Fixed Rank Kriging for spatial data

Andrew Zammit-Mangion

January 24, 2016

Case 1: 2D point data on the plane

In this vignette we apply `FRK` to the case when we have spatial data, either on the plane. For the 2D data on the plane, we consider the `meuse` data, which can be found in the package `sp`.

For this vignette we need to load the following packages

```
library(sp)
library(ggplot2)
library(dplyr)
library(FRK)
```

and, to keep the document tidy, we will set the `progress` package option to `FALSE`. Parallelisation is frequently used, but for the purposes of this document we will set the `parallel` option to 0 as well.

```
opts_FRK$set("progress",FALSE)
opts_FRK$set("parallel",0L)
```

We first load the `meuse` data

```
data(meuse)
print(class(meuse))

## [1] "data.frame"
```

The `meuse` data is of class `data.frame`. However, `FRK` needs all spatial objects to be of class `SpatialPointsDataFrame` or `SpatialPolygonsDataFrame`, depending on whether the object is point-referenced of area-referenced. We change the `meuse` data into a `SpatialPointsDataFrame` by applying the `coordinates` function as follows:

```
coordinates(meuse) = ~x+y # change into an sp object
```

With each data location, we also need to attribute a *fine-scale variation* component. This variation describes variation that occurs on sub-basic-areal-unit (BAU) scale. The fine-scale variation need only be known up to a constant of proportionality; this constant is estimated using maximum likelihood with `SRE.predict` later on. Typically, geographic features such as altitude are appropriate, but in this case we will just set this component to unity. It is important that this field is labelled 'fs':

```
meuse$fs <- 1
```

Based on the data we now generate BAUs. For this we can use the helper function `auto_BAU` as follows:

```
set.seed(1)
GridBAUs <- auto_BAUs(manifold = plane(),
                      cellsize = c(100,100),
                      type = "grid",
                      data = meuse,
                      convex=-0.05)
```

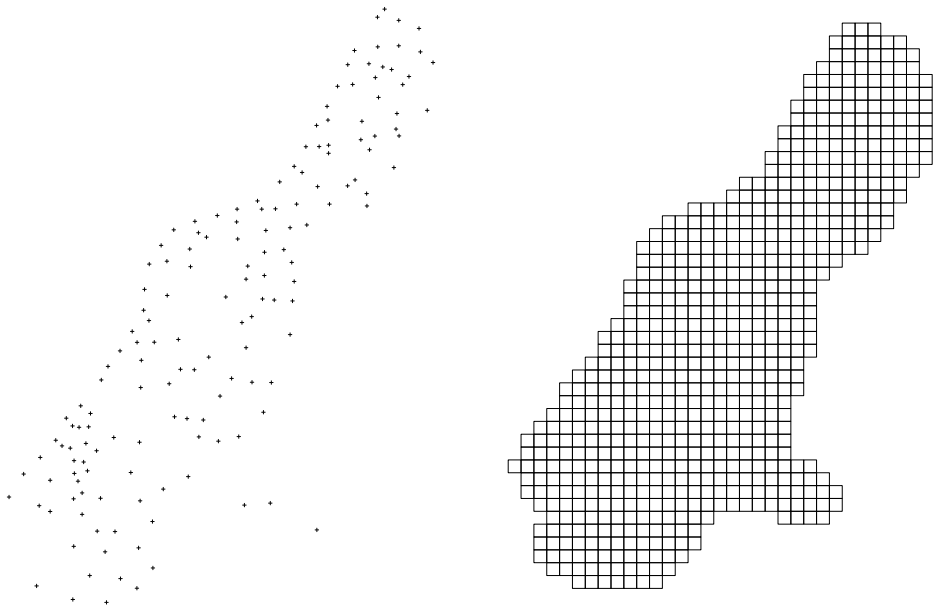


Figure 1: Data locations (left panel) and BAUs (right panel) for the meuse dataset.

The `auto_BAUs` function takes several arguments (see `help(auto_BAUs)` for details). Above, we instruct the helper function to construct BAUs on the plane, centred around the data `meuse` with each BAU of size 100 m $\times$ 100 m. The `type="hex"` input instructs that we want a rectangular grid and not a hexagonal lattice (type `hex` for a rectangular grid), and 'convex=-0.05' is a specific parameter controlling the spatial-domain boundary (see `INLA::inla.nonconvex.hull` for more details). As with the observations, the BAUs need to be associated with a variable that is proportional to the fine-scale variability. This can be set to 1 when we have no prior information on this variability:

```
GridBAUs$fs <- 1
```

The data and BAUs are illustrated in Fig. 1.

```
par(mfrow=c(1,2))
plot(meuse)
plot(GridBAUs)
par(mfrow=c(1,1))
```

`FRK` decomposes the spatial process as a sum of basis functions that may either be user-specified or constructed using helper functions. To create spatial basis functions we use the helper function `auto_basis` as follows:

```
G <- auto_basis(m = plane(),
                data=meuse,
                nres = 2,
                prune=5,
                type = "Gaussian",
                regular = 0)
```

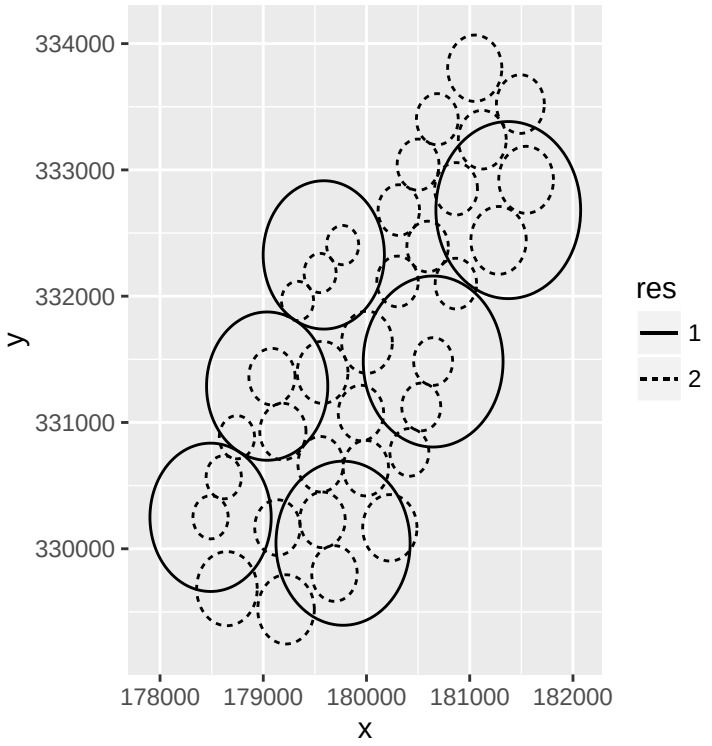


Figure 2: Basis functions automatically generated for the meuse dataset with 2 resolutions.

```
## [1] "Number of basis at resolution 1 = 6"
## [1] "Number of basis at resolution 2 = 34"
```

The argument `nres = 3` indicates how many resolutions we wish, while `type = Gaussian` indicates that the basis set we want is a Gaussian basis. Basis functions are constructed such that the process is accurately represented where data is present, and not represented otherwise. This is controlled by the parameter `prune`, see `help(auto_basis)` for details. The basis can be visualised using `show_basis`, see Fig. 2.

```
show_basis(G)

## Note:  show_basis assumes spherical distance functions when plotting
```

Now that we have the BAUs and the basis functions, we can construct the SRE model. As fixed effects, we just use an intercept; if we wish to use covariates, then these need to be added to the BAUs first. The fixed effects are supplied in a usual `R` formula:

```
f <- log(zinc) ~ 1
S <- SRE(f = f,
         data = list(meuse),
         BAUs = GridBAUs,
         basis = G,
         est_error=T)

## [1] "Binned data in 0.255000000000001 seconds"

## Loading required namespace:  gstat

## [1] "sigma2e estimate = 0.0289267710553666"
```

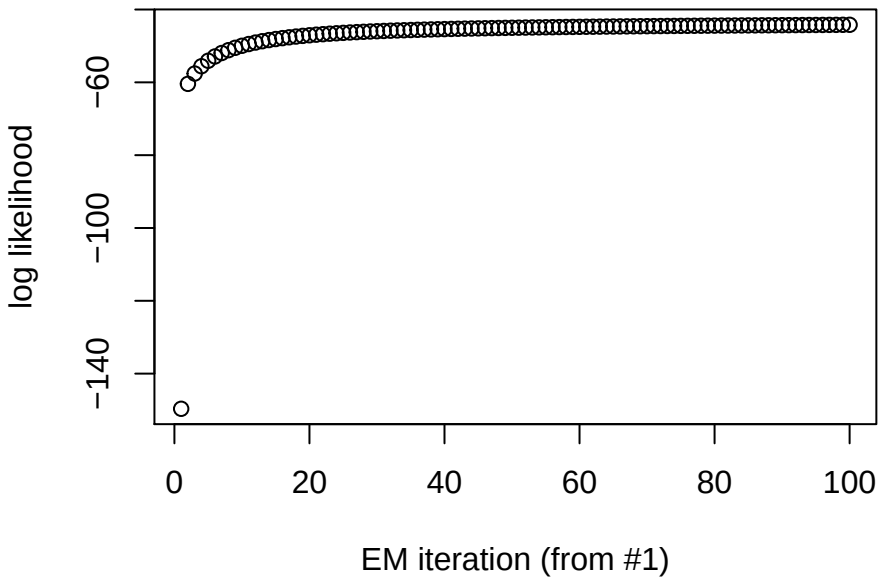


Figure 3: Convergence of the EM algorithm for the meuse dataset

The function `SRE` takes as arguments the formula, the data (as a list, since many datasets may be used), the BAUs, the basis functions and a flag, `est_error`, indicating whether we wish to attempt to estimate the measurement-error variance or not using variogram methods. This is only allowed with spatial data since it's time consuming and, if not supplied, the data needs to also be supplied with a field `std`.

The `SRE` model is fit using the function `SRE.fit`. Maximum likelihood is carried out using the expectation maximisation (EM) algorithm, which stops either when `n_EM` is exceeded, or when the likelihood across subsequent steps does not change by more than `tol`. In this case the EM algorithm converges in 100 iterations; see Fig. 3.

```
S <- SRE.fit(SRE_model = S,
             n_EM = 400,
             tol = 0.01,
             print_lik=TRUE)

## [1] "Minimum tolerance reached"
```

Finally, we predict at all the BAUs with the fitted model. This is done using the function `SRE.predict`. The argument `use_centroid` indicates whether we want to treat the BAU as a prediciton point or as a prediction area, see `help(SRE.predict)` for details.

```
GridBAUs <- SRE.predict(S,use_centroid = TRUE)
```

The object `GridBAUs` contains the prediction mean and prediction variance at the BAU level. These can be plotted using the standard plotting commands in `sp`, or else using `ggplot2`. To use the latter, we first need to convert the `Spatial` objects to data-frames as follows:

```
BAUs_df <- SpatialPolygonsDataFrame_to_df(sp_polys = GridBAUs,
                                          vars = c("mu","var"))

## Joining by:  "id"
```

The function takes as argument the BAUs and the variables we wish to extract from the BAUs. Now `ggplot2` can be used to plot the observations and the prediction mean and variance, for example:

```
g1 <- LinePlotTheme() +
    geom_polygon(data=BAUs_df ,aes(x,y,fill=mu,group=id),
                 colour="light grey") +
    scale_fill_distiller(palette="Spectral",trans="reverse") +
    geom_point(data=data.frame(meuse),
               aes(x,y,fill=log(zinc)),
               colour="black",
               pch=21, size=3) +
    coord_fixed()

g2 <- LinePlotTheme() +
    geom_polygon(data=BAUs_df,aes(x,y,fill=sqrt(var),group=id),
                 colour="light grey") +
    scale_fill_distiller(palette="Spectral",trans="reverse",
                         guide = guide_legend(title="se")) +
    coord_fixed()
```

These plots are shown in Fig. 4.

Now, assume that we wanted to predict over regions encompassing several BAUs. Then we need to set the `pred_BAUs` argument in the function `auto_BAUs`. First, we create this larger regionalisation as follows

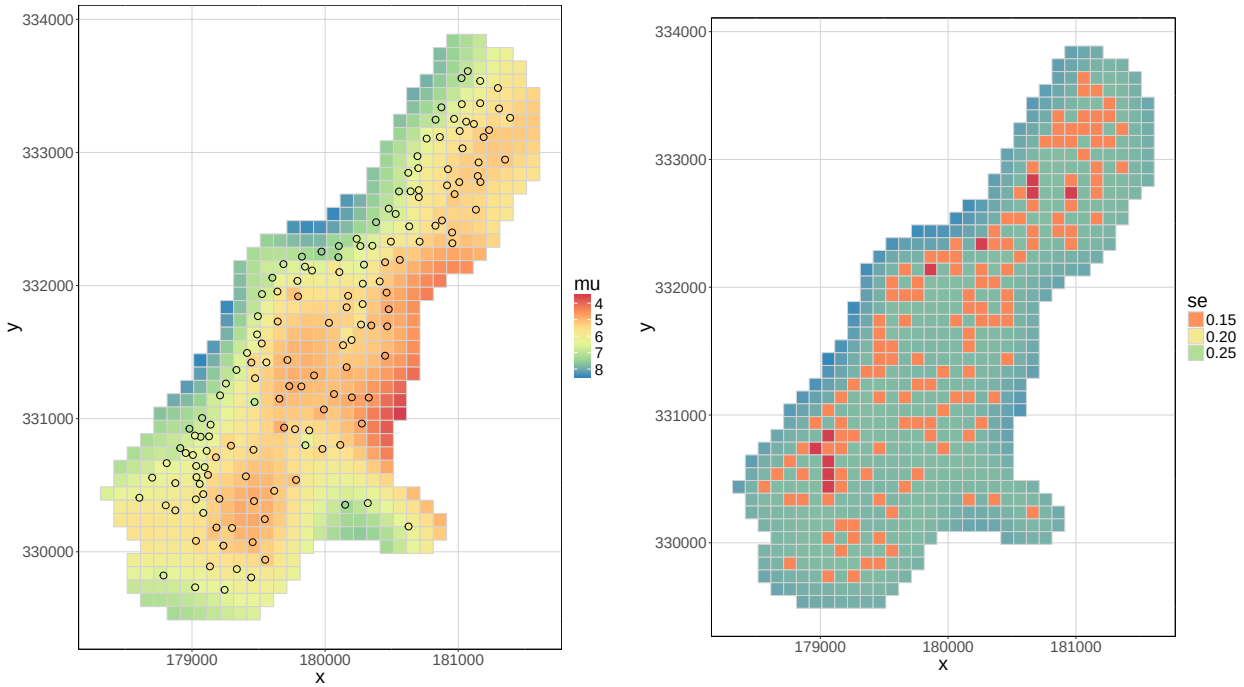
```
set.seed(1)
Pred_regions <- auto_BAUs(manifold = plane(),
                          cellsize = c(600,600),
                          type = "grid",
                          data = meuse,
                          convex=-0.05)
```

and carry out prediction on the larger polygons:

```
Pred_regions <- SRE.predict(S,use_centroid = TRUE,pred_polys = Pred_regions)
```

The mean and variance can be visualised as before. These plots are shown in Fig. 5.

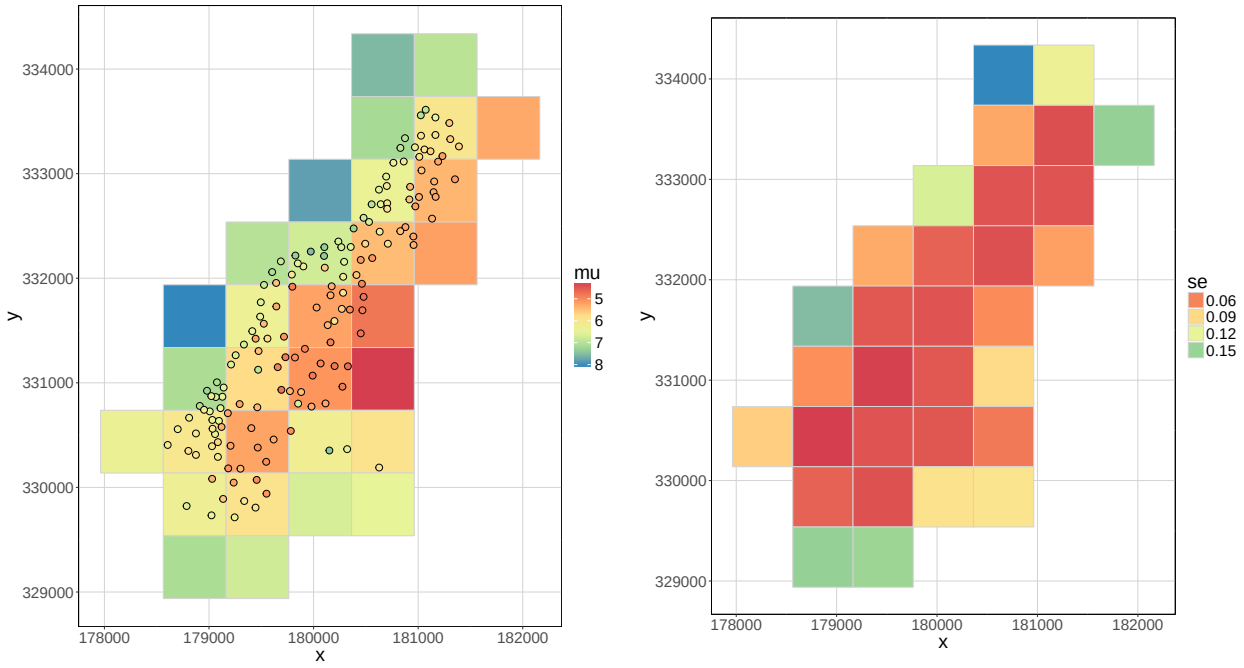
```
## Joining by:  "id"
```



(a) 1

(b) 2

Figure 4: Prediction mean and error obtained with FRK from the meuse dataset at the BAU level.



(a) 1

(b) 2

Figure 5: Prediction mean and error obtained with FRK from the meuse dataset at a custom resolution.