
Applied geostatistics

Exercise 8a: Spatial simulated annealing

D G Rossiter
University of Twente, Faculty of Geo-Information Science & Earth
Observation (ITC)

Vandita Srivastava
Geoinformatics Department, Indian Institute of Remote Sensing, Dehradun

January 2, 2014

Contents

1	Introduction	1
2	SSA optimization for Ordinary Kriging: development	3
2.1	Define the study area	4
2.2	Model of spatial dependence	6
2.3	Fitness function	7
2.4	Setup for SSA	8
2.5	The main SSA loop: one time	13
2.5.1	Maximum distance to move a point	13
2.5.2	Moving one point	16
2.5.3	Plotting a changed scheme	17
2.5.4	Accepting or rejecting the new scheme	18
2.6	The main SSA loop: iterations	18
2.7	Showing the evolution of the scheme	19
2.8	Answers	23
3	SSA optimization for Ordinary Kriging: application	24
3.1	A function for SSA	24
3.1.1	Saving the function to a file for later execution	28
3.2	Initial sampling scheme	28

Version 1.3 Copyright © 2009 ITC; 2010, 2013 University of Twente, Faculty ITC All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited. To adapt or translate please contact the author (<http://www.itc.nl/personal/rossiter>).

3.3	Objective function: minimize the mean kriging variance	30
3.4	Objective function: minimize the maximum kriging variance .	33
3.5	Optimizing from another starting configuration	35
3.6	Answers	39
4	Sampling an irregularly-shaped area	40
4.1	Answers	49
5	SSA optimization for Kriging with External Drift	50
6	SSA optimizing by variogram matching	50
6.1	Indicator variogram, no bin numbers	55
6.2	Indicator variogram, bin numbers	60
6.3	Sampling optimization in a simulated field	63
6.4	Answers	73
7	Challenges	74
	References	76
	Index of R concepts	77

1 Introduction

A typical problem in sampling is how to optimally place a limited number of observations in a study area in order to extract the maximum information at minimum cost. We consider here the information to be a map over some study area, made by kriging from the sample points.

The first question is what do we mean by “optimal”? There must be an objective criterion that can be computed for a sampling scheme, which is to be minimized or maximized.

One obvious criterion is to minimize costs:

- the fewest samples required to reach a given precision;
- the minimum travel and set-up cost.

These are outside the scope of the present discussion; here we assume the sample size is fixed and logistics are not a problem. For example, if the task is to sample a small polluted site where travel costs between observations are not significant.

The first criterion we will examine here is the accuracy of the prediction of the map to be made from the point samples, in particular the **prediction variance**. Recall from the theory of kriging that the prediction variance at a prediction location depends only on:

1. the configuration of the sample points relative to each other and to the prediction location; and
2. the model of spatial dependence (e.g. variogram model)

not on the data values. Therefore, before any observations are made, we can compute the prediction variance at any point in the study area – always assuming we have a correct model of spatial dependence, perhaps from a pilot study in the area or a previous study in a similar area.

From this we can decide on an **optimality criterion**, e.g.

1. **minimize** the **mean** prediction variance over the study area, so that it is, on average, well-mapped; this is called MEAN_OK by van Groenigen [10];
2. **minimize** the **maximum** prediction variance in the study; this was the criterion used in the OSSFIM approach of McBratney and Webster [6, 5]; it guarantees that no location will be poorly-mapped; this is called MAX_OK by van Groenigen [10];

The first would be appropriate when estimating spatial averages to a given precision. The second would be appropriate when the entire area must be mapped to a given precision, e.g. to guarantee there is no health risk in a polluted area.

In a few simple cases the optimum design can be computed analytically; in particular McBratney et al. [6] showed that on an infinite plane a regular triangular scheme gives minimum maximum prediction (MAX_OK) variance

and that a square grid is not much worse. But in reality we often have **constraints**:

1. the study area is never infinite, so there are **edge effects**;
2. the study area is often **irregularly-shaped**;
3. there may be parts of the study area that **can not be sampled** (e.g. soil samples under buildings);
4. we may have some **existing observations** in the study area.

An obvious option is to place samples completely randomly. However, if there is any spatial dependence, a random scheme will not be optimal either for estimating population parameters (e.g. means) or for mapping.

It is not possible, in general, to solve this problem analytically. One solution is to try large numbers of schemes and compare their optimality; but since the number of possible sample locations is usually very large, this is impractical. Just with ten sample points on a 100×100 grid there are approximately $2.7434 \cdot 10^{33}$ possible sample schemes.

Another way is to start with a random or fixed scheme and repeatedly modify it, each time checking if the modification makes the scheme better. This is known as **spatial simulated annealing** (SSA) and was the subject of a thesis [9] and series of papers [10, 11] by van Groenigen.

In this exercise we will implement SSA to optimize sampling design for interpolation by Ordinary Kriging (OK), i.e. where only the target variable is used for interpolation; in §5 we briefly discuss how covariables could be used in the optimization.

Two-phase sampling In many environmental studies there are two sampling phases:

1. to determine the **spatial structure** of the target variable (e.g. to estimate the variogram model);
2. to **map** the target variable.

The first stage could be set up with a nested design [14, 13], transects with variable spacing, or a similar design [8, 7, 2]. Once this is done and the variogram estimated, additional observations are located to map the whole area, using both sets. The second stage can be optimized, for example by MAX_OK, to ensure the quality of mapping.

There are two questions:

1. How many more samples?
2. Where to place them?

In practice this is an iterative procedure:

1. Compute the fitness of the first-stage design, e.g. the maximum kriging prediction variance in the study area, using just these known points;

2. Compare this fitness to the required standard; most likely there will be under-sampled areas far from the first-stage;
3. Looking at the kriging prediction variance in more densely-sampled areas, estimate how dense a network might be required;
4. Use SSA_OK to optimize the placement of these new points;
5. Examine the fitness to see if it is good enough; if too precise some points can be eliminated, if not precise enough more points must be added.
6. Repeat steps 3–5 as necessary.

The techniques of SSA, developed below, can be applied for the second phase.

2 SSA optimization for Ordinary Kriging: development

In this section we develop SSA for OK, first in a square study area and then in an arbitrary polygon (§4). We use R spatial classes [1] because this is a spatial problem.

We will approach the SSA problem with the following steps:

1. Define the study area as a (set of) polygon(s);
2. Discretize the study area into a large number of prediction points;
3. Define an optimality criterion, for example minimizing the mean kriging prediction variance; the computed value is called the **fitness** of the scheme;
4. Define the model of spatial variability (i.e. variogram);
5. Define a starting sampling scheme: number of points and their proposed location;
6. (optionally) define previous sampling points that can contribute to the final map;
7. Compute the fitness of the starting scheme;
8. Annealing: repeat a “large” number of times, each time reducing the “temperature”:
 - (a) Randomly select a point to move;
 - (b) Randomly compute a proposed shift, the maximum distance depending on the “temperature”;
 - (c) Ensure the proposed shifted point is still in the study area (otherwise, repeat previous step);
 - (d) Tentatively move the point; compute the fitness of the modified scheme;
 - (e) If the modified scheme:

- has **better** fitness, accept it;
 - has **worse** fitness, accept it with a certain probability, depending on the so-called “temperature”;
9. Stop after a set number of iterations, or with some other stopping criterion (e.g. small changes in fitness for many steps).

The reason to accept a poorer scheme with a certain probability is to keep the SSA from being stuck in a local optimum. This is known as the Metropolis criterion [10, eqn. 5]; the probability $P(S_0 \rightarrow S_1)$ of accepting the new scheme is:

$$\begin{aligned} P(S_0 \rightarrow S_1) &= 1, \text{ if } \phi(S_1) \leq \phi(S_0) \\ P(S_0 \rightarrow S_1) &= \exp\left(\frac{\phi(S_0) - \phi(S_1)}{c}\right), \text{ if } \phi(S_1) > \phi(S_0) \end{aligned} \quad (1)$$

where S_0 is the fitness of the current scheme, S_1 is the fitness of the proposed new scheme, and c is the so-called **temperature**. This word is used by analogy with physical annealing, i.e. the cooling of a metal into a crystalline structure with minimum internal energy (strain). During this, molecules jump around the structure, with the maximum distance controlled by the physical temperature.

There are some tricky aspects of this procedure. Fortunately, with cheap computer power and the R environment, it costs little to experiment.

Task 1 : Load the necessary libraries. •

```
> require(sp)
> require(gstat)
> require(lattice)
```

2.1 Define the study area

We illustrate the simplest case with a unit square study area. The same technique can be used for any polygon or set of polygons; see §4, below.

We will use spatial overlay (the **over** method) to determine whether a shifted point is in the polygon, so we must set this up as a spatial object of class **SpatialPolygons**. As explained by Bivand et al. [1, §2.6], a **SpatialPolygons** object is made up of identified **list** of **Polygons**¹, which in turn are a **list** of individual **Polygon** objects; this is a two-column matrix of coördinates with the first and last identical.

Note: The **matrix** function has optional arguments **ncol** **nrow** to specify the number of rows and columns, and **byrow** (default **FALSE**) to indicate whether the matrix should be filled from the list by row rather than by columns.

¹ e.g. a legend category of a thematic map

Task 2 : Define a `SpatialPolygons` object composed of one `Polygons` object, covering a 1 by 1 square. •

```
> p <- SpatialPolygons(list(Polygons(list(Polygon(matrix(c(0,
+ 0, 0, 1, 1, 1, 1, 0, 0, 0), byrow = T, nrow = 5,
+ ncol = 2))), ID = 1)))
> summary(p)
```

```
Object of class SpatialPolygons
Coordinates:
  min max
x    0   1
y    0   1
Is projected: NA
proj4string : [NA]
```

To plot this polygon with `spplot` it must have at least one attribute.

Task 3 : Using the polygon ID as the attribute, promote the object to a `SpatialPolygonsDataFrame` object. •

This is accomplished with the `SpatialPolygonsDataFrame` method:

```
> p <- SpatialPolygonsDataFrame(p, data = data.frame(id = 1))
> str(p)
```

```
Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
..@ data      :'data.frame':      1 obs. of  1 variable:
.. ..$ id: num 1
..@ polygons  :List of 1
.. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
.. .. ..@ Polygons :List of 1
.. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
.. .. .. .. ..@ labpt  : num [1:2] 0.5 0.5
.. .. .. .. ..@ area   : num 1
.. .. .. .. ..@ hole   : logi FALSE
.. .. .. .. ..@ ringDir: int 1
.. .. .. .. ..@ coords : num [1:5, 1:2] 0 0 1 1 0 0 1 1 0 0
.. .. .. ..@ plotOrder: int 1
.. .. .. ..@ labpt     : num [1:2] 0.5 0.5
.. .. .. ..@ ID       : chr "1"
.. .. .. ..@ area     : num 1
..@ plotOrder : int 1
..@ bbox      : num [1:2, 1:2] 0 0 1 1
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : chr [1:2] "x" "y"
.. .. ..$ : chr [1:2] "min" "max"
..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
.. .. ..@ projargs: chr NA
```

To normalize the maximum shift we need to know the distance across the study area.

Task 4 : Computed the distance across the study area as the diagonal of the bounding box. •

This can be extracted with the `bbox` method:

```
> (max.dist <- sqrt(((bbox(p)[1, "max"] - bbox(p)[1, "min"])^2) +
+   ((bbox(p)[2, "max"] - bbox(p)[2, "min"])^2)))

[1] 1.4142
```

The study area must be discretized into a `SpatialPixels` object, because kriging interpolation is at points (or centres of small blocks).

Task 5 : Discretize the area into a 100 by 100 regular grid of spatial pixels. •

The `spsample` method creates a `SpatialPoints` object; here we ask for a 100 by 100 regular grid. The `gridded` method converts this to a `SpatialPixels` object, and the `fullgrid` method converts this to a `SpatialGrid` object, which is the most efficient representation of a full grid.

```
> resolution <- 100
> grid <- spsample(p, n = resolution^2, type = "regular")
> class(grid)

[1] "SpatialPoints"
attr(,"package")
[1] "sp"

> gridded(grid) = T
> class(grid)

[1] "SpatialPixels"
attr(,"package")
[1] "sp"

> fullgrid(grid) = T
> class(grid)

[1] "SpatialGrid"
attr(,"package")
[1] "sp"
```

Note that the resolution of this grid controls the precision of the optimality calculation, but does not affect the placement of the sample points. In practice these could be moved to their closest grid point, if more precise location in the field is not possible.

2.2 Model of spatial dependence

We have decided to use some characteristic of the kriging prediction variance over the study area as the fitness criterion. Therefore, we need a model of spatial dependence. In an actual application this would be determined by

variogram analysis of a previous study in the same or a similar area; here we just assume a model. You can change this and see its effect on the resulting sampling plan.

Since the study area is 1 by 1, we use a variogram model with a shorter range, here $1/3$ of the maximum distance. We choose an exponential model, for which the effective range is thrice the model's range parameter, so the maximum distance is divided by $3 \cdot 3 = 9$ to obtain the range parameter:

Task 6 : Specify an exponential variogram model with unit partial sill and an effective range of $1/3$ of the maximum distance. •

```
> (vm <- vgm(psill = 1, model = "Exp", range = max.dist/(3 *
+           3), 0))

      model psill  range
1   Nug     0 0.00000
2   Exp     1 0.15713
```

Q1 : *Why haven't we discussed the partial sill and nugget? What effect will these have on the kriging prediction variance of different schemes?* [Jump to A1](#) •

2.3 Fitness function

For each sampling scheme we want to compute a fitness. We do this with a function which takes as its **argument** a sampling scheme, as a **SpatialPoints** object, and **returns** the fitness, some characteristic of the kriging prediction variance over the discretized study area **grid**, using the model of spatial dependence **vm** from the previous section.

There are several possible characteristics of the kriging prediction variance that are reasonable measures of fitness. The metrics that have been most used are the **mean** (MEAN_OK) and the **maximum** (MAX_OK); one could also use a quantile. To allow flexibility, a second argument to the fitness function is the name of a function to be applied to the results of kriging.

Task 7 : Create an appropriate fitness function. •

```
> obj.k <- function(scheme, f = "mean") {
+   k <- krige(z ~ 1, loc = scheme, model = vm, newdata = grid)
+   return(f(k$var1.var))
+ }
> class(obj.k)

[1] "function"
```

Reasonable choices for **f** are built-in R functions **mean** (for MEAN_OK), which we use as default here, and **max** (for MAX_OK); however any built-

in function can be named, or a function can be defined and passed as the argument.

2.4 Setup for SSA

The first decision we must make is the numeric precision of coordinates for the scheme. This depends on the precision of geo-location in the field and size of support.

Task 8 : Set the precision to 0.001, which is 1/1000 of the bounding box linear dimension. •

```
> prec <- 3
```

To begin, we need a starting sampling scheme, as a `SpatialPointsDataFrame` object.

Task 9 : Create a random discretization of the study area, using the `sp-sample` method, and then add a dummy attribute value created with the `data.frame` method.

To ensure that your results will be the same as the ones here, set the random number seed with the `set.seed` function to the arbitrary number 621. •

Note: Clearly, a random scheme will not be close to optimal. In practice we would generally start with a regular grid and then optimize it (let the new points move) with regard to the fixed (known) points. We will see that approach in §3.2; for now we use a random initial scheme to illustrate how SSA works, since the final configuration will be very different from the initial one.

```
> n.pts <- 10
> set.seed(621)
> scheme <- SpatialPointsDataFrame(spsample(p, n = n.pts,
+   type = "random"), data.frame(z = rep(0, n.pts)))
> scheme@coords <- round(scheme@coords, prec)
> print(scheme)
```

```
      coordinates z
1 (0.753, 0.311) 0
2 (0.126, 0.402) 0
3 (0.192, 0.008) 0
4 (0.286, 0.64) 0
5 (0.932, 0.6) 0
6 (0.379, 0.624) 0
7 (0.377, 0.666) 0
8 (0.006, 0.006) 0
9 (0.155, 0.35) 0
10 (0.996, 0.377) 0
```

The attribute, here named `z`, is just a repetition of 0's. An attribute must be defined in order to interpolate with the `krige` method, used in the fitness

function.

Q2 : Why are the actual data values at these locations not necessary for SSA? Jump to A2 •

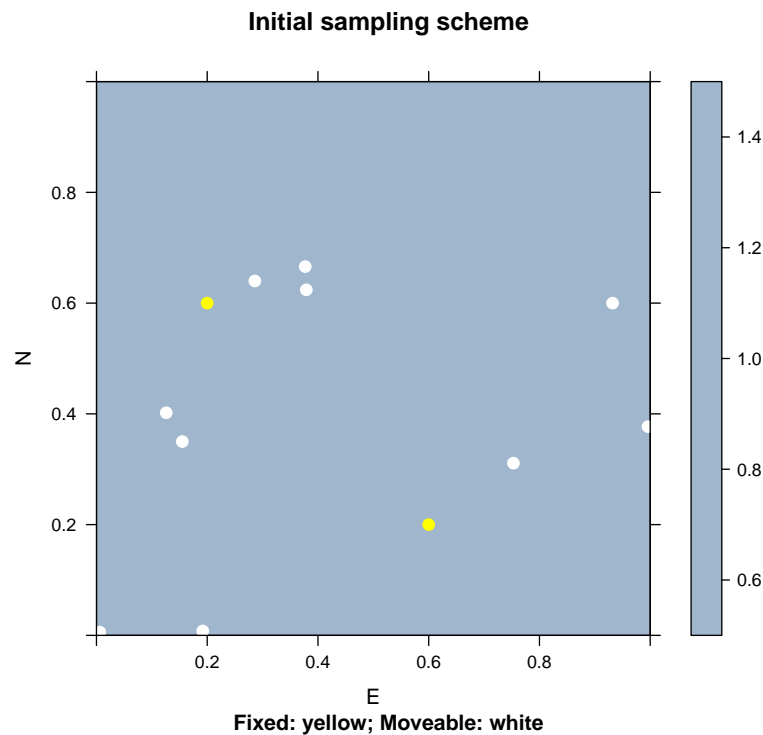
Simulated annealing is often used to refine a previous sampling scheme; these are included as known and fixed points.

Task 10 : Specify a list of two fixed points as a `SpatialPointsDataFrame` object. •

```
> fix.pts <- data.frame(x = c(0.2, 0.6), y = c(0.6, 0.2),  
+   z = 0)  
> coordinates(fix.pts) <- ~x + y
```

Task 11 : Display this initial scheme, along with the fixed points. •

```
> print(spplot(p, xlab="E", ylab="N", col.regions="slategray3",  
+   main="Initial sampling scheme",  
+   sub="Fixed: yellow; Moveable: white",  
+   scales=list(draw=T),  
+   sp.layout = list(list("sp.points", scheme,  
+     col="white", pch=20, cex=1.6),  
+     list("sp.points", fix.pts, col="yellow",  
+       pch=20, cex=1.6))  
+   ))
```



Since we've defined a fitness function (§2.3), we can compute the fitness of this initial sampling scheme.

Task 12 : Compute the fitness, using the mean kriging variance (MEAN_OK) as the criterion: •

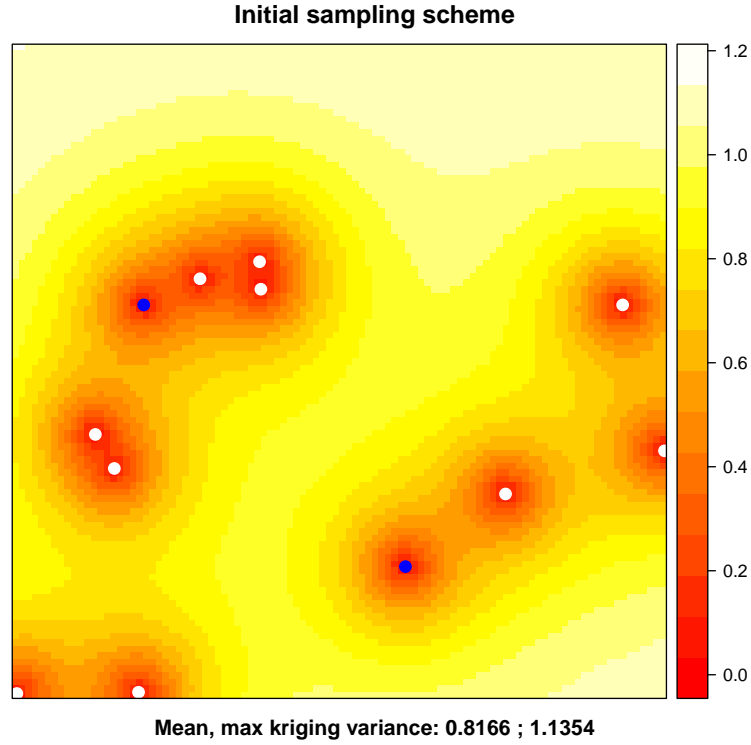
```
> (fitness <- obj.k(rbind(fix.pts, scheme), mean))

[using ordinary kriging]
[1] 0.81658
```

Q3 : What are the units of this number? What does it measure? [Jump to A3](#) •

Task 13 : Display the initial scheme with the optimality criterion (kriging variance) as background; this can be compared to the “optimal” schemes developed later. •

```
> k <- krige(z ~ 1, loc=rbind(fix.pts,scheme), model=vm,
+          newdata=grid)
> gridded(k) <- T; fullgrid(k) <- T
> pts.s <- list("sp.points", scheme, col="white",
+             pch=20, cex=1.6)
> pts.f <- list("sp.points", fix.pts, col="blue",
+             pch=20, cex=1.6)
> print(spplot(k, zcol="var1.var",
+             col.regions=heat.colors(64),
+             main=paste("Initial sampling scheme"),
+             sub=paste("Mean, max kriging variance:",
+                 round(mean(k$var1.var),4),";",
+                 round(max(k$var1.var),4)),
+             sp.layout = list(pts.s, pts.f)
+             ))
> rm(k, pts.s, pts.f)
```



Note: The `gridded` and `fullgrid` methods cast the kriging interpolation results into `SpatialGridDataFrame`; this gives an image-like plot, rather than a set of (regular-spaced) points.

The next item is the “temperature”, which also is reduced in each step:

$$T_{k+1} = \alpha \cdot T_k \quad (2)$$

where k is the step number and $\alpha < 1$ is an empirical factor that reduces the temperature; we must also specify an initial temperature T_0 . If α is too close to 1, the process is inefficient, with too many jumps away from optimality; if α is too small, the solution may get stuck in local minima. So we need to specify the initial temperature and the decay factor.

Note: Another possibility is to keep the same temperature for a fixed number of iterations before lowering it. This makes a longer “cooling” process.

The probability of accepting a poorer (temporary) solution at step k is given by the Metropolis criterion, here re-written from Equation 1 in a slightly simpler form [3, Eqn. 6]:

$$p = e^{-\Delta f / T_k} \quad (3)$$

where T_k is the current “temperature” and Δf is the change in fitness due to the proposed new scheme. Note that this will be positive for a poorer solution, so its complement is used for the exponent.

One way to choose the initial temperature is so the initial probability of accepting a poorer solution equals some constant, conventionally 0.8 [4].

The initial temperature can be calculated from this probability, normalized by the logarithm of the average increase in fitness when all increases (i.e., poorer configurations) are accepted [3, Eqn. 7]; this inverts Equation 3:

$$T_0 = \frac{-\overline{\Delta f^+}}{\log(p_0)} \quad (4)$$

with a default probability p_0 at 0.8, an empirical value. Note that $\log(0.8)$ is equal to -0.2231; a higher probability will lead to a smaller logarithm (closer to zero); since the log-probability is the denominator this will increase the initial temperature. A “hotter” system is more likely to accept worse moves.

Task 14 : Compute this average by shifting points and evaluate the fitness 200 times, and averaging the increases over the starting fitness, i.e., only the poorer solutions. Set the maximum shift in both coördinates at $\sqrt{2}/6$ of the diagonal distance across the bounding box; this is 1/3 of each dimension. •

Note: Two hundred (200) steps is a small sample of possible positive changes in fitness, but it should give a reasonable estimate of the change in fitness of initial changes.

```
> max.shift <- round(max.dist * sqrt(2)/6, prec)
> n <- 200
> sum.delta.fit <- 0
> i <- 0
> while (i < n) {
+   pt <- sample(1:n.pts, 1)
+   pt.old <- coordinates(scheme)[pt, ]
+   repeat {
+     xnew <- pt.old[1] + runif(1, min = -max.shift,
+     max = max.shift)
+     ynew <- pt.old[2] + runif(1, min = -max.shift,
+     max = max.shift)
+     pt.new <- data.frame(x = xnew, y = ynew)
+     coordinates(pt.new) <- c(1, 2)
+     if (!is.na(over(pt.new, p)))
+       break
+   }
+   scheme.try <- scheme
+   scheme.try@coords[pt, ] <- c(xnew, ynew)
+   fitness.new <- obj.k(rbind(fix.pts, scheme.try),
+   mean)
+   if (fitness.new > fitness) {
+     sum.delta.fit <- sum.delta.fit + (fitness.new -
+     fitness)
+     i <- i + 1
+   }
+ }
> mean.delta.fit <- sum.delta.fit/n
> rm(max.shift, n, i, pt, pt.old, xnew, ynew, scheme.try,
+   fitness.new)
```

Task 15 : Display the average change in fitness after one random move from the random starting scheme, for the new schemes that have poorer fitness.

•

```
> print(paste("Average positive change in fitness:", round(mean.delta.fit,
+ 4)))

[1] "Average positive change in fitness: 0.0077"
```

The average change is negative, i.e. the new schemes have on average a lower mean kriging prediction variance, and so are better.

Task 16 : Specify the starting temperature as the average change to worse fitness, normalized by the logarithm of the desired probability of accepting a poorer scheme (here, 0.8).

•

```
> (metr.temp <- metr.temp.init <- -mean.delta.fit/log(0.8))

[1] 0.034569

> rm(mean.delta.fit)
> alpha <- 0.99
```

At this temperature the probability of accepting a poorer scheme is 0.8 to begin with; this will be reduced by 0.99 in each iteration.

2.5 The main SSA loop: one time

In this section we'll break down the SSA algorithm for one pass, to see how it works. In the next section we'll run the loop multiple times to optimize the sampling scheme.

Task 17 : Initialize the step number to 1.

•

```
> step <- 1
```

2.5.1 Maximum distance to move a point

The first step in the loop is to compute a maximum distance to move a point. This should be reduced as the simulation proceeds, otherwise there will be many long jumps that almost certainly will not be accepted, since the temperature of the system is cooling and the distribution of points is getting closer to optimal.

We use the empirical formula:

$$s_{\max} = d_{\max} \cdot e^{-(s-1)/h} \quad (5)$$

where:

- s_{\max} is the maximum shift for this step;

- d_{\max} is the maximum shift for the first step, some proportion of the distance across the bounding box computed above (§2.1) as `max.dist`;
- s is the step number;
- h is an empirical factor that controls the decay.

Note that at the first step $e^{-(s-1)/h} = e^{-(0/h)} = 1$.

The empirical decay factor h is set to 256; a lower decay factor would lead to a larger negative exponent, a larger $e^{-(s-1)/h}$ and thus a faster decay. For example, at the second step the decay would be:

```
> exp(-1/256)

[1] 0.9961

> exp(-1/512)

[1] 0.99805

> exp(-1/128)

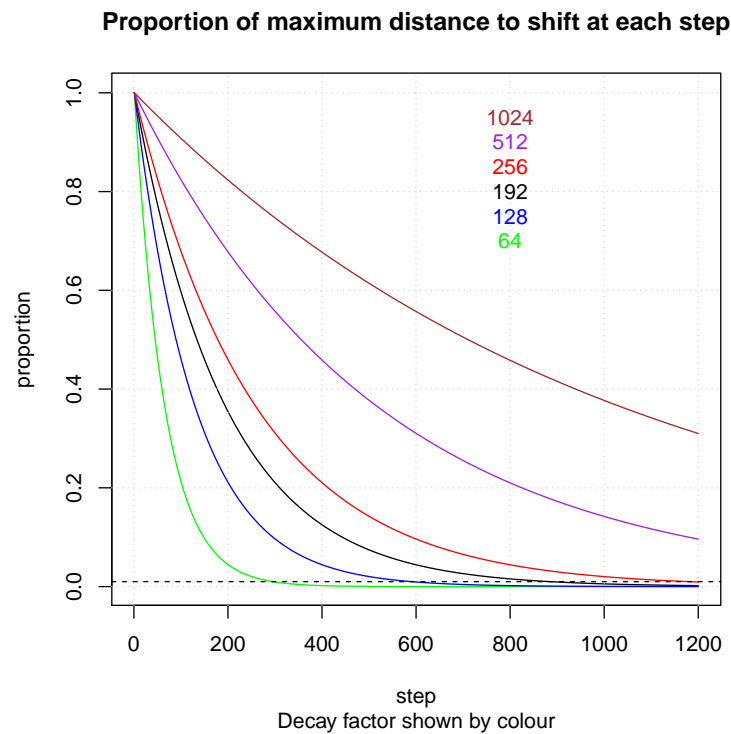
[1] 0.99222
```

for constants 256, 512, and 128 respectively.

Note: There seems to be no way except trial-and-error to set the decay factor. If it is too fast, points can not move far enough after a few steps; if too slow, there is a lot of wasted motion and many steps with no improvement in fitness, because many of the proposed shifts are too far and end up too close to other points, rather than adjusting a point that is more-or-less in the right place.

Task 18 : Visualize this decay function for several decay factors. •

```
> plot(1:1200, exp(-(0:1199)/192), type="l",
+      xlab="step", ylab="proportion",
+      main="Proportion of maximum distance to shift at each step",
+      sub="Decay factor shown by colour")
> grid()
> lines(1:1200, exp(-(0:1199)/64), col="green")
> lines(1:1200, exp(-(0:1199)/128), col="blue")
> lines(1:1200, exp(-(0:1199)/256), col="red")
> lines(1:1200, exp(-(0:1199)/512), col="purple")
> lines(1:1200, exp(-(0:1199)/1024), col="brown")
> text(800, .7, "64", col="green")
> text(800, .75, "128", col="blue")
> text(800, .8, "192")
> text(800, .85, "256", col="red")
> text(800, .9, "512", col="purple")
> text(800, .95, "1024", col="brown")
> abline(h=0.01, lty=2)
```

Task 19 : Compute the step at which the movement is only 1% of the initial maximum for several decay factors: •

```
> for (i in c(64, 128, 192, 256, 512, 1024)) print(floor(-i *
+   log(1/100) + 1))

[1] 295
[1] 590
[1] 885
[1] 1179
[1] 2358
[1] 4716
```

To ensure that points can move far enough even if the random scheme, by chance, placed all the points in a small portion of the study area, we set the beginning maximum shift in both x and y to $\pm\sqrt{2}/6$ of the diagonal distance across the bounding box. For a square bounding box, this is just another way to write one-third of the linear dimension (side) of the box. This is also an empirical factor, which could be reduced if there were many points. With a smaller value points can still move across the box, but in several steps, as long as the cooling is not too fast.

```
> (max.shift <- round((max.dist * sqrt(2)/6), prec))

[1] 0.333
```

This is the maximum shift for the first step; in later steps it will be a shorter distance.

Note: Another way to set the maximum shift is as some function of the point density.

2.5.2 Moving one point

Task 20 : Select a point to move and save its coördinates. •

This is a random selection from the list of points, using the `sample` “random sample” function. Again we start the random number generator at a known position, so your results will match:

```
> set.seed(621)
> (pt <- sample(1:n.pts, 1))

[1] 8

> pt.old <- coordinates(scheme)[pt, ]
```

Task 21 : Compute the proposed shift in x and y, using the `runif` function to select a uniform random number on the interval from the maximum negative to the maximum positive shift for this step, and checking that the proposed shift is within the study area. •

To ensure that the shifted point is within the study area we use the `over` “overlay” method, which returns the polygon number containing a point, or NA if it is outside any polygon. We check for this with the `is.na` function. We put the shift and test in a `repeat` loop and `break` out of the loop when the shift is within the study area.

```
> set.seed(621)
> repeat {
+   xnew <- round(pt.old[1] + runif(1, min = -max.shift,
+     max = max.shift), prec)
+   ynew <- round(pt.old[2] + runif(1, min = -max.shift,
+     max = max.shift), prec)
+   pt.new <- data.frame(x = xnew, y = ynew)
+   coordinates(pt.new) <- ~x + y
+   if (!is.na(over(pt.new, p)))
+     break
+ }
```

Task 22 : Construct a proposed new scheme, by replacing the selected point with the proposed shift, and compute its fitness. •

```
> scheme.try <- scheme
> scheme.try@coords[pt, ] <- c(xnew, ynew)
> (fitness.new <- obj.k(rbind(scheme.try, fix.pts), mean))

[using ordinary kriging]
[1] 0.81139
```

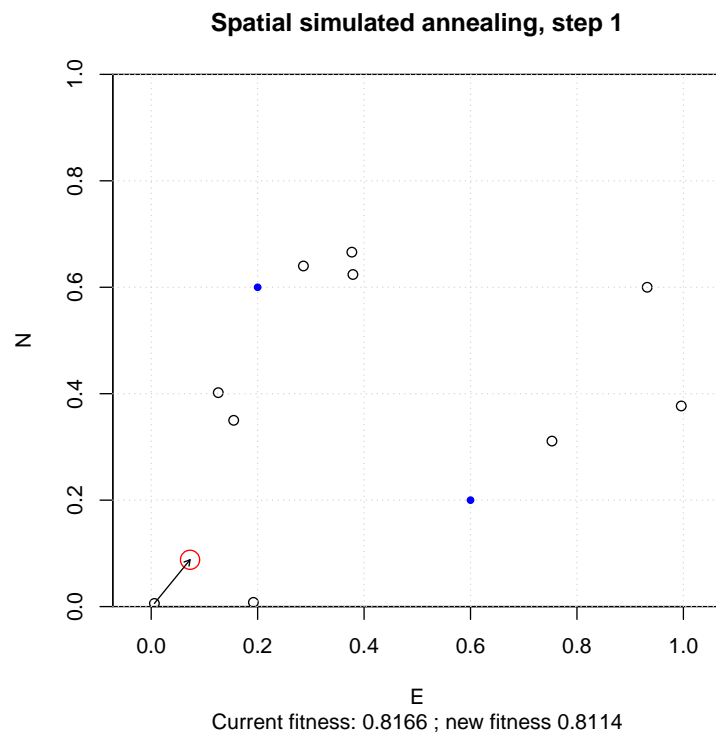
Q4 : Does this proposed shift give a better or worse sampling scheme?
Jump to A4 •

2.5.3 Plotting a changed scheme

It's instructive to see how the scheme evolves.

Task 23 : Plot the current scheme, the proposed changed point, with an arrow from the point to be shifted. •

```
> plot(coordinates(scheme), xlim = c(bbox(p)[1, "min"],
+   bbox(p)[1, "max"]), ylim = c(bbox(p)[2, "min"], bbox(p)[2,
+   "max"]), xaxs = "i", yaxs = "i", xlab = "E", ylab = "N",
+   main = paste("Spatial simulated annealing, step",
+   step), asp = 1, sub = paste("Current fitness:",
+   round(fitness, 4), "; new fitness", round(fitness.new,
+   4)))
> grid()
> points(coordinates(fix.pts), col = "blue", pch = 20)
> points(xnew, ynew, col = "red", cex = 2)
> arrows(pt.old[1], pt.old[2], xnew, ynew, length = 0.05)
```



Q5 : Looking at the geometry of the original and proposed point configurations, explain the change in fitness, i.e., mean kriging prediction variance.
Jump to A5 •

2.5.4 Accepting or rejecting the new scheme

With the current and new fitnesses, we accept the new scheme:

1. if it is better;
2. if it is worse, if a uniform random number is less than the probability to accept a worse scheme, i.e. the Metropolis criterion:

Task 24 : Decide whether to accept or reject this proposed scheme. •

```
> if (fitness.new < fitness) {
+   scheme <- scheme.try
+   fitness <- fitness.new
+   print("Better scheme, accepted")
+ } else if (runif(1) < (metr.temp * exp(fitness - fitness.new))) {
+   scheme <- scheme.try
+   fitness <- fitness.new
+   print("Worse scheme, accepted")
+ } else print("Worse scheme, rejected")

[1] "Better scheme, accepted"
```

This process should be repeated “many” times to converge on a solution.

2.6 The main SSA loop: iterations

Here we put the pieces from the previous section together in a loop, and accumulate the results. To illustrate the concept in a reasonable time we will only do 300 steps, although we will not have reached an optimum. We use the `system.time` function to see how much.

```
> n.steps <- 300
> fits <- matrix(nrow = n.steps, ncol = 4)
> colnames(fits) <- c("actual", "proposed", "max.shift",
+   "temperature")
```

Now the main loop:

```
> (time <- system.time(for (step in 1:n.steps) {
+   max.shift <- max.dist * 0.5 * round(exp(-(step -
+   1)/256), prec)
+   metr.temp <- metr.temp * alpha
+   pt <- sample(1:n.pts, 1)
+   pt.old <- coordinates(scheme)[pt, ]
+   repeat {
+     xnew <- round(pt.old[1] + runif(1, min = -max.shift,
+     max = max.shift), prec)
+     ynew <- round(pt.old[2] + runif(1, min = -max.shift,
+     max = max.shift), prec)
+     pt.new <- data.frame(x = xnew, y = ynew)
+     coordinates(pt.new) <- ~x + y
+     if (!is.na(over(pt.new, p)))
+       break
+   }
+ })
```

```

+     scheme.try <- scheme
+     scheme.try@coords[pt, ] <- c(xnew, ynew)
+     fitness.new <- obj.k(rbind(fix.pts, scheme.try),
+       mean)
+     if (fitness.new < fitness) {
+       scheme <- scheme.try
+       fitness <- fitness.new
+     }
+     else if (runif(1) < (metr.temp * exp(fitness - fitness.new))) {
+       scheme <- scheme.try
+       fitness <- fitness.new
+     }
+     fits[step, "actual"] <- fitness
+     fits[step, "proposed"] <- fitness.new
+     fits[step, "max.shift"] <- max.shift
+     fits[step, "temperature"] <- metr.temp
+   })

> print(time)

      user  system elapsed
28.963    2.336   31.372

```

2.7 Showing the evolution of the scheme

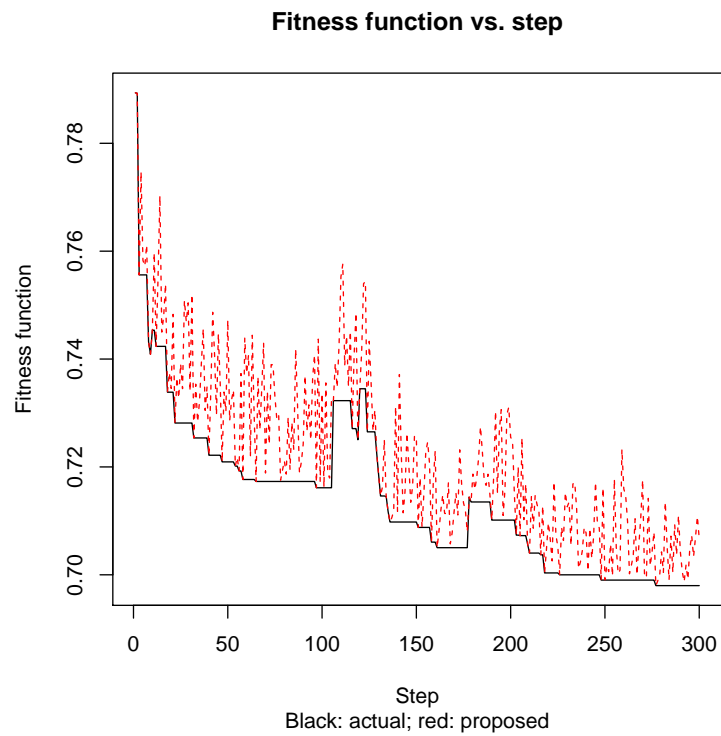
Recall that we saved the actual and proposed fitness for each step during the main SSA loop; so we can show how fitness evolved.

Task 25 : Display a graph of the fitness vs. step number. •

```

> plot(1:n.steps, fits[, "actual"], xlab="Step",
+     ylab="Fitness function", type="l",
+     main="Fitness function vs. step",
+     sub="Black: actual; red: proposed",
+     ylim=c(min(fits[,1]), max(fits[,2])))
> lines(1:n.steps, fits[, "proposed"], type="l",
+     col="red", lty=2, pch=3)

```

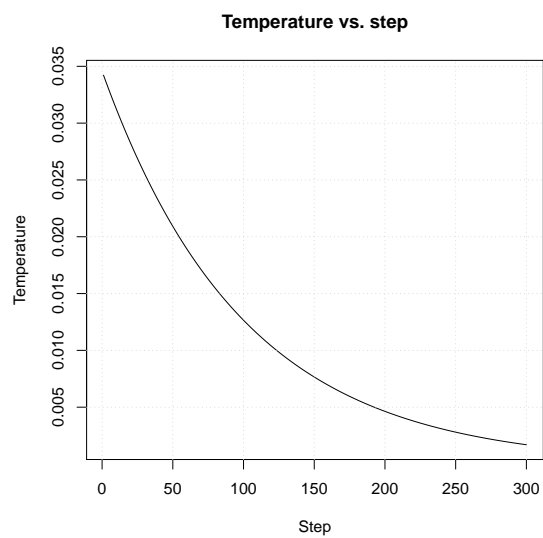


Q6 : *Describe the evolution of the fitness.*

Jump to A6 •

Task 26 : Plot the evolution of the temperature. •

```
> plot(1:n.steps, fits[, "temperature"],
+      xlab="Step", ylab="Temperature",
+      type="l", main="Temperature vs. step")
> grid()
```

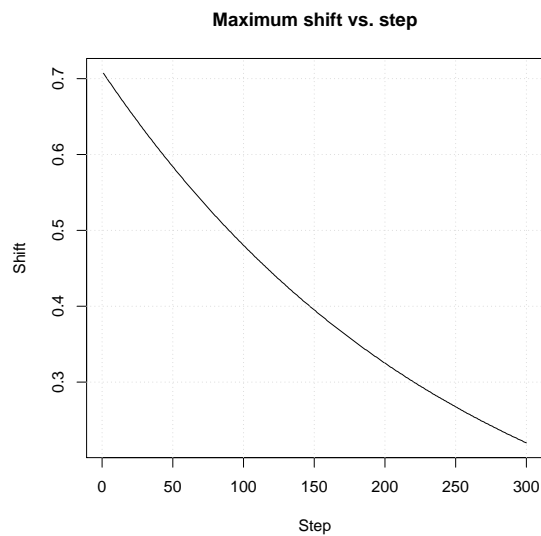


The lowering temperature affects the maximum shift.

Task 27 : Plot the evolution of the maximum shift. •

```
> plot(1:n.steps, fits[, "max.shift"], xlab = "Step", ylab = "Shift",
+      type = "l", main = "Maximum shift vs. step")
> grid()
> print(paste("Maximum shift after", n.steps, ":", round(fits[n.steps,
+      "max.shift"], 4)))

[1] "Maximum shift after 300 : 0.2199"
```



Q7 : Comparing the maximum shift now allowed with the sample point configuration after 200 steps, is this shift too large (wasted motion), too small (won't reach an optimum), or about right? Jump to A7 •

Task 28 : Display the final scheme, with the optimality criterion (kriging variance) as background: •

```
> print(scheme)

      coordinates z
1 (0.628, 0.836) 0
2 (0.105, 0.319) 0
3 (0.229, 0.122) 0
4 (0.126, 0.873) 0
5 (0.85, 0.17) 0
6 (0.302, 0.787) 0
7 (0.858, 0.546) 0
8 (0.461, 0.349) 0
9 (0.827, 0.845) 0
10 (0.558, 0.511) 0
```

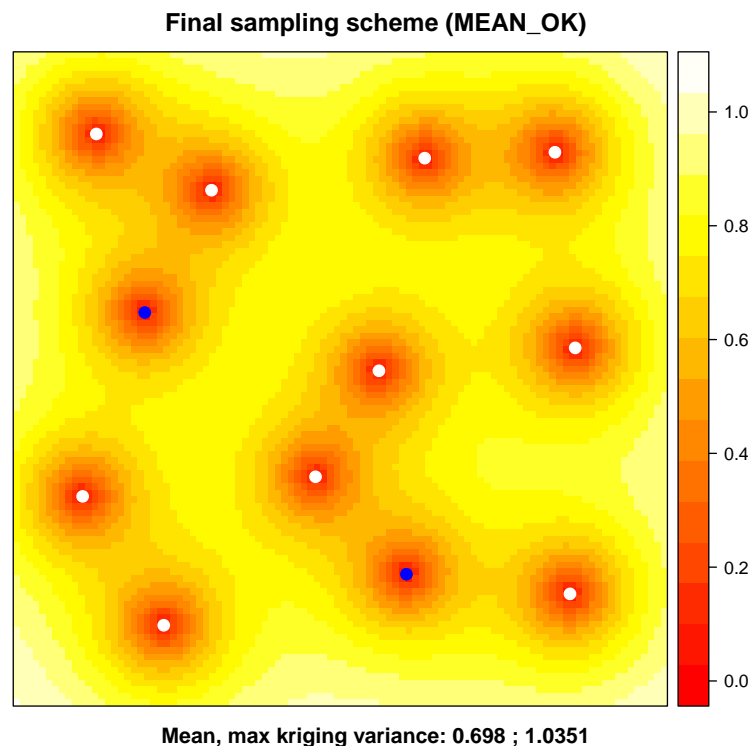
```

> k <- krige(z ~ 1, loc=rbind(fix.pts,scheme),
+           model=vm, newdata=grid)

[using ordinary kriging]

> gridded(k) <- T; fullgrid(k) <- T
> pts.s <- list("sp.points", scheme, col="white",
+             pch=20, cex=1.6)
> pts.f <- list("sp.points", fix.pts, col="blue",
+             pch=20, cex=1.6)
> print(spplot(k, zcol="var1.var",
+             col.regions=heat.colors(64),
+             main=paste("Final sampling scheme (MEAN_OK)"),
+             sub=paste("Mean, max kriging variance:",
+             round(mean(k$var1.var),4),";",
+             round(max(k$var1.var),4)),
+             sp.layout = list(pts.s, pts.f)
+             ))

```



Q8 : *How much has this configuration changed from the original random configuration? Does it appear optimal? Why or why not?* Jump to A8 •

Task 29 : Clean up from this section. •

```

> rm(list = ls())

```


2.8 Answers

A1 : The nugget affects the kriging prediction variance uniformly for the whole area, for any scheme. The partial sill affects the kriging prediction variance non-uniformly over the area, but the ranking of different schemes will not change. Hence only the model form and range are important in comparing schemes.

However, [10] showed that the **structural sill / nugget ratio** does affect the scheme: as the nugget becomes an increasing proportion of the total sill the scheme remains more-or-less random, so that any initial configuration would not be changed.

So we use a zero nugget, to force the scheme away from randomness. [Return to Q1](#) •

A2 : The fitness function only depends on the kriging prediction variance, and this does not depend on the data values. [Return to Q2](#) •

A3 : This is the mean kriging prediction variance of the sampling scheme (including the fixed points) over the discretized study area; the units of measure are squares of some undefined unit, since the variable z is synthetic. [Return to Q3](#) •

A4 : If the new value of the mean kriging variance is lower than the original, the scheme is better; and vice-versa. [Return to Q4](#) •

A5 : If the shifted point is close to one or more other points, it had some spatial dependence with these. If in the changed scheme, it moves to a more “unoccupied” area, the kriging variance in the area now “covered” by the shifted point will be lower; the variance near the original point will be higher but not too much, because of the other nearby points. So the overall average should be lower (better).

If, by contrast, the shifted point “covered” an area more or less by itself and is then shifted into a more “occupied” area, the reverse will occur: the mean kriging variance of the overall scheme will increase. [Return to Q5](#) •

A6 : The scheme in general improves, with local jumps to poorer fitness to avoid a local optimum. [Return to Q6](#) •

A7 : The maximum shift now allowed is 0.2199 which is still almost 1/3 of the bounding box dimensions (1 by 1). Looking at the final configuration, it seems some points should be shifted by 0.1 to 0.2, so this is a reasonable value. It indicates that the decay constant $\alpha = 0.99$ is about right. [Return to Q7](#) •

A8 : The scheme has changed considerably. The twelve points (two fixed, ten moveable) are spread more evenly. But they do not yet seem to yield a uniform map. [Return to Q8](#) •

3 SSA optimization for Ordinary Kriging: application

Now that we see how this works, we can run the SSA for a realistic number of steps. You can experiment to determine how many iterations; here we use a **stopping criterion**, which is that the fitness does not improve for a set number of iterations.

3.1 A function for SSA

Task 30 : Write a function to optimize a sampling scheme by SSA. •

The main loop developed in the previous section is re-written as a function (using the **function** method) with some required arguments and some options; this allows us to call it for different scenarios without changing the code.

Several of the **arguments** are inputs:

- `s.init` : the initial sampling scheme, as a **SpatialPointsDataFrame** including a (dummy) variable named `z`;
- `s.area` : the (set of) polygons comprising the study area, as a **SpatialPolygonsDataFrame**;
- `s.model` : the variogram model, as produced by the `vgm` function;
- `s.fix.pts` : any fixed points, also as a **SpatialPointsDataFrame** including a (dummy) variable named `z`, otherwise undefined.

Others are options that control model behaviour:

- `s.f` : the fitness criterion function, default `mean`;
- `s.res` : resolution of the discretization of `s.area` for computing the fitness by kriging; this is the number of divisions of the bounding box in each dimension; distance units as the variogram;
- `s.decay` : the distance decay factor, default `256`;
- `s.max.shift` : initial maximum shift, default $\sqrt{2}/4$
- `s.alpha` : the cooling parameter, default `0.99`;
- `s.prob` : the initial probability of accepting a worse scheme, default `0.8`;
- `eq.steps` : stopping criterion: how many steps of unchanged fitness, default `200`;
- `max.steps` : stopping criterion: how many steps total, default undefined; this could be used to call the function and interrupt it before it reaches an optimum;
- `s.prec` : precision of numeric comparisons of fitness; this prevents many steps with almost identical fitness at the end of the run. No default; if not specified it is computed as $1/1000$ of the initial fitness.

The function **returns** a list containing:

`scheme` : the final scheme, as a `SpatialPointsDataFrame`;
`metr.temp` : the initial Metropolis temperature;
`s.prec` : the fitness matching precision.

As a side effect, it plots the evolution of the fitness.

```

> ssa.ok <- function(s.init, s.area, s.res,
+                   s.model, s.fix.pts=NA, s.f=mean, s.decay=256,
+                   s.max.shift = sqrt(2)/4,
+                   s.alpha=0.99, s.prob=0.8,
+                   eq.steps=200,
+                   max.steps=NA, s.prec=NA) {
+
+
+   # helper function: compute fitness by kriging variance
+   # interpolation `grid' and variogram model `s.model'
+   # are in callers environment
+   # data field must be named `z'
+   obj.k <- function(pts, f) {
+     k <- krige(z ~ 1, loc=pts, model=s.model,
+               newdata=grid, debug.level=0)
+     return(f(k$var1.var))
+   }
+
+   # helper function: compute initial temperature
+   metr.temp.init <- function(n.steps) {
+     s <- s.init # initial `fitness' in environ
+     max.shift <- max.dist*sqrt(2)/4
+     sum.fit <- 0; i <- 0
+     while (i < n.steps) {
+       pt <- sample(1:n.pts, 1)
+       pt.old <- coordinates(scheme)[pt,]
+       s.try <- NA
+       while (is.logical(s.try)) {
+         xnew <- pt.old[1] + runif(1, min=-max.shift/2,
+                                max=max.shift/2)
+         ynew <- pt.old[2] + runif(1, min=-max.shift/2,
+                                max=max.shift/2)
+         pt.new <- data.frame(x = xnew, y = ynew)
+         coordinates(pt.new) <- c(1,2)
+         # over() requires identical CRS, note order
+         proj4string(pt.new) <- proj4string(s.area)
+         if (!is.na(over(pt.new, s.area))) s.try <- s
+       }
+       s.try@coords[pt,] <- c(xnew, ynew)
+       # fitness function in environ
+       if (is.logical(s.fix.pts)) pts <- s.try
+       else pts <- rbind(s.fix.pts, s.try)
+       # check new fitness, only use if worse
+       fit.new <- obj.k(pts, s.f)
+       if (fit.new > fitness) {
+         sum.fit <- sum.fit + fit.new
+         i <- i+1 }
+     } # note change of sign to ensure negative sum
  
```

```

+       return((fitness-(sum.fit/n.steps))/log(s.prob))
+   }
+
+   ## main body
+   ## 1 - setup
+   # 1.1 - set up maximum distance
+   max.dist <- sqrt(((bbox(s.area)[1,"max"]
+                     - bbox(s.area)[1,"min"])^2)
+                 + ((bbox(s.area)[2,"max"]
+                     - bbox(s.area)[2,"min"])^2))
+   n.pts <- length(s.init$z)
+
+   # 1.2 - set up interpolation grid -- may not be full
+   #       so leave as SpatialPoints, do not promote to SpatialGrid
+   grid <- spsample(s.area, n=s.res^2, type="regular")
+   if (is.logical(s.fix.pts)) pts <- s.init
+   else pts <- rbind(s.fix.pts, s.init)
+   fitness <- obj.k(pts, s.f) # initial fitness
+
+   # 1.3 - if precision was not specified, set as 3rd significant
+   #       figure of initial fitness
+   i <- 0
+   while (is.na(s.prec)) {
+     if (fitness%%(10^-i) != 0) s.prec=i+4 else i <- i + 1
+   }
+
+   # 1.4 - initialize temperature
+   metr.temp <-
+     ret.metr.temp.init <-
+     metr.temp.init(128); # this is enough for a rough guess
+
+   # 1.5 - set up to record evolution of fits
+   fits <- matrix(nrow = 0, ncol=4)
+   colnames(fits) <- c("actual","proposed","max.shift","temperature")
+   this.step <- 0; equal.steps <- 0;
+   s <- s.init;
+
+   ## 2 - main loop
+   # 2.1 get a proposed new point in the study area
+   repeat {
+     max.shift <- (max.dist*s.max.shift)*
+       exp(-this.step/s.decay)
+     metr.temp <- metr.temp * s.alpha
+
+     pt <- sample(1:n.pts, 1)
+     pt.old <- coordinates(s)[pt,]
+     repeat {
+       xnew <- round(pt.old[1] + runif(1, min=-max.shift,
+                                     max=max.shift),prec)
+       ynew <- round(pt.old[2] + runif(1, min=-max.shift,
+                                     max=max.shift),prec)
+       pt.new <- data.frame(x = xnew, y = ynew)
+       coordinates(pt.new) <- ~ x + y
+       proj4string(pt.new) <- proj4string(p)
+       if (!is.na(over(pt.new, p))) break

```

```

+     }
+
+     # 2.2 increment step when we have a proposed point
+     this.step <- this.step + 1;
+     s.try <- s;
+     # update the selected point's coordinates
+     s.try@coords[pt,] <- c(xnew, ynew);
+     # 2.3 new fitness, to the desired precision
+     if (is.logical(s.fix.pts)) pts <- s.try
+     else pts <- rbind(s.fix.pts, s.try)
+     fitness.new <- round(obj.k(pts, s.f), s.prec);
+
+     # 2.4 acceptance criteria; maybe update scheme
+     if (fitness.new < fitness) {
+       s <- s.try;
+       fitness <- fitness.new;
+       equal.steps <- 0
+     } else if (runif(1) < (metr.temp
+       * exp(fitness - fitness.new))) {
+       s <- s.try;
+       fitness <- fitness.new;
+       equal.steps <- 0
+     } else { equal.steps <- (equal.steps + 1) }
+
+     # 2.5 save record of fits
+     fits <- rbind(fits, c(fitness, fitness.new,
+       max.shift, metr.temp))
+
+     # 2.6 check stopping criteria
+     if (equal.steps >= eq.steps) break;
+     if (!is.na(max.steps) & (this.step >= max.steps)) break;
+   }
+
+   # 3 - plot the fitness vs. step
+   plot(1:length(fits[,1]), fits[, "actual"],
+     xlab="Step", ylab="Fitness function", type="l",
+     main="Fitness function vs. step",
+     sub="Black: actual; red: proposed",
+     ylim=c(min(fits[,1]), max(fits[,2])))
+   lines(1:length(fits[,1]), fits[, "proposed"],
+     type="l", col="red", lty=2)
+
+   # 4 - return the final sampling scheme
+   # and the computed annealing parameters
+   return(list(scheme=s, metr.temp=ret.metr.temp.init, s.prec=s.prec))
+ }

```

Note: Some programming notes for R aficionados:

- The argument `s.fix.pts` may be undefined, with default value `NA`, or a set of points as a `SpatialPointsDataFrame`. If defined, the fixed points must be appended to the moveable points in order to re-compute the kriging variance after each proposed move. But it is only possible to append objects of like type with `rbind`, so the data type must be tested. The `NA` value is of type `logical`, which can be tested for with the `is.logical` function.

- The `debug.level` argument when set to 0 suppresses all printed information during kriging; this avoids many pages of repeated [using ordinary kriging].

3.1.1 Saving the function to a file for later execution

The `ssa_ok` function can be saved either as text or as an R object, for later use.

Task 31 : Save the `ssa_ok` function as both a text R script and as an R object file. •

To save as a text file, we need to also include the command to load into a workspace, i.e., as an R script which defines the function when run. We use the `file` function to establish a file handle and open a connection to the file. We then send output to this file.

We first write a text string `"ssa_ok.R <- "` with the `cat` “catenate” function; this is the assignment operator and the name of the object. We then write the function definition with the `print` function, but to place it in the file as it would be displayed on the console we first need to redirect output with `capture.output`, specifying the file handle of the open file. A call to `close` with the file handle closes the file.

```
> tmp <- file("ssa_ok.R", open = "wt")
> cat("ssa.ok <- ", file = tmp)
> capture.output(print(ssa.ok), file = tmp)
> close(tmp)
```

To get this function into your workspace, you would use the `source` function:

```
> source("ssa_ok.R")
```

To save the function in compiled form as an R object, we use the `save` function:

```
> save(ssa.ok, file = "ssa_ok.RData")
```

To get this compiled function into your workspace, you would use the `load` function:

```
> load("ssa_ok.RData")
```

3.2 Initial sampling scheme

Task 32 : Set up an initial sampling scheme to be optimized. •

We set up so your results look the same as the ones here; you are welcome to try other starting schemes and fixed points.

First, the study area:

```

> p <- SpatialPolygonsDataFrame(
+   SpatialPolygons(
+     list(Polygons(
+       list(Polygon(
+         matrix(c(0,0, 0,1, 1,1, 1,0, 0,0),
+           byrow=T, nrow=5, ncol=2) # matrix
+       ) # Polygon
+     ) # list of Polygon
+     , ID=1 ) # Polygons, with identifier
+   ) # list of Polygons
+ ) # Spatial Polygons
+ , data=data.frame(id=1)) # SpatialPolygonsDataFrame

```

Then, the fixed points; as before we specify two of these.

```

> fix.pts <- data.frame(x = c(0.2, 0.6), y = c(0.6, 0.2),
+   z = 0)
> coordinates(fix.pts) <- ~x + y

```

Then, the initial sampling scheme of the new points. As before, we assume there is budget to make ten new observations.

We start from a **regular grid**, since we know from the OSSFIM approach of McBratney and Webster [6, 5] that a regular triangular grid gives the lowest mean kriging variance; a regular square grid is not much worse. Here we have constraints (the fixed points) so the grid will be modified during SSA.

The `spsample` method's `type` argument specifies the sampling scheme; we specify "hexagonal" to get a hexagonal lattice (i.e., a triangular grid). Note that in this case, and also with `type` as "regular" (square grid), the requested number of points (`n` argument) is only approximate, because the regular pattern has to fit evenly within the bounding polygon, and the starting point (random seed) may place some points outside the polygon. So we have to experiment to make sure we have ten new points. The “experiment” is with an `repeat` loop that stops (using the `break` statement) when the required number of points have been placed.

```

> prec <- 4
> n.pts <- 10
> set.seed(619)
> repeat {
+   try <- spsample(p, n = n.pts, type = "hexagonal")
+   if (length(coordinates(try)[, 1]) == n.pts)
+     break
+ }
> scheme <- SpatialPointsDataFrame(try, data.frame(z = rep(0,
+   n.pts)))
> scheme@coords <- round(scheme@coords, prec)
> rm(try)

```

Task 33 : Plot the starting scheme. •

```

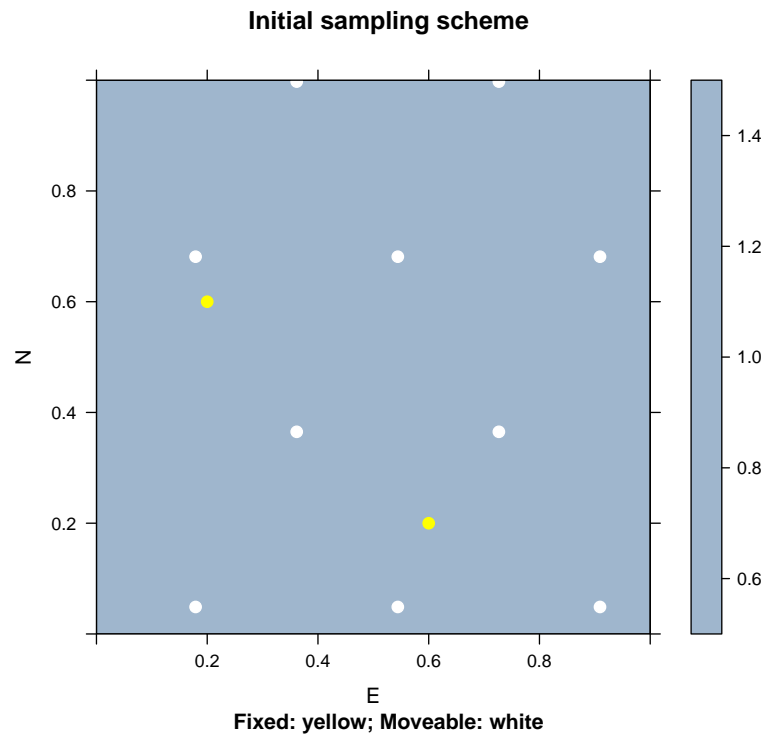
> print(spplot(p, xlab="E", ylab="N",
+   col.regions="slategray3",

```

```

+         main="Initial sampling scheme",
+         sub="Fixed: yellow; Moveable: white",
+         scales=list(draw=T),
+         sp.layout = list(list("sp.points",
+                               scheme, col="white", pch=20, cex=1.6),
+                           list("sp.points", fix.pts, col="yellow",
+                               pch=20, cex=1.6))
+     ))

```



Q9 : Describe the initial scheme. Which points do you think will move during optimization, and to where? Jump to A9 •

3.3 Objective function: minimize the mean kriging variance

Task 34 : Optimize this starting scheme, minimizing the **mean kriging variance** as the optimization criterion. •

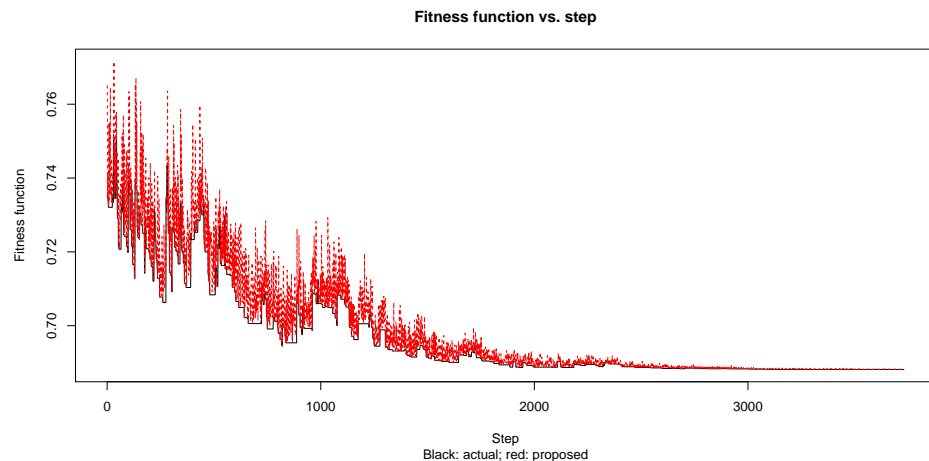
We choose to stop after 200 steps without improvement, cooling more slowly, accept a higher probability of accepting a worse scheme, and with a slower distance decay, than the default. All of these must be set by trial-and-error.

Note: The variogram model is specified as an argument to `ssa.ok`; only the model form and distance influence the scheme. Here we use an exponential model and 1/3 of the maximum distance across the bounding box; in an actual application the variogram would be estimated from previous studies.

Recall that the distance parameter of an exponential variogram model is 1/3 of the effective range.

This may take quite some time; we check this with `system.time`.

```
> max.dist <- sqrt(((bbox(p)[1, "max"] - bbox(p)[1, "min"])^2) +
+   ((bbox(p)[2, "max"] - bbox(p)[2, "min"])^2))
> vm <- vgm(psil1 = 1, model = "Exp", range = max.dist/(3 *
+   3), 0)
> time <- system.time(tmp <- ssa.ok(s.init = scheme, s.fix.pts = fix.pts,
+   s.area = p, s.res = 100, s.model = vm, s.f = mean,
+   s.probab = 0.9, s.alpha = 0.999, s.decay = 1024, eq.steps = 200))
```



```
> print(time)

      user  system elapsed
241.162    8.413  253.184

> print(paste("Precision of fitness matching:", tmp[["s.prec"]],
+   "decimal places"))

[1] "Precision of fitness matching: 5 decimal places"

> print(paste("Initial Metropolis temperature:", round(tmp[["metr.temp"]],
+   4)))

[1] "Initial Metropolis temperature: 0.0753"

> s.final <- tmp[["scheme"]]
```

On the test system, this took about 4 minutes.

Task 35 : Show and print the final configuration. •

```
> print(s.final)

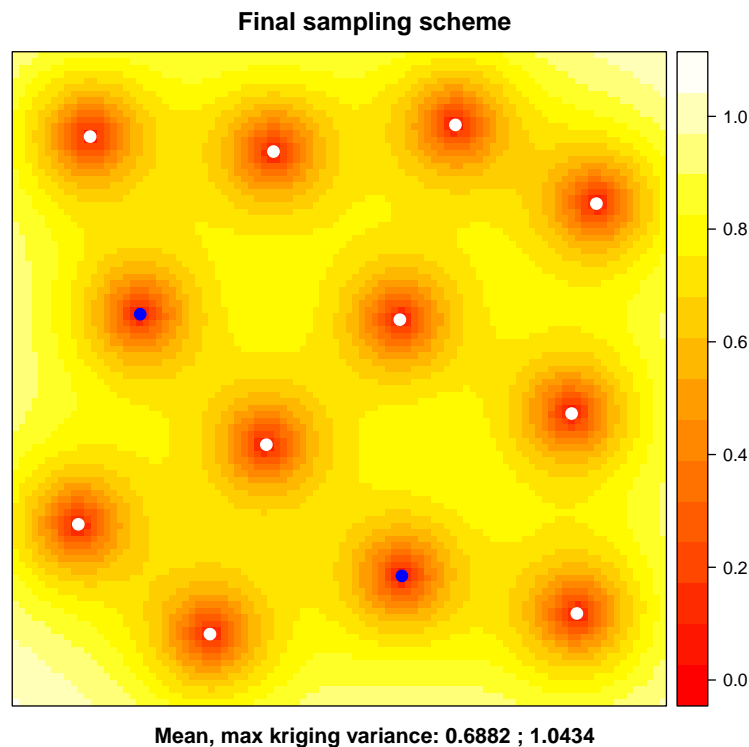
      coordinates z
1 (0.3928, 0.4006) 0
2 (0.5968, 0.5917) 0
3 (0.8674, 0.1426) 0
4 (0.3066, 0.1111) 0
5 (0.1053, 0.2789) 0
```

```

6 (0.8973, 0.7691) 0
7 (0.4037, 0.8487) 0
8 (0.1234, 0.8715) 0
9 (0.6818, 0.8894) 0
10 (0.8594, 0.4482) 0

> grid <- spsample(p, n=100^2, type="regular")
> gridded(grid) = T; fullgrid(grid) = T
> k <- krige(z ~ 1, loc=rbind(fix.pts, s.final),
+           model=vm, newdata=grid, debug.level=0)
> gridded(k) = T; fullgrid(k) = T
> pts.s <- list("sp.points", s.final, col="white",
+             pch=20, cex=1.6)
> pts.f <- list("sp.points", fix.pts, col="blue",
+             pch=20, cex=1.6)
> print(spplot(k, zcol="var1.var",
+             col.regions=heat.colors(64),
+             main=paste("Final sampling scheme"),
+             sub=paste("Mean, max kriging variance:",
+             round(mean(k$var1.var),4),";",
+             round(max(k$var1.var),4)),
+             sp.layout = list(pts.s, pts.f)
+             ))
> rm(grid, k, pts.s, pts.f)

```



In §3.5 we will compare this result to that obtained from another starting configuration, so now we save the result.

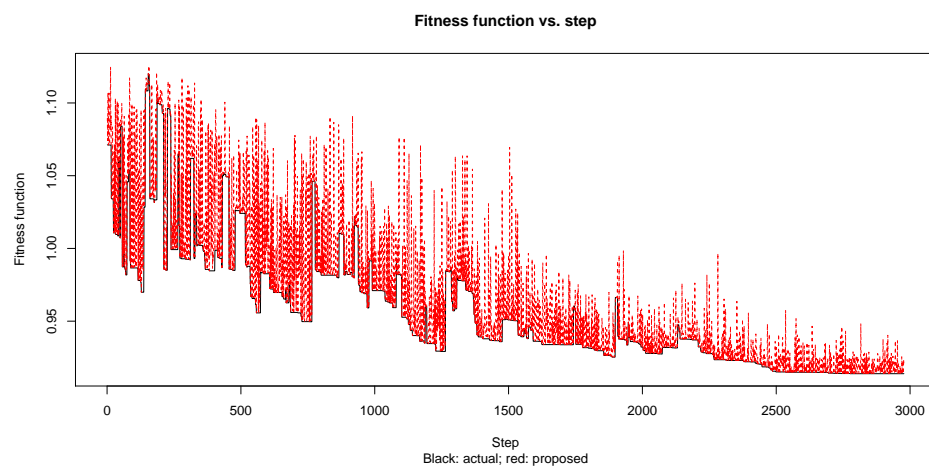
```
> s.final.mean.1 <- s.final
```

3.4 Objective function: minimize the maximum kriging variance

Now we see the effect of using the **maximum** kriging variance (MAX_OK), instead of the **mean** (MEAN_OK), as the fitness criterion. The setup does not need to be repeated, just the optimization. We start from the same initial configuration, and again use `system.time` to see how long this takes.

Task 36 : Optimize the sampling scheme, minimizing the **maximum kriging variance** as the optimization criterion. •

```
> time <- system.time(tmp <- ssa.ok(s.init = scheme, s.fix.pts = fix.pts,
+   s.area = p, s.res = 100, s.model = vm, s.f = max,
+   s.prob = 0.9, s.alpha = 0.999, s.decay = 1024, eq.steps = 200))
```



```
> print(time)

      user  system elapsed
197.60    6.84   205.83

> print(paste("Precision of fitness matching:", tmp[["s.prec"]],
+   "decimal places"))

[1] "Precision of fitness matching: 4 decimal places"

> print(paste("Initial Metropolis temperature:", round(tmp[["metr.temp"]],
+   4)))

[1] "Initial Metropolis temperature: 0.0595"

> s.final <- tmp[["scheme"]]
```

On the test system, this took about 3 minutes.

Task 37 : Show and print the final configuration. •

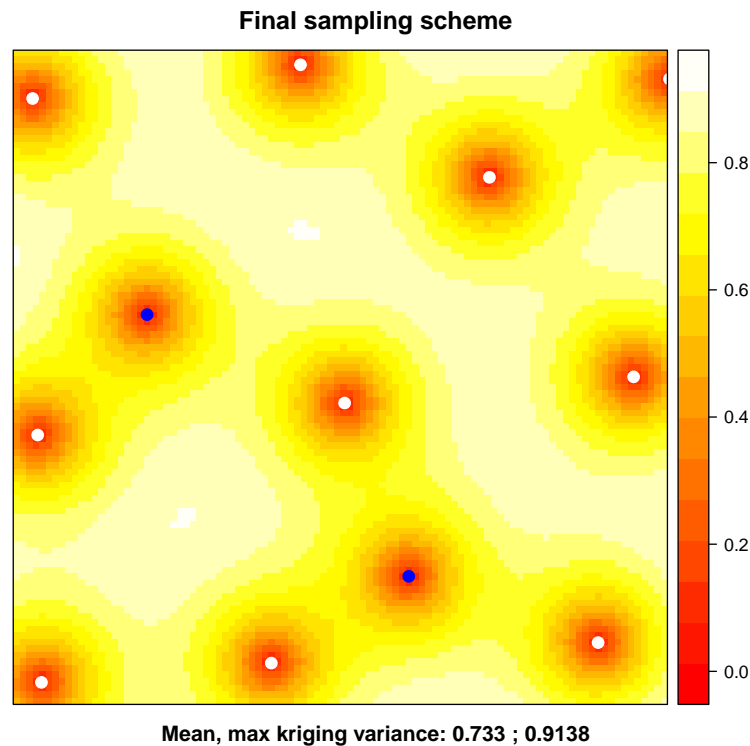
```
> print(s.final)
```

```

      coordinates z
1 (0.9437, 0.5049) 0
2 (0.8894, 0.0987) 0
3 (0.3899, 0.0674) 0
4 (0.0328, 0.416) 0
5 (0.025, 0.9304) 0
6 (0.0385, 0.0379) 0
7 (0.5019, 0.4649) 0
8 (0.9985, 0.9602) 0
9 (0.4345, 0.9819) 0
10 (0.7233, 0.8099) 0

> grid <- spsample(p, n=100^2, type="regular")
> gridded(grid) = T; fullgrid(grid) = T
> k <- krige(z ~ 1, loc=rbind(fix.pts, s.final),
+           model=vm, newdata=grid, debug.level=0)
> gridded(k) = T; fullgrid(k) = T
> pts.s <- list("sp.points", s.final, col="white",
+             pch=20, cex=1.6)
> pts.f <- list("sp.points", fix.pts, col="blue",
+             pch=20, cex=1.6)
> print(spplot(k, zcol="var1.var",
+             col.regions=heat.colors(64),
+             main=paste("Final sampling scheme"),
+             sub=paste("Mean, max kriging variance:",
+             round(mean(k$var1.var),4),";",
+             round(max(k$var1.var),4)),
+             sp.layout = list(pts.s, pts.f)
+             ))
> rm(grid, k, pts.s, pts.f)

```



Q10 : *How does this final scheme (criterion MAX_OK) differ from the previous final scheme (criterion MEAN_OK)? Explain the reasons for the difference.* *Jump to A10 •*

3.5 Optimizing from another starting configuration

If this procedure really achieves an optimum, it should be the same from any starting configuration. We try again with the **mean** (MEAN_OK) kriging variance, but from another initial scheme; again we use `set.seed` so your results will be the same. This time we experiment with a rectangular grid starting scheme; this must have 9 points (3x3) which we assign randomly; we then add a single random point (using `spsample` with `type` argument "random", and then the `rbind` "row bind" function to add it to the list of coördinates) to reach 10.

Note: In this square polygon we could of course assign the nine points to be as central as possible; but in the general case of irregular polygon(s) that would not be possible. So we let `spsample` randomly place the regular grid, and rely on SSA to adjust it.

Task 38 : Initialize and display another random configuration. •

Again we use `set.seed` so your results are the same as shown here; you are free to omit this and take another random scheme.

```

> set.seed(318)
> repeat {
+   try <- spsample(p, n = 9, type = "regular")
+   if (length(coordinates(try)[, 1]) == 9)
+     break
+ }
> try <- rbind(try, spsample(p, n = 1, type = "random"))
> scheme <- SpatialPointsDataFrame(try, data.frame(z = rep(0,
+   10)))
> scheme@coords <- round(scheme@coords, prec)
> rm(try)

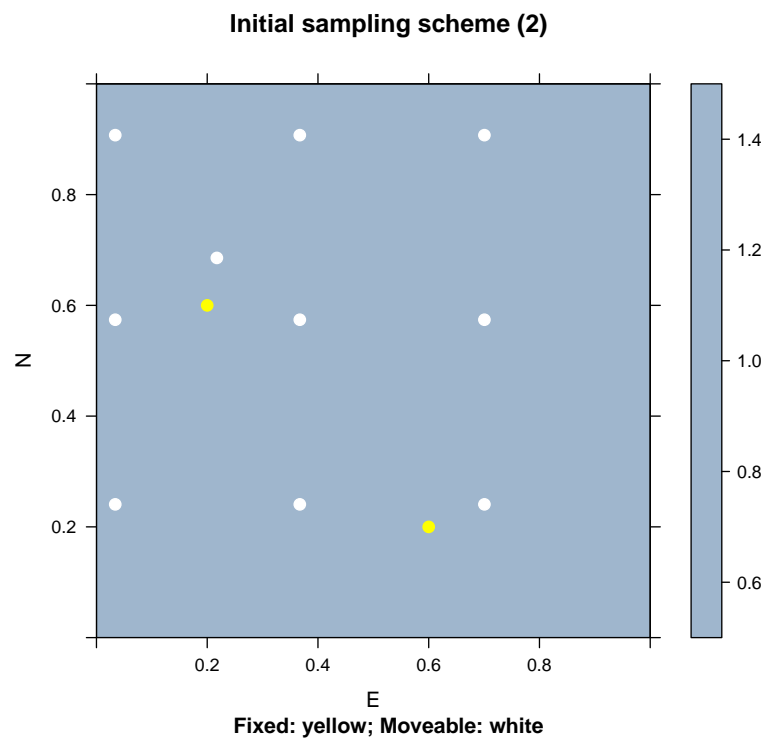
```

Here is the new starting scheme:

```

> print(spplot(p, xlab="E", ylab="N", col.regions="slategray3",
+   main="Initial sampling scheme (2)",
+   sub="Fixed: yellow; Moveable: white", scales=list(draw=T),
+   sp.layout = list(list("sp.points", scheme, col="white",
+   pch=20, cex=1.6),
+   list("sp.points", fix.pts, col="yellow",
+   pch=20, cex=1.6))
+   ))

```

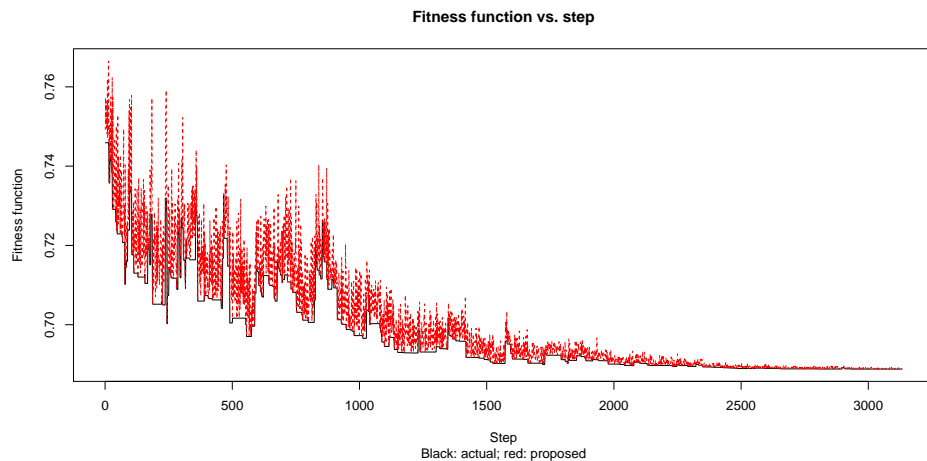


Task 39 : Optimize this scheme and display the results. •

```

> time <- system.time(tmp <- ssa.ok(s.init = scheme, s.fix.pts = fix.pts,
+   s.area = p, s.res = 100, s.model = vm, s.f = mean,
+   s.probab = 0.9, s.alpha = 0.999, s.decay = 1024, eq.steps = 200))

```



```
> print(time)

      user  system elapsed
205.102    7.334  215.151

> print(paste("Precision of fitness matching:", tmp[["s.prec"]],
+            "decimal places"))

[1] "Precision of fitness matching: 5 decimal places"

> print(paste("Initial Metropolis temperature:", round(tmp[["metr.temp"]],
+            4)))

[1] "Initial Metropolis temperature: 0.0646"

> s.final <- tmp[["scheme"]]
```

On the test system, this took about 4 minutes.

```
> print(s.final)

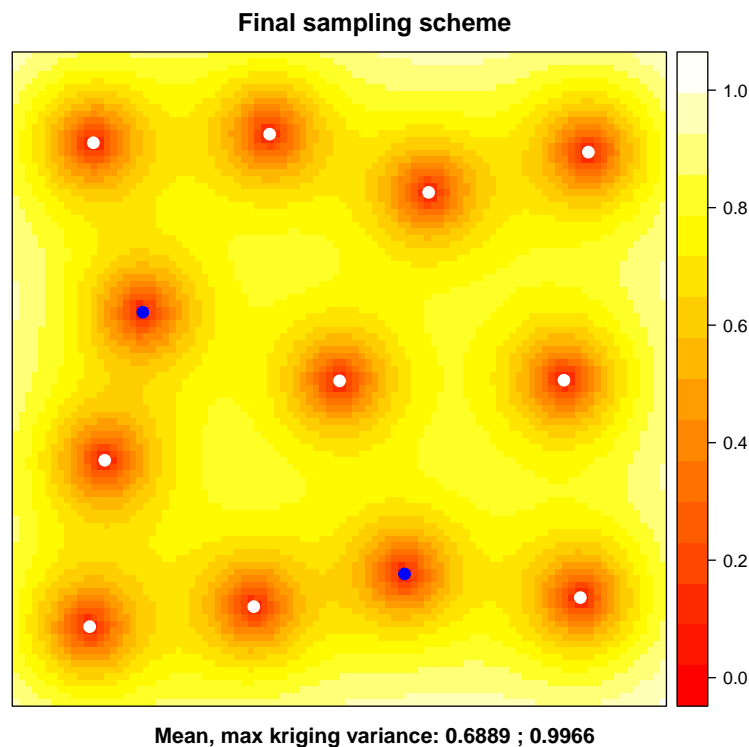
      coordinates z
1  (0.3696, 0.15) 0
2  (0.1417, 0.3737) 0
3  (0.1244, 0.859) 0
4  (0.881, 0.8447) 0
5  (0.3938, 0.8722) 0
6  (0.8688, 0.1638) 0
7  (0.5007, 0.4951) 0
8  (0.1188, 0.1194) 0
9  (0.6369, 0.7832) 0
10 (0.8437, 0.4961) 0

> grid <- spsample(p, n=100^2, type="regular")
> gridded(grid) = T; fullgrid(grid) = T
> k <- krige(z ~ 1, loc=rbind(fix.pts, s.final),
+           model=vm, newdata=grid, debug.level=0)
> gridded(k) = T; fullgrid(k) = T
> pts.s <- list("sp.points", s.final, col="white",
+              pch=20, cex=1.6)
```

```

> pts.f <- list("sp.points", fix.pts, col="blue",
+             pch=20, cex=1.6)
> print(spplot(k, zcol="var1.var",
+             col.regions=heat.colors(64),
+             main=paste("Final sampling scheme"),
+             sub=paste("Mean, max kriging variance:",
+                 round(mean(k$var1.var),4),";",
+                 round(max(k$var1.var),4)),
+             sp.layout = list(pts.s, pts.f)
+             ))
> rm(grid, k, pts.s, pts.f)

```



Q11 : *Is there a difference between the two final configurations? If so, what does that imply about our procedures?* Jump to A11 •

Task 40 : Visualize the difference between the final schemes as two point-sets on the same graph. •

```

> print(spplot(p, xlab="E", ylab="N", col.regions="white",
+             main="Comparing final sampling schemes",
+             sub="Fixed: blue; Moveable: green (scheme 1), red (scheme 2)",
+             scales=list(draw=T),
+             sp.layout = list(
+                 list("sp.points", s.final, col="red",
+                     pch=20, cex=1.6),
+                 list("sp.points", s.final.mean.1, col="green",
+                     pch=20, cex=1.6),

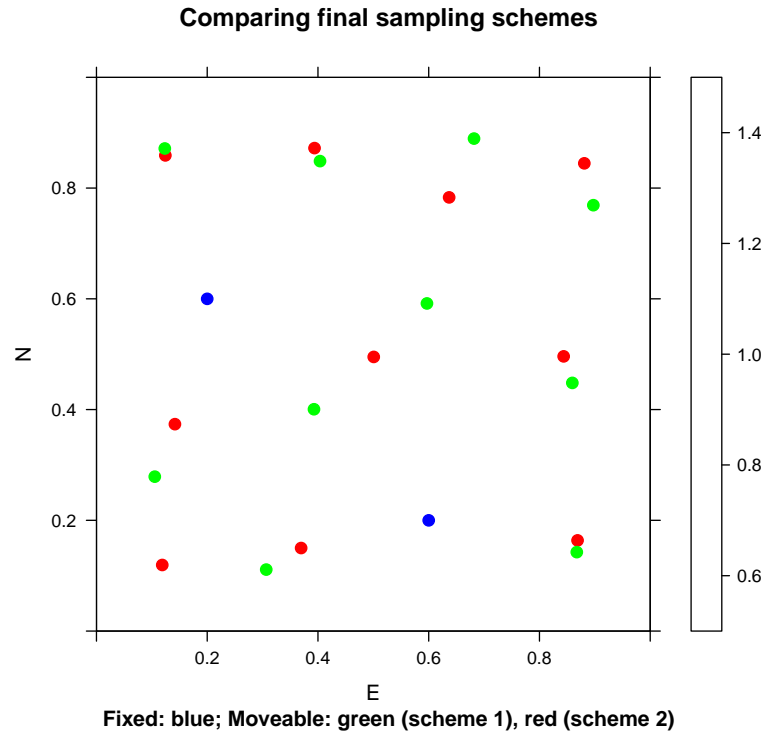
```



```

+         list("sp.points", fix.pts, col="blue",
+             pch=20, cex=1.6))
+     ))

```



Task 41 : Clean up from this section. •

```

> rm(max.dist, n.pts, p, prec, s.final, scheme, vm, ssa.ok,
+     s.final.mean.1, time)

```

3.6 Answers

A9 : The new points are in a regular hexagonal grid, displaced to the right and upwards; the points at the top are right on the boundary. The new point near (0.18, 0.68) is quite close to the fixed point at (0.2, 0.6) and should move upwards and to the left. If the optimization criterion is for mean kriging variance, the points at the top should move down, away from the top boundary. *Return to Q9* •

A10 : In the MAX_OK scheme eight of the points are near the edges; this ensures that no location is poorly-mapped. In the MEAN_OK scheme the points are more evenly spread out over map. Note the difference in maximum kriging prediction variance: it is much lower for the MAX_OK scheme than for the MEAN_OK scheme; the reverse is the case for the previous scheme, where the mean variance is much lower. *Return to Q10* •

A11 : Yes, they are different. Ideally, the optimum should be the same. Some parameters of the SSA should be adjusted. [Return to Q11](#) •

4 Sampling an irregularly-shaped area

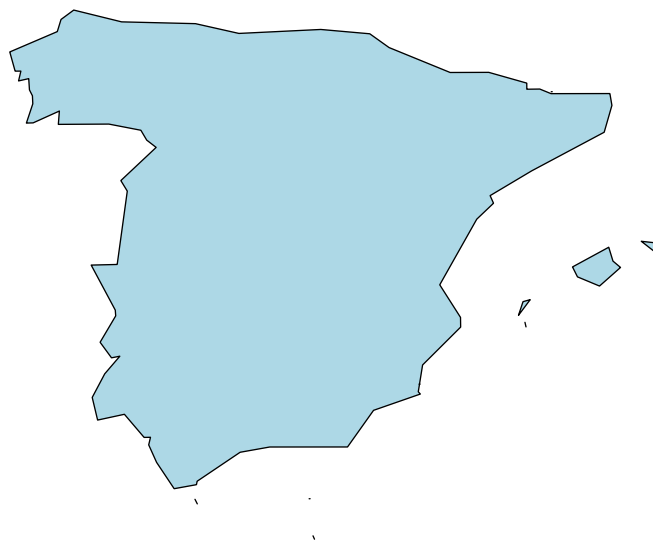
This section shows another practical application of simulated annealing: when a study area is irregularly-shaped, a regular hexagonal grid will not be optimal. Annealing will adjust the scheme to account for the irregularities.

Task 42 : Import a polygon map of Spain from the low-resolution world polygon map database of the `mapdata` package. •

The import function is `map`.

Note: There is also a high-resolution map; specify `"worldHires"` as the first argument to `map`; here we use the low-resolution for speed and because the details of the coastline will have almost no effect on the sampling scheme, since we optimize based on a 10 km grid, see below.

```
> require(mapdata)
> tmp <- map("world", "Spain", fill = TRUE, plot = T, col = "lightblue")
```



Task 43 : Convert this map to an `sp` object, i.e., `SpatialPolygons`, at the same time specifying the initial Coördinate Reference System (CRS) •

Conversion is by the `map2SpatialPolygons` function of the `maptools` package.

From the documentation of the world maps, we know the CRS is geographic coördinates on the WGS84 datum; we build a proper `proj4string` argument with the CRS function defined by the `rgdal` package and also as a stub in the `sp` package.

```
> require(maptools)
> Spain.polys <- map2SpatialPolygons(tmp, IDs=tmp$names,
+   proj4string=
+   CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"))
```

Task 44 : Specify a metric system appropriate for Spain; convert the imported map to the metric system. •

We use the ETRS89-LAEA / TM30 (European Terrestrial Reference System – Lambert Azimuthal Equal-Area / UTM zone 30 coördinates) conformal projection², designed for statistical applications in Europe; the coördinates match UTM zone 30N, centred on -3° longitude, although the origin of the projection is at 52° N, 10° E³.

Note: An equal-area projection is used so that areal-based statistics, such as crop areas, are uniform across Europe.

Conversion is done by the `spTransform` method of the `rgdal` “R interface to the Geospatial Data Abstraction Library⁴” package.

² http://georepository.com/crs_3042/ETRS89-TM30.html

³ near the tiny village of Evensen, south of Hannover in Lower Saxony (D)

⁴ <http://www.gdal.org/>

```

> require(rgdal)
> bbox(Spain.polys)

      min      max
x -9.2347  4.242
y 35.2744 43.776

> laea_crs <-
+   "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80 +units=m +no_defs"
> Spain.polys <- spTransform(Spain.polys, CRS(laea_crs))
> (bb <- bbox(Spain.polys))

      min      max
x 2767850 3825942
y 1456999 2468383

> (max.dist <- sqrt(((bb[1,"max"] - bb[1,"min"])^2)
+                  + ((bb[2,"max"] - bb[2,"min"])^2)
+                  ))

[1] 1463713

> print(paste("Diagonal of bounding box:",
+             round(max.dist/1000), "km"))

[1] "Diagonal of bounding box: 1464 km"

```

Task 45 : Load the simulated annealing function saved in §3.1.1. •

We choose to load the compiled form, with load:

```
> load("ssa_ok.RData")
```

Task 46 : Place 80 points randomly across the study area. •

```

> prec <- 4
> n.pts <- 80
> set.seed(316)
> repeat {
+   sample.try <- spsample(Spain.polys, n = n.pts, type = "random")
+   if (length(coordinates(sample.try)[, 1]) == n.pts)
+     break
+ }
> scheme <- SpatialPointsDataFrame(sample.try, data.frame(z = rep(0,
+   n.pts)))
> scheme@coords <- round(scheme@coords, prec)
> rm(sample.try)

```

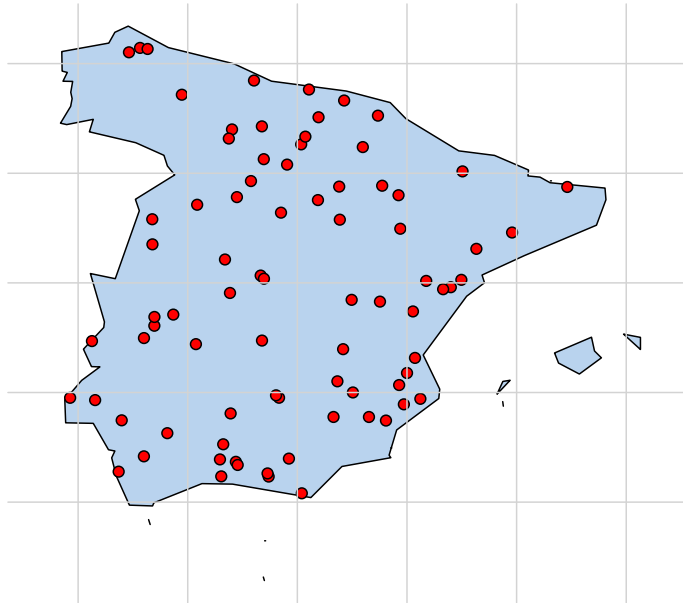
Task 47 : Plot the starting scheme. •

```

> plot(Spain.polys, col = "slategray2")
> points(coordinates(scheme), col = "black", bg = "red",

```

```
+      pch = 21)
> grid(lty = 1)
```



Q12 : *Describe the random starting scheme. Does it cover Spain evenly?*
Jump to A12 •

We suppose a previous study has established that the target variable has an exponential spatial auto-correlation, with a range of 120 km; an example might be some weather variable such as 10-day precipitation total.

Task 48 : Specify the variogram model as required by `gstat`. •

```
> vm <- vgm(psill = 1, model = "Exp", range = 120000/3,
+      0)
```

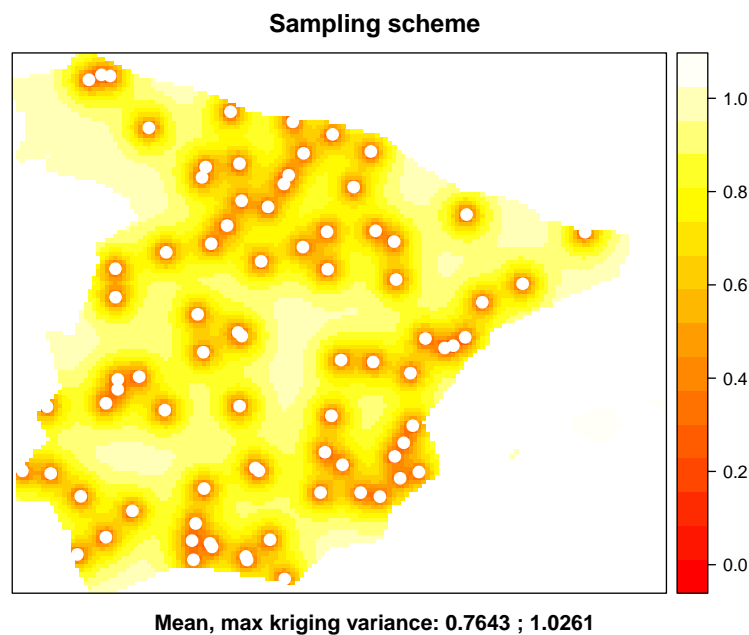
Task 49 : Plot the kriging prediction variance of the starting scheme, on a 10 km interpolation grid. •

```
> res <- round(max.dist/sqrt(2)/10000)
> grid <- spsample(Spain.polys, n=res^2, type="regular")
> k <- krige(z ~ 1, loc=scheme,
+      model=vm, newdata=grid, debug.level=0)
> gridded(k) = T
```

```

> pts.s <- list("sp.points", scheme, col="white",
+             pch=20, cex=1.6)
> print(spplot(k, zcol="var1.var",
+             col.regions=heat.colors(64),
+             main=paste("Sampling scheme"),
+             sub=paste("Mean, max kriging variance:",
+             round(mean(k$var1.var),4),";",
+             round(max(k$var1.var),4)),
+             sp.layout = list(pts.s)
+             ))
> rm(grid, k, pts.s)

```



Q13 : Does this scheme appear to minimize the average or maximum kriging variance? *Jump to A13* •

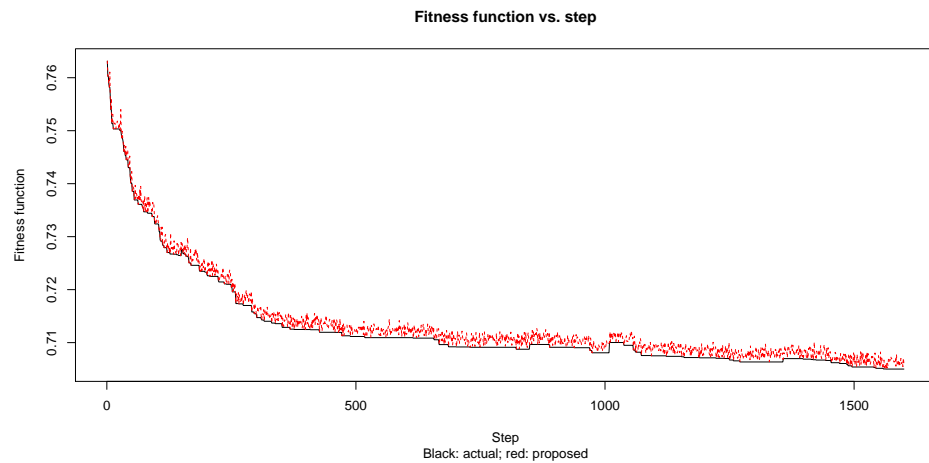
Task 50 : Call the simulated annealing function, specifying a 10 km interpolation grid for kriging. •

Here we limit the total number of steps to complete in a reasonable time; you could raise or remove the limit.

```

> p <- Spain.polys
> (res <- round(max.dist/sqrt(2)/10000))
> time <- system.time(tmp <- ssa.ok(s.init = scheme, s.area = p,
+   s.res = res, s.model = vm, s.f = mean, s.prob = 0.9,
+   s.alpha = 0.999, s.decay = 1024, eq.steps = 200,
+   max.steps = 1600))

```



```
> print(time)

      user  system elapsed
394.505    4.414  401.006

> print(paste("Precision of fitness matching:", tmp[["s.prec"]],
+           "decimal places"))

[1] "Precision of fitness matching: 5 decimal places"

> print(paste("Initial Metropolis temperature:", round(tmp[["metr.temp"]],
+           4)))

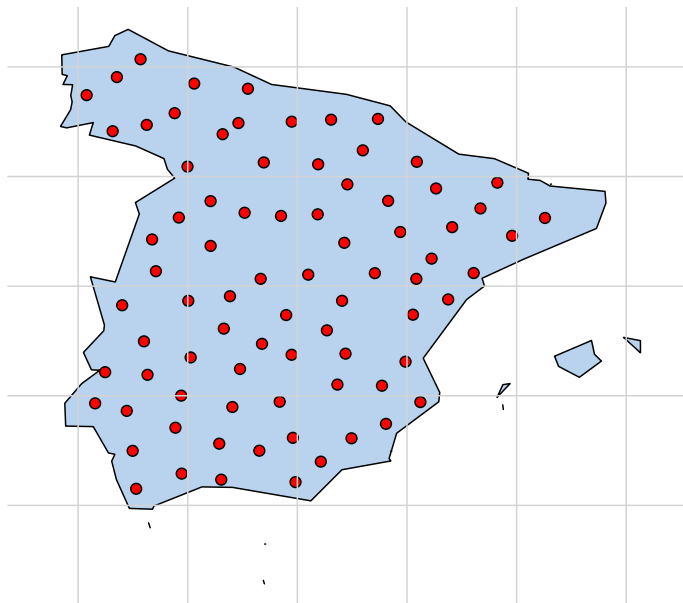
[1] "Initial Metropolis temperature: 0.0129"

> s.final <- tmp[["scheme"]]
```

On the test system, this took about 7 minutes.

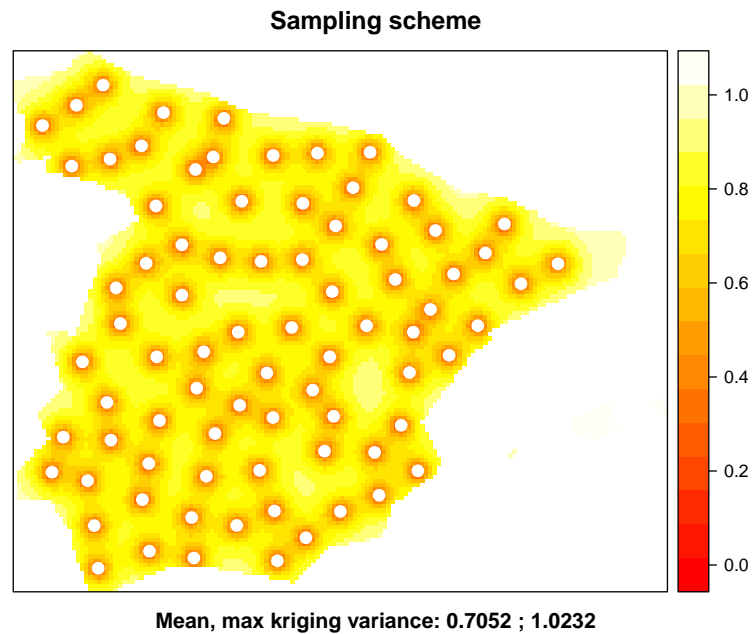
Task 51 : Plot the ending scheme. •

```
> plot(Spain.polys, col = "slategray2")
> points(coordinates(s.final), col = "black", bg = "red",
+       pch = 21)
> grid(lty = 1)
```



Task 52 : Plot the kriging prediction variance of the final scheme. •

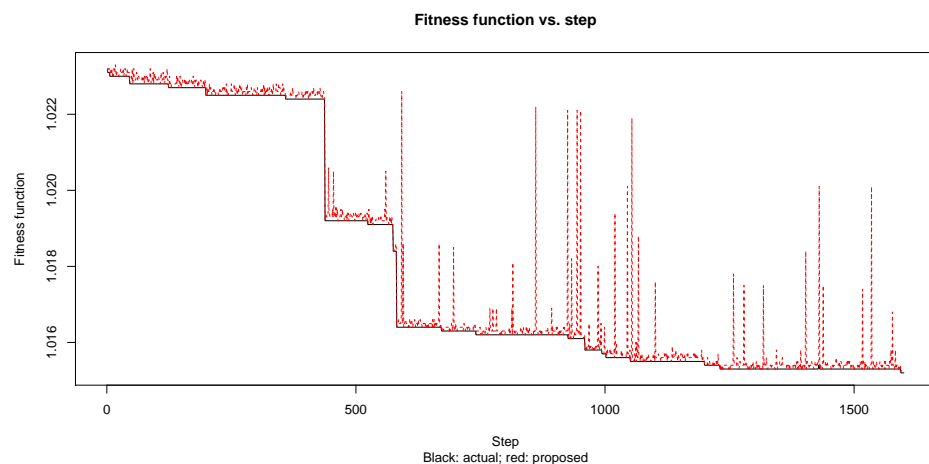
```
> scheme <- s.final
> res <- round(max.dist/sqrt(2)/10000)
> grid <- spsample(Spain.polys, n=res^2, type="regular")
> k <- krige(z ~ 1, loc=scheme,
+           model=vm, newdata=grid, debug.level=0)
> gridded(k) = T
> pts.s <- list("sp.points", scheme, col="white",
+              pch=20, cex=1.6)
> print(spplot(k, zcol="var1.var",
+              col.regions=heat.colors(64),
+              main=paste("Sampling scheme"),
+              sub=paste("Mean, max kriging variance:",
+              round(mean(k$var1.var),4),";",
+              round(max(k$var1.var),4)),
+              sp.layout = list(pts.s)
+              ))
> rm(grid, k, pts.s)
```

Q14 : *How does this scheme compare to the original random scheme? Can it be improved?* *Jump to A14* •

Task 53 : Repeat the optimization, but minimizing the maximum kriging variance, as the optimization criterion. •

```
> time <- system.time(tmp <- ssa.ok(s.init = scheme, s.area = p,
+   s.res = res, s.model = vm, s.f = max, s.prob = 0.9,
+   s.alpha = 0.999, s.decay = 1024, eq.steps = 200,
+   max.steps = 1600))
```



```

> print(time)

      user  system elapsed
375.624    4.061  380.881

> print(paste("Precision of fitness matching:", tmp[["s.prec"]],
+           "decimal places"))

[1] "Precision of fitness matching: 4 decimal places"

> print(paste("Initial Metropolis temperature:", round(tmp[["metr.temp"]],
+           4)))

[1] "Initial Metropolis temperature: 0.001"

> s.final <- tmp[["scheme"]]

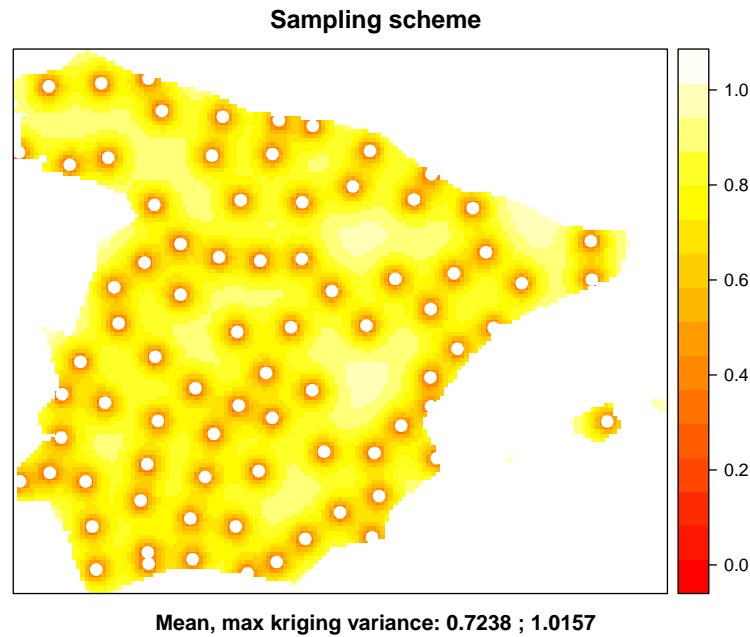
```

Task 54 : Plot the kriging prediction variance of the final scheme. •

```

> scheme <- s.final
> res <- round(max.dist/sqrt(2)/10000)
> grid <- spsample(Spain.polys, n=res^2, type="regular")
> k <- krige(z ~ 1, loc=scheme,
+           model=vm, newdata=grid, debug.level=0)
> gridded(k) = T
> pts.s <- list("sp.points", scheme, col="white",
+             pch=20, cex=1.6)
> print(spplot(k, zcol="var1.var",
+             col.regions=heat.colors(64),
+             main=paste("Sampling scheme"),
+             sub=paste("Mean, max kriging variance:",
+             round(mean(k$var1.var),4),";",
+             round(max(k$var1.var),4)),
+             sp.layout = list(pts.s)
+             ))
> rm(grid, k, pts.s)

```



Q15 : How does this scheme compare to the scheme where the mean kriging variance is minimized? [Jump to A15](#) •

4.1 Answers

A12 : The scheme is quite uneven and does not cover the area evenly. For example S Galicia has almost no points whereas three points are clustered together in N Galicia near Vilalba. Note no points were assigned to the Balearic isles. [Return to Q12](#) •

A13 : The scheme has “holes” with high kriging variance, far from sample points; these have some inefficient clusters. Definitely the maximum kriging variance is not minimized, especially at the borders in Galicia. The mean kriging variance is likely to be sub-optimal. [Return to Q13](#) •

A14 : This scheme is a big improvement – points have moved out of clusters and are spread more evenly. [Return to Q14](#) •

A15 : This scheme moves points closer to the borders. One large jump has placed a point in Mallorca, which had the largest kriging variance in the original scheme, since it had no points and was very far from any point in peninsular Spain. [Return to Q15](#) •

5 SSA optimization for Kriging with External Drift

In the previous sections we implemented SSA to optimize sampling design for interpolation by Ordinary Kriging (OK), i.e. where only the target variable is used for interpolation. Brus and Heuvelink [3] extended SSA for the situation when there are secondary predictors, known over the study area, that can be used to help predict a primary variable. A typical example is the concentration of some constituent of the soil (e.g. organic matter or heavy metal) with the help of some environmental co-variables such as terrain parameters, a vegetation index, or land use. This optimizes sampling design for Kriging with External Drift (KED). We have not developed that here; however adapting our code to this situation would be fairly easy:

1. The variogram model and kriging formulas would have to be of the **residuals** from the linear model developed to describe the external drift; for OK the formula only depends on the “intercept”, i.e., the target variable described by its spatial autocorrelation: $z \sim 1$. In KED this has the form $z \sim \dots$ where \dots can be any linear model. For example, if the external drift is an additive model on covariables elevation, named `elev`, and a vegetation index, named `ndvi`, the formula would be $z \sim \text{elev} + \text{ndvi}$.
2. The variogram model must be fit against the residuals from the linear model specified by the KED formula, using the known (fixed) points, e.g.:

```
v <- variogram(z ~ elev + ndvi, data=fix.pts)
vm <- fit.variogram(v, model=vgm(...))
```

where \dots here represents an initial variogram model.

If there are no fixed points, the model must be estimated from other studies, but in any case using the residuals.

3. The data frames of sample points and prediction grid (to evaluate fitness) must contain fields with values for all the variables in the KED formula, as well as a field for the target variable. These fields must be populated; that is, they must have the actual values of the covariable. This is because the KED variance depends on the prediction variance of the linear model, as well as the spatial configuration of the residuals, and the linear model, implicit in the KED kriging system of equations, is fit with the values of the covariable against the target variable.

6 SSA optimizing by variogram matching

The optimization of the previous sections was for mapping given a known variogram. Another reason to sample is to determine the spatial dependence structure, e.g., by estimating the variogram model. Here we develop an optimization that can be applied to design a sampling scheme to estimate the actual variogram, assuming that it is not too different from the variogram model against which we are optimizing. That is:

1. assume a variogram model from previous studies and expert knowledge;
2. place points so as to estimate this variogram from the sample;
3. sample and estimate the actual variogram; this will not be identical to the assumed variogram but if it is not too different the sampling scheme should be a good design to determine a better model.

There are two criteria that can be applied:

1. Match the empirical variogram of the proposed sample to the assumed variogram;
2. Match a desired distribution of point-pairs in the empirical variogram.

The second criterion is important because a sampling plan to estimate a variogram must place point-pairs at different separations, especially with enough pairs at close range to reliably estimate the short-range structure.

The first criterion is the match between known and empirical variogram. There are two ways to match:

1. fit a variogram model to the empirical variogram and check the discrepancy in parameters;
2. convert the variogram model to variogram bins matching the empirical model and check the (possibly weighted) discrepancy between the bins.

We choose the latter approach, because automated variogram fitting may be unreliable. The first criterion is thus:

$$SS_{\text{vgm}} = \sum_{i=1}^c w_i (\gamma_i^* - \gamma_i)^2 \quad (6)$$

where c is the number of variogram bins, w_i are the weights assigned to each bin, γ_i^* are the semivariance of each bin from the assumed model, and γ_i are the semivariance of each bin from the actual point distribution.

For the second criterion we use the idea of Warrick and Myers [12], who optimized according to the number of point-pairs at each separation, according to a pre-specified distribution:

$$SS_{\text{np}} = \sum_{i=1}^c (f_i^* - f_i)^2 \quad (7)$$

where f_i^* is the desired number of point-pairs, and f_i is the actual number of point-pairs, in each bin.

Task 55 : Modify function `ssa_ok` to optimize on the basis of Equations 6 and 7. •

The main optimization function must be adapted to change the objective function. New arguments are:

- `s.vm` : a variogram model to match, of class `variogramModel`, generally as created with the `vgm` function;
- `s.vm.nc` : number of bins with which to discretize the variogram; default is 15;
- `s.sep.pow` : power with which to weight SS_{vgm} , default inverse distance, i.e., 1; e.g., specify 2 for inverse-distance squared or 0 for no weighting; default is 2;
- `s.sep.np` : a vector with length equal to or less than the number of variogram bins (discretization, in argument `s.vm.nc`) with the desired number of point-pairs for SS_{np} , starting from the closest separation; if not defined this part of the optimization is not used; this will be normalized to sum to 1 if necessary;
- `s.ss.wt` : proportion of optimization function allocated to SS_{vgm} ; default is equal weighting with SS_{np} , both normalized by their approximate maxima.

All references to a kriging grid can be removed, because here the optimization is directly from the point set.

```
> ssa.matchv <- function(s.init, s.area,
+                         s.vm, s.vm.nc=15, s.sep.pow=2,
+                         s.ss.wt=0.5, s.sep.np=NA,
+                         s.fix.pts=NA, s.decay=256,
+                         s.alpha=0.99, s.prob=0.8,
+                         eq.steps=200,
+                         max.steps=NA, s.prec=NA) {
+
+   # fitness function
+   #   argument: current point configuration
+   #   implicit: discretized model 'vm.l', cutoff 'cut', bin width 'bin.w'
+   #   cutoff is fixed because automatic cutoff will vary as points move
+   #   opt.wt is the proportional weight for ss.vgm
+   #
+   # weights vgm difference by number of point-pairs
+   #   and inverse distance squared
+   v.diff <- function(pts) {
+     v <- variogram(z ~ 1, loc=pts, cutoff=cut, width=bin.w)
+     # compare the semivariances
+     ss.vgm <- sum((((v$gamma - vm.l$gamma)^2 * v$np) /
+                     v$dist^s.sep.pow))/sum(v$np)
+     # compare the point-pair distribution, only for requested bins
+     ss.np <- ifelse(is.logical(s.sep.np), 0,
+                     sum(((s.sep.np - v$np)[1:length(s.sep.np)])^2))
+     value <- opt.wt*ss.vgm + (1-opt.wt)*ss.np
+     return(value)
+   }
+
+   # helper function: compute initial temperature
+   metr.temp.init <- function(n.steps) {
+     s <- s.init # initial 'fitness' in environ
+     sum.fit <- 0; i <- 0
+     while (i < n.steps) {
+       pt <- sample(1:n.pts, 1)
```

```

+       pt.old <- coordinates(scheme)[pt,]
+       s.try <- NA
+       while (is.logical(s.try)) {
+         xnew <- pt.old[1] + runif(1, min=-s.max.shift/2,
+                                   max=s.max.shift/2)
+         ynew <- pt.old[2] + runif(1, min=-s.max.shift/2,
+                                   max=s.max.shift/2)
+         pt.new <- data.frame(x = xnew, y = ynew)
+         coordinates(pt.new) <- c(1,2)
+         # over() requires identical CRS, note order
+         proj4string(pt.new) <- proj4string(s.area)
+         if (!is.na(over(pt.new, s.area))) s.try <- s
+       }
+       s.try@coords[pt,] <- c(xnew, ynew)
+       # fitness function in environ
+       if (is.logical(s.fix.pts)) pts <- s.try
+       else pts <- rbind(s.fix.pts, s.try)
+       # check new fitness, only use if worse
+       fit.new <- v.diff(pts)
+       if (fit.new > fitness) {
+         sum.fit <- sum.fit + fit.new
+         i <- i+1 }
+     } # note change of sign to ensure negative sum
+     return((fitness-(sum.fit/n.steps))/log(s.prob))
+   }
+
+   ## main body
+   ## 0 - check and adjust arguments if necessary
+
+   ## 1 - setup
+   # 1.1 - compute maximum distance across bbox
+   max.dist <- sqrt(diff(bbox(s.area)[1,])^2
+                     + diff(bbox(s.area)[2,])^2)
+   n.pts <- length(s.init$z)
+
+   # 1.2 - initialize variogram matching, fixed cutoff
+   cut <- max.dist/3
+   bin.w <- cut/s.vm.nc
+   v <- variogram(z ~ 1, loc=scheme, cutoff=cut, width=bin.w)
+   vm.l <- variogramLine(object=s.vm, dist_vector=v$dist)
+
+   # 1.2a - normalize the two parts of the objective function
+   # use the initial values as rough guides to the relative magnitude
+   m.1 <- s.ss.wt*sum((((v$gamma - vm.l$gamma)^2 * v$np) /
+                       v$dist^s.sep.pow))/sum(v$np)
+   m.2 <- (1-s.ss.wt)*
+   ifelse(is.logical(s.sep.np), 0,
+          sum((s.sep.np - v$np)[1:length(s.sep.np)]^2))
+   # opt.wt is the proportional weight for ss.vgm
+   opt.wt <- ifelse(m.2==0, 1, m.2/(m.1+m.2))
+   rm(m.1, m.2)
+
+   # 1.3 - initialize point set
+   if (is.logical(s.fix.pts)) pts <- s.init
+   else pts <- rbind(s.fix.pts, s.init)

```

```

+ fitness <- v.diff(pts) # initial fitness
+
+ # 1.4 - if precision was not specified, set as 3rd significant
+ # figure of initial fitness
+ i <- 0
+ while (is.na(s.prec)) {
+   if (fitness%%(10^-i) != 0) s.prec=i+4 else i <- i + 1
+ }
+
+ # 1.5 - maximum shift depends on point density
+ # this is an empirical value
+ s.max.shift <- 2*max.dist/sqrt(n.pts)
+
+ # 1.6 - initialize temperature
+ metr.temp <-
+   ret.metr.temp.init <-
+   metr.temp.init(128); # this is enough for a rough guess
+
+ # 1.7 - set up to record evolution of fits
+ fits <- matrix(nrow = 0, ncol=4)
+ colnames(fits) <- c("actual","proposed","max.shift","temperature")
+ this.step <- 0; equal.steps <- 0;
+ s <- s.init;
+
+ ## 2 - main loop
+ # 2.1 get a proposed new point in the study area
+ repeat {
+   max.shift <- s.max.shift * exp(-this.step/s.decay)
+   metr.temp <- metr.temp * s.alpha
+
+   pt <- sample(1:n.pts, 1)
+   pt.old <- coordinates(s)[pt,]
+   repeat {
+     xnew <- round(pt.old[1] + runif(1, min=-max.shift,
+                                   max=max.shift),prec)
+     ynew <- round(pt.old[2] + runif(1, min=-max.shift,
+                                   max=max.shift),prec)
+
+     pt.new <- data.frame(x = xnew, y = ynew)
+     coordinates(pt.new) <- ~ x + y
+     proj4string(pt.new) <- proj4string(p)
+     if (!is.na(over(pt.new, p))) break
+   } # end repeat get a point
+
+   # 2.2 increment step when we have a proposed point
+   this.step <- this.step + 1;
+   s.try <- s;
+   # update the selected point's coordinates
+   s.try@coords[pt,] <- c(xnew, ynew);
+   # 2.3 new fitness, to the desired precision
+   if (is.logical(s.fix.pts)) pts <- s.try
+   else pts <- rbind(s.fix.pts, s.try)
+   fitness.new <- round(v.diff(pts), s.prec);
+
+   # 2.4 acceptance criteria; maybe update scheme

```



```

+         if (fitness.new < fitness) {
+             s <- s.try;
+             fitness <- fitness.new;
+             equal.steps <- 0
+         } else if (runif(1) < (metr.temp
+                               * exp(fitness - fitness.new))) {
+             s <- s.try;
+             fitness <- fitness.new;
+             equal.steps <- 0
+         } else { equal.steps <- (equal.steps + 1) }
+
+     # 2.5 save record of fits
+     fits <- rbind(fits, c(fitness
+                           , fitness.new,
+                           max.shift, metr.temp))
+
+     # 2.6 check stopping criteria
+     if (equal.steps >= eq.steps) break;
+     if (!is.na(max.steps) & (this.step >= max.steps)) break;
+ } # end repeat main loop
+
+ # 3 - plot the fitness vs. step
+ plot(1:length(fits[,1]), fits[, "actual"],
+      xlab="Step", ylab="Fitness function", type="l",
+      main="Fitness function vs. step",
+      sub="Black: actual; red: proposed",
+      ylim=c(min(fits[,1]), max(fits[,2])))
+ lines(1:length(fits[,1]), fits[, "proposed"],
+       type="l", col="red", lty=2)
+
+ # 4 - return the final sampling scheme
+ #       and the computed annealing parameters
+ return(list(scheme=s,
+            metr.temp=ret.metr.temp.init,
+            s.prec=s.prec))
+ }

```

We begin with a unit square area,

Task 56 : Set up the bounding polygon and location precision. •

```

> p <- SpatialPolygons(list(Polygons(list(Polygon(matrix(c(0,
+     0, 0, 1, 1, 1, 1, 0, 0, 0), byrow = T, nrow = 5,
+     ncol = 2))), ID = 1)))
> p <- SpatialPolygonsDataFrame(p, data = data.frame(id = 1))
> max.dist <- sqrt(2)
> prec <- 3

```

6.1 Indicator variogram, no bin numbers

We begin with points with binary values (indicators), i.e., either 0 or 1, in a random pattern. The `rbinom` function simulates binomial trials; with each trial being one “coin flip” the outcome is either 0 or 1.

```

> n.pts <- 80
> prob = 0.5
> set.seed(25)
> scheme <- SpatialPointsDataFrame(spsample(p, n = n.pts,
+   type = "random"), data.frame(z = rep(rbinom(n = n.pts,
+   size = 1, prob = prob))))
> scheme@coords <- round(scheme@coords, prec)

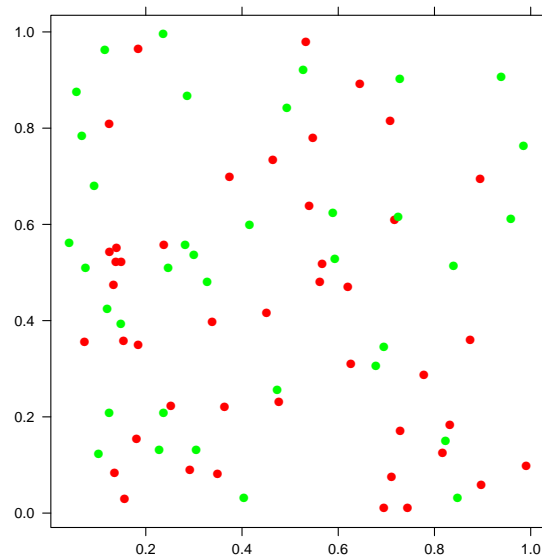
```

Plot the initial scheme:

```

> spplot(scheme, col.regions = c("red", "green"), key.space = "",
+   scales = list(draw = T))

```



Q16 : *Is there any pattern to the spatial distribution of the red (0) and green (1) points?* *Jump to A16* •

Task 57 : Set up a variogram to match. •

For an indicator variogram the structural sill is the expected variance of the random sample of the given size. We choose a zero nugget (to maximize the effect of point distribution) and a range that will cause points with the same value to cluster:

```

> (sill <- (prob) * (1 - prob))

[1] 0.25

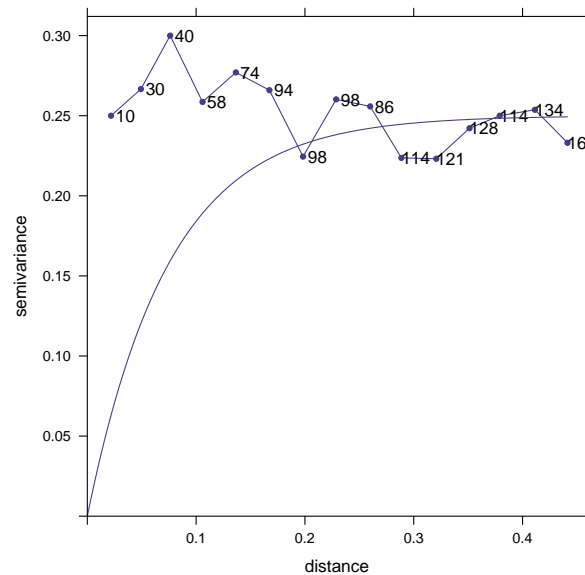
> vm <- vgm(psill = sill, model = "Exp", range = 0.075,
+   0)

```

Task 58 : Plot the empirical variogram from the initial scheme vs. the

model, also with the number of point-pairs in each bin. •

```
> v <- variogram(z ~ 1, loc = scheme)
> plot(v, model = vm, plot.numbers = T, pch = 20, type = "b")
```

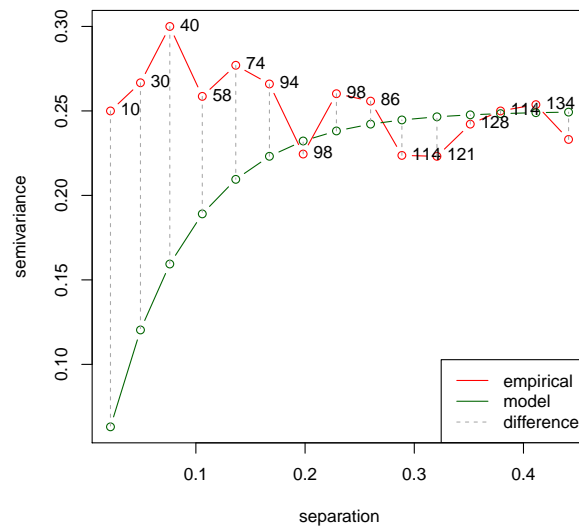


Q17 : Does the empirical variogram match the model? *Jump to A17* •

Task 59 : Convert the variogram model to a set of bins and plot the two variograms together, also with the number of point-pairs in each bin. •

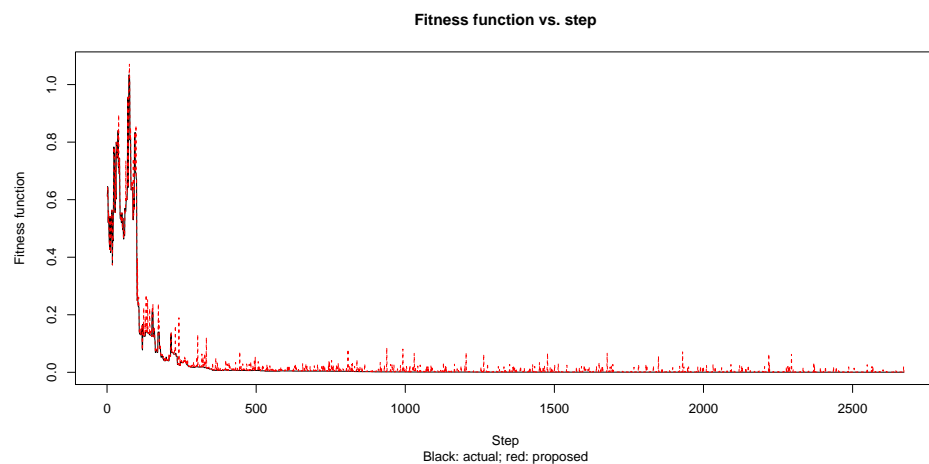
This can be done with the `variogramLine` function:

```
> vl <- variogramLine(object = vm, dist_vector = v$dist)
> plot(v$gamma ~ v$dist, type = "b", xlab = "separation",
+      ylab = "semivariance", col = "red", ylim = c(min(v$gamma,
+      vl$gamma), max(v$gamma, vl$gamma)))
> lines(vl$gamma ~ v$dist, type = "b", col = "darkgreen")
> for (i in 1:length(v$dist)) {
+   lines(x = c(v$dist[i], v$dist[i]), y = c(v$gamma[i],
+   vl$gamma[i]), lty = 2, col = "darkgray")
+   text(v$dist[i], v$gamma[i], v$np[i], pos = 4)
+ }
> legend("bottomright", c("empirical", "model", "difference"),
+       lty = c(1, 1, 2), col = c("red", "darkgreen", "darkgray"))
```



Task 60 : Optimize the sampling scheme with simulated annealing, accepting the defaults. •

```
> time <- system.time(tmp <- ssa.matchv(s.init = scheme,
+   s.area = p, s.vm = vm, s.prob = 0.95, s.alpha = 0.99,
+   s.decay = 1024, eq.steps = 200))
```



```
> print(time)

   user  system elapsed
32.363   0.086  32.494

> print(paste("Precision of fitness matching:", tmp[["s.prec"]],
+   "decimal places"))

[1] "Precision of fitness matching: 5 decimal places"
```

```

> print(paste("Initial Metropolis temperature:", round(tmp[["metr.temp"]],
+ 4)))

[1] "Initial Metropolis temperature: 1.3743"

> s.final <- tmp[["scheme"]]

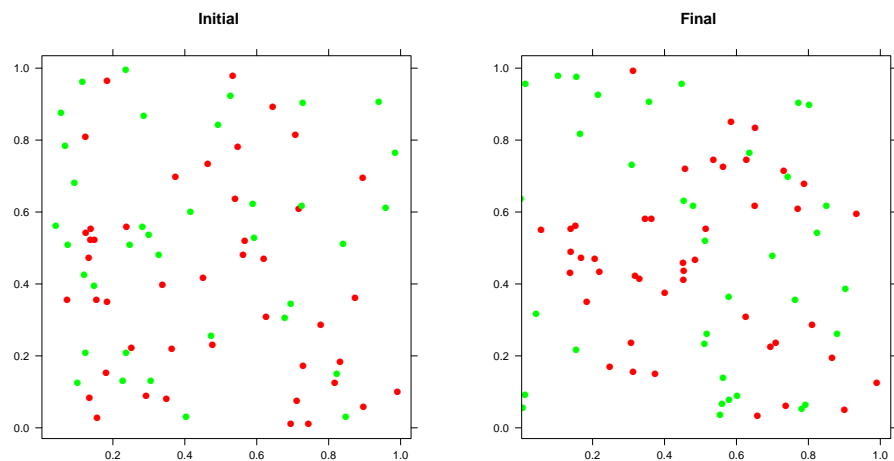
```

Task 61 : Show the final scheme (point pattern), compared to the initial (random scheme). •

```

> p1 <- spplot(scheme, col.regions = c("red", "green"),
+   key.space = "", scales = list(draw = T), main = "Initial")
> p2 <- spplot(s.final, col.regions = c("red", "green"),
+   key.space = "", scales = list(draw = T), main = "Final")
> print(p1, split = c(1, 1, 2, 1), more = T)
> print(p2, split = c(2, 1, 2, 1), more = F)

```



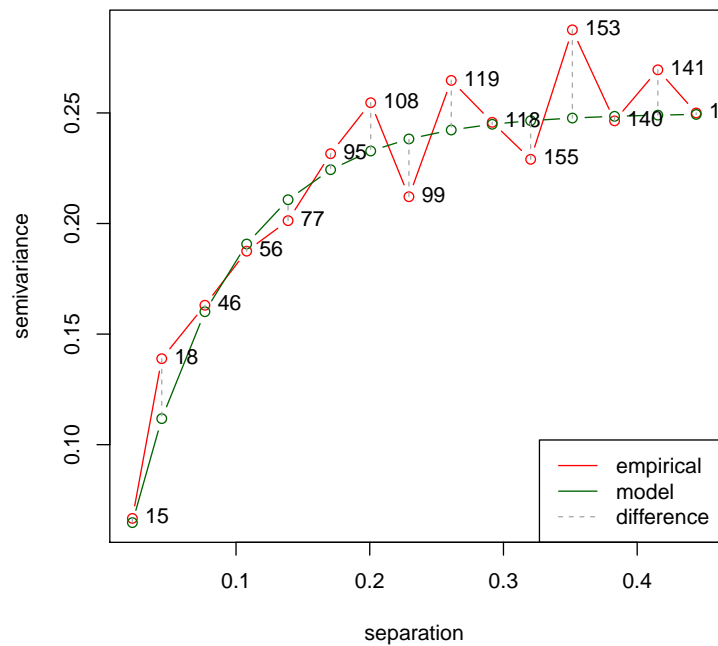
Q18 : What has changed in the spatial arrangement of the red (0) and green (1) points, after the optimization? *Jump to A18* •

Task 62 : Show the final variogram, compared to the model. •

```

> v <- variogram(z ~ 1, s.final)
> vl <- variogramLine(object = vm, dist_vector = v[, "dist"])
> plot(v$gamma ~ v$dist, type = "b", xlab = "separation",
+   ylab = "semivariance", col = "red", ylim = c(min(v$gamma,
+   vl$gamma), max(v$gamma, vl$gamma)))
> lines(vl$gamma ~ v$dist, type = "b", col = "darkgreen")
> for (i in 1:length(v$dist)) {
+   lines(x = c(v$dist[i], v$dist[i]), y = c(v$gamma[i],
+   vl$gamma[i]), lty = 2, col = "darkgray")
+   text(v$dist[i], v$gamma[i], v$np[i], pos = 4)
+ }
> legend("bottomright", c("empirical", "model", "difference"),
+   lty = c(1, 1, 2), col = c("red", "darkgreen", "darkgray"))

```



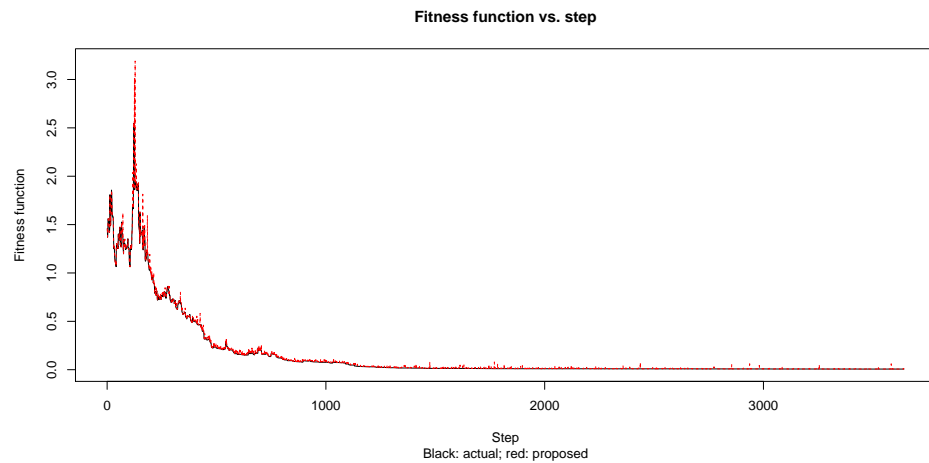
Q19 : *How well do the empirical and model variograms match, after the optimization?* Jump to A19 •

6.2 Indicator variogram, bin numbers

Task 63 : Optimize the same starting scheme, against the same variogram model, but also specifying a point-pair distribution as part of the optimization. •

Some distributions of bin numbers will not be possible. We want to ensure there are enough point-pairs in the close-range bins to accurately estimate the variogram, so we specify the first four bin's preferred number of point-pairs:

```
> vl.np <- c(60, 80, 120)
> time <- system.time(tmp <- ssa.matchv(s.init = scheme,
+   s.area = p, s.vm = vm, s.sep.np = vl.np, s.ss.wt = 0.8,
+   s.prob = 0.95, s.alpha = 0.995, s.decay = 1024, eq.steps = 200))
```



```
> print(time)

      user  system elapsed
41.468    0.108   41.618

> print(paste("Precision of fitness matching:", tmp[["s.prec"]],
+            "decimal places"))

[1] "Precision of fitness matching: 4 decimal places"

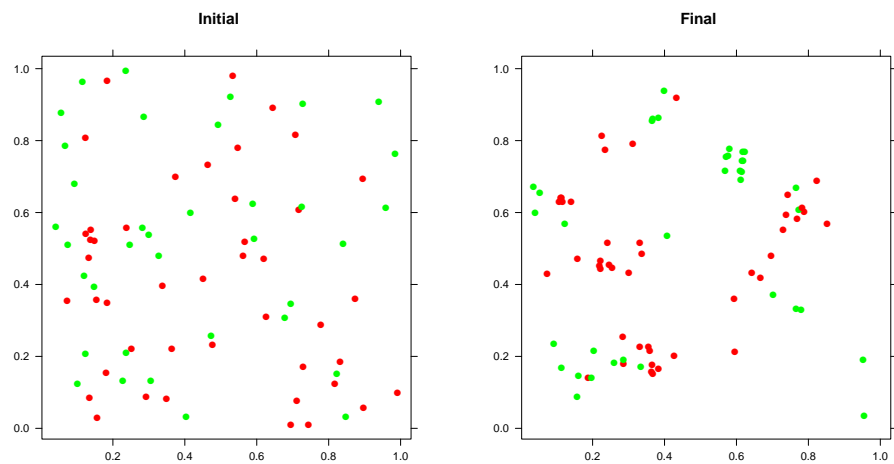
> print(paste("Initial Metropolis temperature:", round(tmp[["metr.temp"]],
+            4)))

[1] "Initial Metropolis temperature: 1.507"

> s.final <- tmp[["scheme"]]
```

Task 64 : Display the optimized point pattern, along with the original random pattern. •

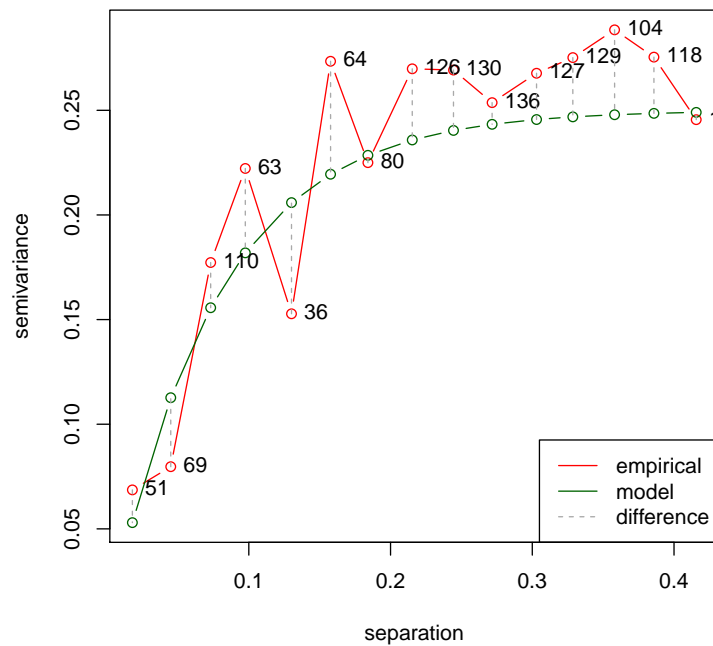
```
> p1 <- spplot(scheme, col.regions = c("red", "green"),
+             key.space = "", scales = list(draw = T), main = "Initial")
> p2 <- spplot(s.final, col.regions = c("red", "green"),
+             key.space = "", scales = list(draw = T), main = "Final")
> print(p1, split = c(1, 1, 2, 1), more = T)
> print(p2, split = c(2, 1, 2, 1), more = F)
```



Q20 : What has changed in the spatial arrangement of the red (0) and green (1) points, after the optimization? Jump to A20 •

Task 65 : Display the empirical variogram of the optimized scheme, against the empirical variogram of the assumed model. •

```
> v <- variogram(z ~ 1, s.final)
> vl <- variogramLine(object = vm, dist_vector = v[, "dist"])
> plot(v$gamma ~ v$dist, type = "b", xlab = "separation",
+      ylab = "semivariance", col = "red", ylim = c(min(v$gamma,
+      vl$gamma), max(v$gamma, vl$gamma)))
> lines(vl$gamma ~ v$dist, type = "b", col = "darkgreen")
> for (i in 1:length(v$dist)) {
+   lines(x = c(v$dist[i], v$dist[i]), y = c(v$gamma[i],
+   vl$gamma[i]), lty = 2, col = "darkgray")
+   text(v$dist[i], v$gamma[i], v$np[i], pos = 4)
+ }
> legend("bottomright", c("empirical", "model", "difference"),
+       lty = c(1, 1, 2), col = c("red", "darkgreen", "darkgray"))
```

Q21 : How well do the empirical and model variograms match (both the form and the number of point-pairs per bin), after the optimization? [Jump to A21](#) •

Task 66 : Determine how closely the actual distribution of point-pairs over the variogram bins matches the requested distribution. •

```
> tmp <- rbind(t1 <- v$np[1:length(vl.np)], vl.np, t1 -
+   vl.np)
> rownames(tmp) <- c("actual", "expected", "difference")
> colnames(tmp) <- round(vl$dist[1:length(vl.np)], 2)
> print(tmp)

          0.02 0.04 0.07
actual      51   69  110
expected    60   80  120
difference   -9  -11  -10

> rm(tmp, t1)
```

6.3 Sampling optimization in a simulated field

The idea here is to place sampling points to determine the variogram structure, for a continuous-valued variable. The outline of the procedure is:

1. Assume a variogram model from previous study;

2. unconditionally simulate a field that reproduces that variogram model;
3. take a random sample of points from that field; by definition this more or less reproduces the assumed variogram, but is not optimal for sampling;
4. anneal with an objective function to match a required distribution of point-pairs, especially to have enough pairs at close separations.

Note that it is not necessary to match the variogram, since the simulated field is built from the assumed variogram, and so any sampling plan by definition matches it.

Once the scheme is designed, the sample is taken and then the actual variogram can be estimated from it. If this does not vary too much from the assumed variogram, the sampling scheme was satisfactory, if not optimal.

Task 67 : Modify the simulated annealing program to (1) optimize by matching a required number of point-pairs per variogram bin; and (2) use a grid instead of polygon to determine if the proposed sampling points are in the study area. •

Arguments are as above, but removing the variogram model `s.vm`, the relative weighting of model vs. point-pairs `s.ss.wt`, and replacing the polygon `s.area` with a grid `s.grid`.

Recall the initial sampling scheme `s.init` must be passed as a `Spatial-PointsDataFrame` including a (dummy) variable named `z`.

```
> ssa.matchnp <- function(s.init, s.grid,
+                          s.vm.nc=15,
+                          s.sep.np=NA,
+                          s.fix.pts=NA, s.decay=256,
+                          s.alpha=0.99, s.prob=0.8,
+                          eq.steps=200,
+                          max.steps=NA, s.prec=NA) {
+
+   # fitness function
+   #   argument: current point configuration
+   #   implicit: cutoff 'cut', bin width 'bin.w'
+   #   cutoff is fixed because automatic cutoff will vary as points move
+   #   opt.wt is the proportional weight for ss.vgm
+   v.np.diff <- function(pts) {
+     v <- variogram(z ~ 1, loc=pts, cutoff=cut, width=bin.w)
+     # compare the point-pair distribution, only for requested bins
+     return(sum(((s.sep.np - v$np)[1:length(s.sep.np)])^2))
+   }
+
+   # helper function: compute initial temperature
+   metr.temp.init <- function(n.steps) {
+     s <- s.init # initial 'fitness' in environ
+     sum.fit <- 0; i <- 0
+     while (i < n.steps) {
+       pt <- sample(1:n.pts, 1)
```

```

+       pt.old <- coordinates(scheme)[pt,]
+       s.try <- NA
+       while (is.logical(s.try)) {
+         xnew <- pt.old[1] + runif(1, min=-s.max.shift/2,
+                                   max=s.max.shift/2)
+         ynew <- pt.old[2] + runif(1, min=-s.max.shift/2,
+                                   max=s.max.shift/2)
+         pt.new <- data.frame(x = xnew, y = ynew)
+         coordinates(pt.new) <- c(1,2)
+         # over() requires identical CRS, note order
+         proj4string(pt.new) <- proj4string(s.grid)
+         if (!is.na(over(pt.new, s.grid))) s.try <- s
+       }
+       s.try@coords[pt,] <- c(xnew, ynew)
+       # fitness function in environ
+       if (is.logical(s.fix.pts)) pts <- s.try
+       else pts <- rbind(s.fix.pts, s.try)
+       # check new fitness, only use if worse
+       fit.new <- v.np.diff(pts)
+       if (fit.new > fitness) {
+         sum.fit <- sum.fit + fit.new
+         i <- i+1 }
+     } # note change of sign to ensure negative sum
+     return((fitness-(sum.fit/n.steps))/log(s.prob))
+   }
+
+   ### main body
+   ## 1 - setup
+   # 1.1 - compute maximum distance across bbox
+   max.dist <- sqrt(diff(bbox(s.grid)[1,])^2
+                     + diff(bbox(s.grid)[2,])^2)
+   n.pts <- length(s.init$z)
+
+   # 1.2 - initialize point set
+   if (is.logical(s.fix.pts)) pts <- s.init
+   else pts <- rbind(s.fix.pts, s.init)
+
+   # 1.3 - initialize variogram matching, fixed cutoff
+   cut <- max.dist/3
+   bin.w <- cut/s.vm.nc
+   v <- variogram(z ~ 1, loc=pts, cutoff=cut, width=bin.w)
+
+   # initial fitness
+   fitness <- v.np.diff(pts)
+
+   # 1.4 - if precision was not specified, set as 3rd significant
+   #       figure of initial fitness
+   i <- 0
+   while (is.na(s.prec)) {
+     if (fitness%%(10^-i) != 0) s.prec=i+4 else i <- i + 1
+   }
+
+   # 1.5 - maximum shift depends on point density
+   #       this is an empirical value
+   s.max.shift <- 2*max.dist/sqrt(n.pts)

```

```

+
+ # 1.6 - initialize temperature
+ metr.temp <-
+   ret.metr.temp.init <-
+   metr.temp.init(128); # this is enough for a rough guess
+
+ # 1.7 - set up to record evolution of fits
+ fits <- matrix(nrow = 0, ncol=4)
+ colnames(fits) <- c("actual","proposed","max.shift","temperature")
+ this.step <- 0; equal.steps <- 0;
+ s <- s.init;
+
+ ## 2 - main loop
+ repeat {
+   max.shift <- s.max.shift * exp(-this.step/s.decay)
+   metr.temp <- metr.temp * s.alpha
+
+   pt <- sample(1:n.pts, 1)
+   pt.old <- coordinates(s)[pt,]
+   ## 2.1 get a proposed new point in the study area
+   i <- 1
+   repeat {
+     xnew <- round(pt.old[1] + runif(1, min=-max.shift/2,
+                                     max=max.shift/2),prec)
+     ynew <- round(pt.old[2] + runif(1, min=-max.shift/2,
+                                     max=max.shift/2),prec)
+
+     pt.new <- data.frame(x = xnew, y = ynew)
+     coordinates(pt.new) <- ~ x + y
+     proj4string(pt.new) <- proj4string(p)
+     if (!is.na(over(pt.new, s.grid))) break
+     else { i <- i + 1
+           if (i > max.steps) {
+             stop(paste("Can't find a new point in", max.steps,"steps"))
+           }
+         }
+   } # end repeat get a point
+
+   ## 2.2 increment step when we have a proposed point
+   this.step <- this.step + 1;
+   s.try <- s;
+   ## update the selected point's coordinates
+   s.try@coords[pt,] <- c(xnew, ynew);
+   ## 2.3 new fitness, to the desired precision
+   if (is.logical(s.fix.pts)) pts <- s.try
+   else pts <- rbind(s.fix.pts, s.try)
+   fitness.new <- round(v.np.diff(pts), s.prec);
+
+   # 2.4 acceptance criteria; maybe update scheme
+   if (fitness.new < fitness) {
+     s <- s.try;
+     fitness <- fitness.new;
+     equal.steps <- 0
+   } else if (runif(1) < (metr.temp
+                         * exp(fitness - fitness.new))) {
+     s <- s.try;

```

```

+         fitness <- fitness.new;
+         equal.steps <- 0
+       } else { equal.steps <- (equal.steps + 1) }
+
+       # 2.5 save record of fits
+       fits <- rbind(fits, c(fitness
+                             , fitness.new,
+                             max.shift, metr.temp))
+
+       # 2.6 check stopping criteria
+       if (equal.steps >= eq.steps) break;
+       if (!is.na(max.steps) & (this.step >= max.steps)) break;
+     } # end repeat main loop
+
+     # 3 - plot the fitness vs. step
+     plot(1:length(fits[,1]), fits[, "actual"],
+          xlab="Step", ylab="Fitness function", type="l",
+          main="Fitness function vs. step",
+          sub="Black: actual; red: proposed",
+          ylim=c(min(fits[,1]), max(fits[,2])))
+     lines(1:length(fits[,1]), fits[, "proposed"],
+          type="l", col="red", lty=2)
+
+     # 4 - return the final sampling scheme
+     #       and the computed annealing parameters
+     return(list(scheme=s,
+                 metr.temp=ret.metr.temp.init,
+                 s.prec=s.prec))
+   }

```

Task 68 : Determine a variogram model for the logarithm of the zinc content of the Meuse topsoils. •

```

> data(meuse)
> coordinates(meuse) <- ~x + y
> meuse$lzn <- log(meuse$zinc)
> summary(meuse$lzn)

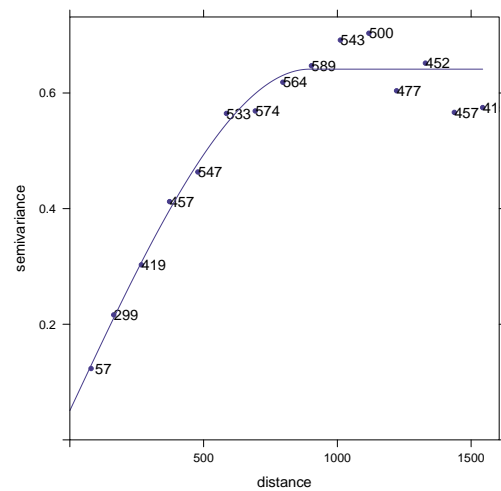
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.73   5.29   5.79   5.89   6.51   7.52

> v <- variogram(lzn ~ 1, loc = meuse)
> (vm <- fit.variogram(v, vgm(0.4, "Sph", 1000, 0.2)))

   model    psill range
1  Nug 0.050659    0
2  Sph 0.590605  897

> plot(v, model = vm, pl = T)

```

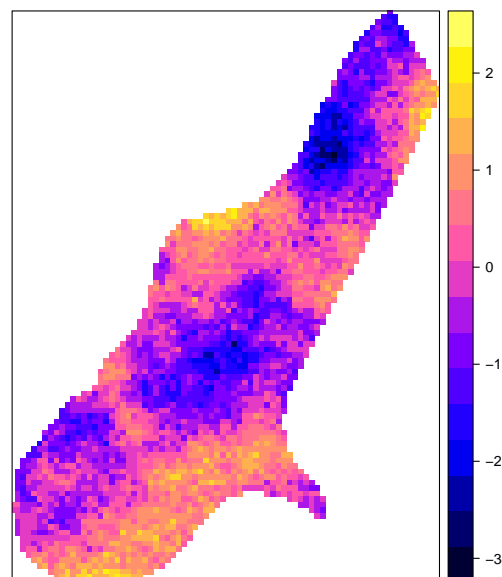


Task 69 : Simulate a field for the logarithm of the zinc content of the Meuse topsoils. •

```
> data(meuse.grid)
> coordinates(meuse.grid) <- ~x + y
> gridded(meuse.grid) <- T
> set.seed(99)
> k.sim <- krige(lzn ~ 1, loc = meuse, newdata = meuse.grid,
+   model = vm, beta = 0, dummy = T, nsim = 1, nmax = 120)

[using unconditional Gaussian simulation]

> spplot(k.sim, col.regions = bpy.colors(64))
```



This is almost surely *not* a true representation of the Zn contents; however it is plausible given the assumed variogram. Now we place samples to determine

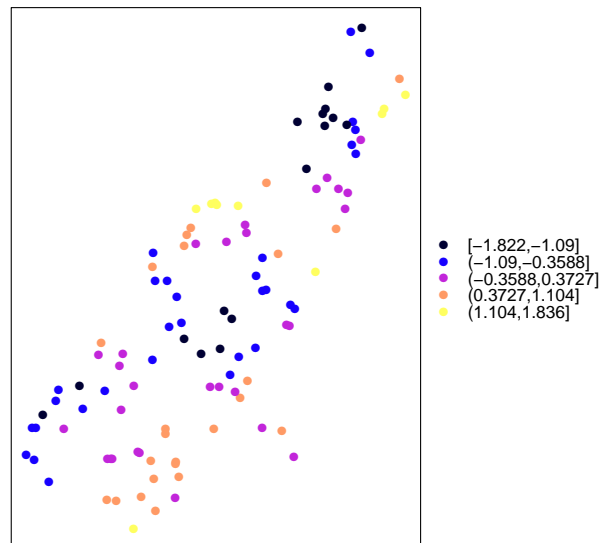
the actual variogram.

Task 70 : Make a starting scheme. •

```
> set.seed(7523)
> scheme <- spsample(meuse.grid, 100, type="random")
> tmp <- over(scheme, k.sim)
> summary(tmp)

      sim1
Min.   :-1.822
1st Qu.: -0.679
Median :-0.225
Mean   :-0.104
3rd Qu.:  0.573
Max.   :  1.836

> # variable must be named 'z'
> scheme.df <- SpatialPointsDataFrame(scheme,
+                                     data=data.frame(z=tmp$sim1))
> spplot(scheme.df, zcol="z", col.regions=bpy.colors(64),
+         key.space="right")
```



Note this by definition reproduces the variogram:

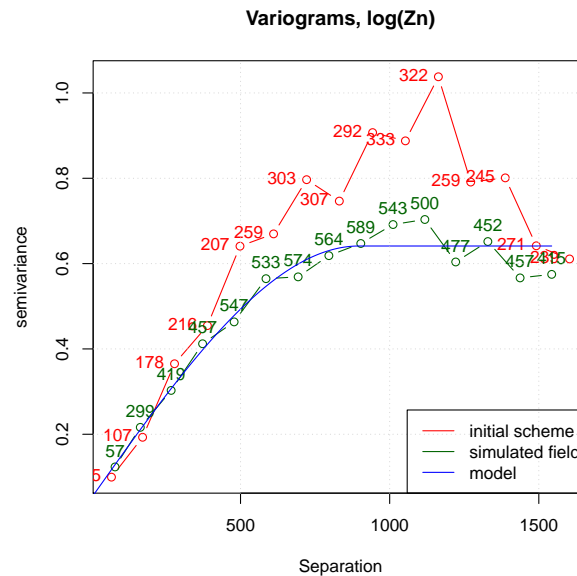
Task 71 : Display the variogram from the initial scheme, from the simulated field, and the fitted variogram model, on the same graph. •

```
> v.scheme <- variogram(z ~ 1, scheme.df)
> plot(v.scheme$gamma ~ v.scheme$dist, type = "b", col = "red",
+      xlab = "Separation", ylab = "semivariance", main = "Variograms, log(Zn)")
> text(v.scheme$dist, v.scheme$gamma, v.scheme$np, pos = 2,
+      col = "red")
```

```

> lines(v$gamma ~ v$dist, type = "b", col = "darkgreen")
> text(v$dist, v$gamma, v$np, pos = 3, col = "darkgreen")
> lines(variogramLine(v, maxdist = max(v$dist)), col = "blue")
> grid()
> legend("bottomright", c("initial scheme", "simulated field",
+   "model"), lty = 1, col = c("red", "darkgreen", "blue"))

```



Q22 : How well does the initial placement reproduce the model? [Jump to A22](#) •

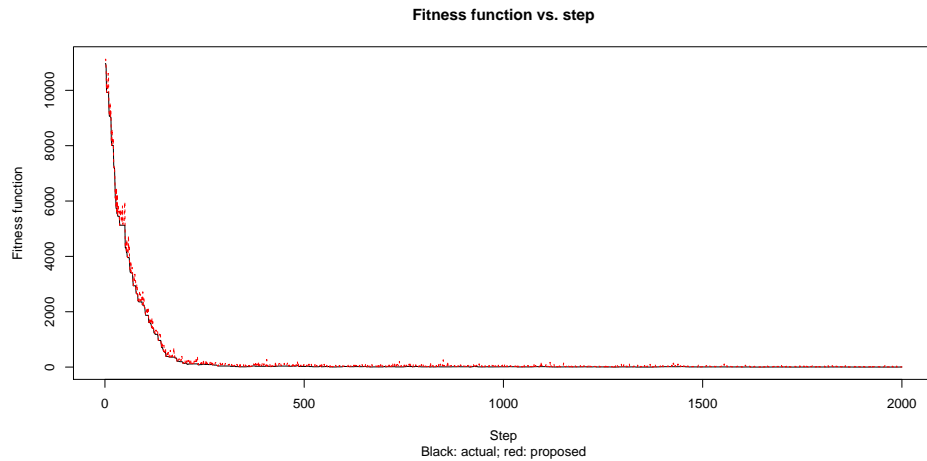
Task 72 : Use simulated annealing to place sampling locations, matching the a specified distribution of point-pairs at closer separations. •

Again we choose to specify a number of point-pairs only for the first three bins.

```

> vl.np <- c(60, 80, 120, 240, 280)
> time <- system.time(tmp <- ssa.matchnp(s.init = scheme.df,
+   s.grid = meuse.grid, s.sep.np = vl.np, s.prob = 0.95,
+   s.alpha = 0.995, s.decay = 1024, eq.steps = 200,
+   max.steps = 2000))

```

```
> print(time)

      user  system elapsed
28.82    0.09   28.94

> print(paste("Precision of fitness matching:", tmp[["s.prec"]],
+            "decimal places"))

[1] "Precision of fitness matching: 4 decimal places"

> print(paste("Initial Metropolis temperature:", round(tmp[["metr.temp"]],
+            4)))

[1] "Initial Metropolis temperature: 8189.271"

> s.final <- tmp[["scheme"]]
```

The points were moved with their original data values; we need to recover the values at the positions they were moved to:

```
> summary(s.final$z)

      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.820  -0.679  -0.225  -0.104   0.573   1.840

> s.final$z <- over(s.final, k.sim)$sim1
> summary(s.final$z)

      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.8900 -0.6240 -0.0229 -0.0385  0.5190  1.7100
```

Here is the proposed sampling scheme:

```
> str(s.final)

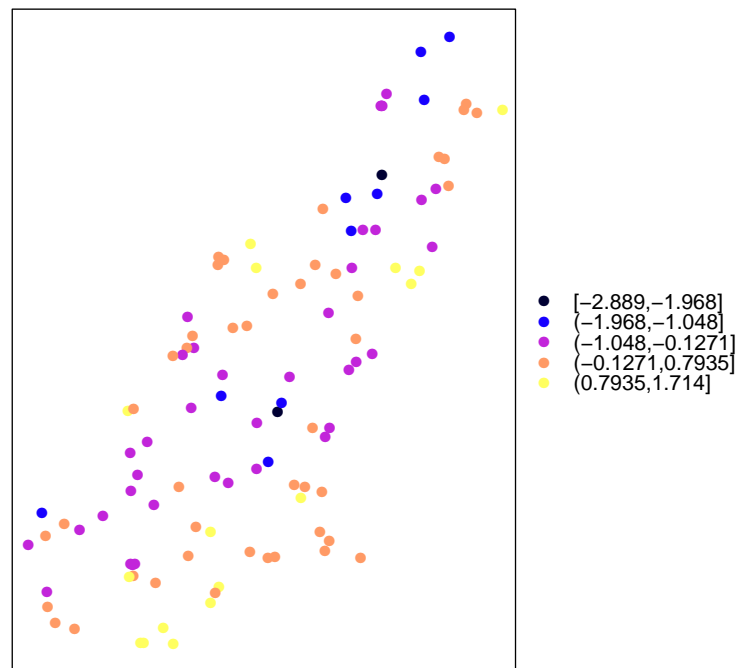
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
 ..@ data      :'data.frame':      110 obs. of  1 variable:
 .. ..$ z: num [1:110] -0.522 1.446 1.593 -0.849 0.99 ...
 ..@ coords.nrs : num(0)
 ..@ coords     : num [1:110, 1:2] 181094 180858 179230 180766 180246 ...
```

```

.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : NULL
.. .. ..$ : chr [1:2] "x" "y"
..@ bbox      : num [1:2, 1:2] 178503 329647 181510 333625
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : chr [1:2] "x" "y"
.. .. ..$ : chr [1:2] "min" "max"
..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
.. .. ..@ projargs: chr NA

> spplot(s.final, col.regions = bpy.colors(64), key.space = "right")

```

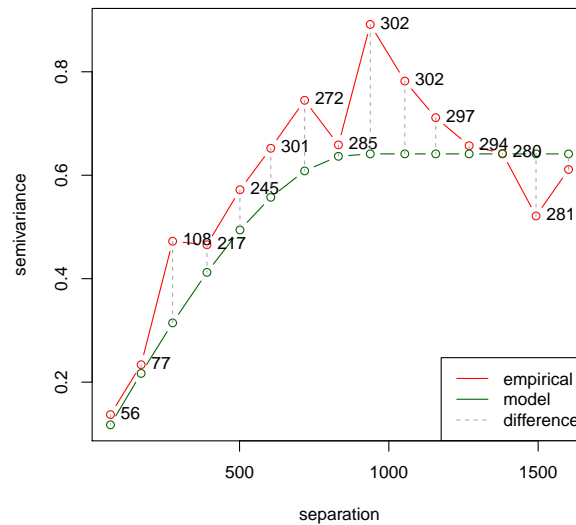


And how closely the empirical variogram using this scheme matches the assumed theoretical variogram:

```

> v <- variogram(z ~ 1, s.final)
> vl <- variogramLine(object = vm, dist_vector = v[, "dist"])
> plot(v$gamma ~ v$dist, type = "b", xlab = "separation",
+      ylab = "semivariance", col = "red", ylim = c(min(v$gamma,
+      vl$gamma), max(v$gamma, vl$gamma)))
> lines(vl$gamma ~ v$dist, type = "b", col = "darkgreen")
> for (i in 1:length(v$dist)) {
+   lines(x = c(v$dist[i], v$dist[i]), y = c(v$gamma[i],
+   vl$gamma[i]), lty = 2, col = "darkgray")
+   text(v$dist[i], v$gamma[i], v$np[i], pos = 4)
+ }
> legend("bottomright", c("empirical", "model", "difference"),
+      lty = c(1, 1, 2), col = c("red", "darkgreen", "darkgray"))

```



Q23 : How well does the new sampling scheme reproduce the variogram model? *Jump to A23 •*

The new sampling scheme places points at proper separations to estimate the true variogram (which may be somewhat different from the assumed field).

Task 73 : Determine how closely the actual distribution of point-pairs over the variogram bins matches the requested distribution. •

```
> tmp <- rbind(t1 <- v$np[1:length(vl.np)], vl.np, t1 -
+   vl.np)
> rownames(tmp) <- c("actual", "expected", "difference")
> colnames(tmp) <- round(vl$dist[1:length(vl.np)], 2)
> print(tmp)

          67.77 170.33 275.9 390.8 501.32
actual      56     77   108   217   245
expected    60     80   120   240   280
difference  -4     -3  -12  -23  -35

> rm(tmp, t1)
```

6.4 Answers

A16 : They are randomly distributed (as planned) with no pattern to the red and green points. *Return to Q16 •*

A17 : Clearly the empirical variogram does not match the model – indeed, it

appears to be pure nugget, as expected by theory, since we assigned values to points at random. [Return to Q17](#) •

A18 : Points with the same colour are now clustered; there are almost no points of the opposite colour inside a cluster. [Return to Q18](#) •

A19 : The empirical variograms match almost perfectly. [Return to Q19](#) •

A20 : Points with the same colour are now clustered; and there is much empty space (areas with no points). This is because of the requirement to have a sufficient number of point-pairs at close spacing, in order to estimate the actual variogram model. [Return to Q20](#) •

A21 : The empirical variograms match somewhat, but there is quite a bit of deviation from the model. [Return to Q21](#) •

A22 : The initial placement reproduces the model well up to about 700 m, but then has much higher variability (total sill). [Return to Q22](#) •

A23 : It matches well at the closer lags (the ones where we specified the number of point-pairs), less well at further lags, which are less important for variogram estimation. [Return to Q23](#) •

7 Challenges

There is no self-test for this exercise; rather a set of challenges. Experiment with the optimization in one of more of the following ways:

- Change the variogram model form and/or range: what happens with a very short- and long-range variogram, relative to the study area size?
- Change the stopping criterion: increase or decrease the number of steps with unchanged fitness before stopping, or use a different stopping criterion.
- Change the number of sampling points and then the starting scheme; does the solution converge earlier or later with more points?
- Change the fixed points; obviously they will affect the solution; does a regular grid of fixed points lead to a regular grid of additional points?
- Change the resolution of the discretization grid; how coarse can it be without affecting the solution?
- Change some simulation parameters:
 1. the proportional decay in the temperature `s.alpha` (must be < 1 , default in §3.1 was set to 0.99);

2. the decay factor for maximum distance, `s.decay` (default 192; a lower factor implies faster decay);
3. the initial probability of accepting a worse scheme, `s.prob` (default 0.8);
4. the proportion of the bounding box to use for the initial maximum shift distance (default $\pm\sqrt{2}/4$).

These will all affect the speed and “noise” of the solution. How do these changes affect the solution?

- Repeat the optimization of §4 for your own country, province or state. Experiment with different numbers of points and assumed models of spatial dependence.

And of course you can apply simulated annealing to your own sampling plan.

Finally, you can implement simulated annealing optimization for KED (§5).

References

- [1] R. S. Bivand, E. J. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*. UseR! Springer, 2008. <http://www.asdar-book.org/>. 3, 4
- [2] P. Bogaert and D. Russo. Optimal spatial sampling design for the estimation of the variogram based on a least squares approach. *Water Resources Research*, 35(4):1275–1289, 1999. 2
- [3] D. J. Brus and G. B. M. Heuvelink. Optimization of sample patterns for universal kriging of environmental variables. *Geoderma*, 138(1-2): 86–95, 2007. 11, 12, 50
- [4] S. Kirkpatrick. Optimization by simulated annealing: quantitative studies. *Journal of Statistical Physics*, 34:975–986, 1984. 11
- [5] A. B. McBratney and R. Webster. The design of optimal sampling schemes for local estimation and mapping of regionalized variables - II. *Computers & Geosciences*, 7(4):335–365, 1981. 1, 29
- [6] A. B. McBratney, R. Webster, and T. M. Burgess. The design of optimal sampling schemes for local estimation and mapping of regionalized variables - I. *Computers & Geosciences*, 7(4):331–334, 1981. 1, 29
- [7] W. G. Müller and D. L. Zimmerman. Optimal designs for variogram estimation. *Environmetrics*, 10(1):23–37, 1999. 2
- [8] J. W. van Groenigen. Sampling strategies for effective variogram estimation. In *Constrained optimisation of spatial sampling*, ITC Publication 65, pages 105–124. ITC, Enschede, NL, 1999. 2
- [9] J. W. van Groenigen. *Constrained optimisation of spatial sampling*. ITC, Enschede, NL, 1999. 2
- [10] J. W. van Groenigen. The influence of variogram parameters on optimal sampling schemes for mapping by kriging. *Geoderma*, 97(3-4):223–236, 2000. 1, 2, 4, 23
- [11] J. W. van Groenigen, A. Stein, and R. Zuurbier. Optimization of environmental sampling using interactive gis. *Soil Technology*, 10(2):83–97, 1997. 2
- [12] A W Warrick and D E Myers. Optimization of sampling locations for variogram calculations. *Water Resources Research*, 23(3):496–500, 1987. 51
- [13] R. Webster and M. A. Oliver. *Geostatistics for environmental scientists*. Wiley & Sons, Chichester, 2001. 2
- [14] R. Webster, S. J. Welham, J. M. Potts, and M. A. Oliver. Estimating the spatial scales of regionalized variables by nested sampling, hierarchical analysis of variance and residual maximum likelihood. *Computers & Geosciences*, 32(9):1320–1333, 2006. 2

Index of R Concepts

bbox (package:sp), 6
break, 29
break operator, 16
byrow argument (matrix function), 4

capture.output, 28
cat, 28
close, 28
CRS (package:rgdal), 41

data.frame, 8
debug.level argument (krige function), 28

file, 28
fullgrid (package:sp), 6, 11
function, 24

gridded (package:sp), 6, 11
gstat package, 43

is.logical, 27
is.na, 16

krige (package:gstat), 8

list, 4
load, 28, 42

map (package:mapdata), 40
map2SpatialPolygons (package:maptools), 40
mapdata package, 40
maptools package, 40
matrix, 4
max, 7
mean, 7, 24

n argument (spsample function), 29
ncol argument (matrix function), 4
nrow argument (matrix function), 4

over (package:sp), 4, 16

Polygon (sp class), 4
Polygons (sp class), 4, 5
print, 28
proj4string argument (map2SpatialPolygons function), 41

rbind, 27, 35

rbinom, 55
rep, 8
repeat, 29
repeat operator, 16
returns, 24
rgdal package, 41
runif, 16

sample, 16
save, 28
set.seed, 8, 35
source, 28
sp package, 40, 41
SpatialGrid (sp class), 6
SpatialGridDataFrame (sp class), 11
SpatialPixels (sp class), 6
SpatialPoints (sp class), 6, 7
SpatialPointsDataFrame (sp class), 8, 9, 24, 25, 27, 64
SpatialPolygons (sp class), 4, 5, 40
SpatialPolygonsDataFrame (package:sp), 5
SpatialPolygonsDataFrame (sp class), 5, 24
spplot (package:sp), 5
spsample (package:sp), 6, 8, 29, 35
spTransform (package:rgdal), 41
system.time, 18, 31, 33

type argument (spsample function), 29
type sp argument, 35

variogramLine (package:gstat), 57
variogramModel class, 52
vgm (package:gstat), 24, 52