the Royal*

Statistical Society, Series B, 73(2011), Part 4, pp. 423–498.

[4] H. Rue, S. Martino, and N. Chopin, Approximate Bayesian Inference for Latent Gaussian Models Using Integrated Nested Laplace Approximations (with discussion), *Journal of the Royal Statistical Society, Series B*, 71, pp. 319–392.

[5] D. Simpson, F. Lindgren, H. Rue, Think continuous: Markovian Gaussian models in spatial statistics, *Spatial Statistics*, 1, pp. 16–29.

[6] D. Simpson, F. Lindgren, H. Rue, In order to make spatial statistics computationally feasible, we need to forget about the covariance function, *Environmetrics*, 23(1), pp. 65–74.

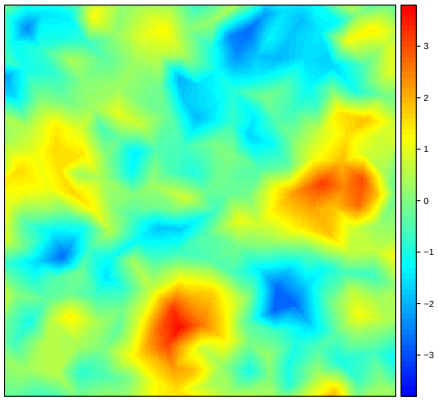

Figure 2: *Simulated true field*

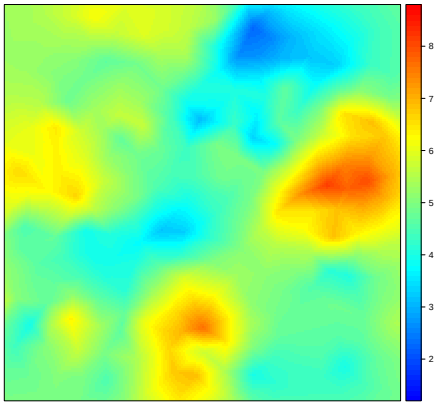

Figure 1: *Posterior mean*

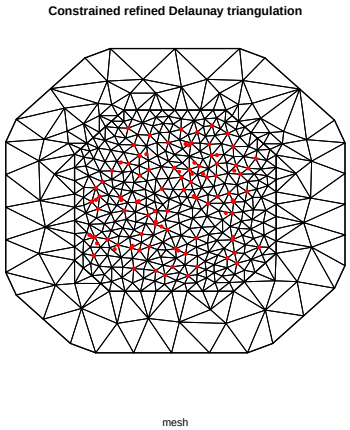

Figure 1: *Triangulated mesh*

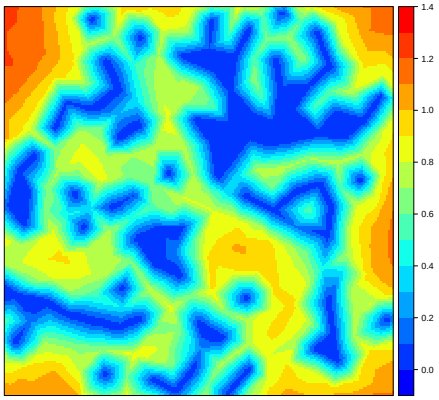

Figure 4: *Posterior standard deviations*