Applied geostatistics
Exercise 8a: Spatial simulated annealing

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

June 26, 2010

Contents

1 Introduction **1**

2 SSA Optimization for Ordinary Kriging **3**
2.1 Define the study area . . . 4
2.2 Model of spatial dependence . . . 6
2.3 Fitness function . . . 6
2.4 Setup for SSA . . . 7
2.5 The main SSA loop: one time . . . 11
2.5.1 Maximum distance to move a point . . . 11
2.5.2 Moving one point . . . 14
2.5.3 Plotting a changed scheme . . . 15
2.5.4 Accepting or rejecting the new scheme . . . 15
2.6 The main SSA loop: iterations . . . 16
2.7 Showing the evolution of the scheme . . . 17
2.8 Answers . . . 20

3 SSA in practice: an "optimal" scheme **20**
3.0.1 Another optimization criterion . . . 26
3.0.2 Another starting configuration . . . 28
3.1 Optional exercises . . . 32
3.2 Answers . . . 33

Version 1.0 Copyright © 2009 ITC; 2010 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).

4 Self-test 33

References 34

Index of R concepts 35

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. An example would be sampling a small polluted site where travel costs between observations are not important.

The 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 [8];
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 [8];

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.

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 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 [11] and series of papers [8, 9] 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. 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.

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 [13, 12], transects with variable spacing, or a similar design [10, 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 undersampled 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

In this section we develop SSA for ordinary kriging, first in a square study area and then in an arbitrary polygon. We use R spatial classes [1] because this is essentially 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 [8, eqn. 5]; the probability $P(S_0 \rightarrow S_1)$ of accepting the new scheme is:

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

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 actual annealing, i.e. the cooling of a metal into a crystalline structure with minimum energy. During this, molecules jump around the structure, with the maximum distance controlled by the actual temperature.

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

We first load the necessary libraries:

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

2.1 Define the study area

We illustrate the simplest case with a square study area of size 1 by 1. Although this is very simple, the same technique could be used for any polygon.

We will use spatial overlay 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 in 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 coordinates with the first and last identical.

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

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

```
+     ncol = 2))), ID = 1)))
> summary(p)

Object of class SpatialPolygons
Coordinates:
   min max
r1   0   1
r2   0   1
Is projected: NA
proj4string : [NA]
```

To plot this polygon with `spplot` it must have at least one attribute. We can use the polygon ID as the attribute, and promote the object to `SpatialPolygonsDataFrame` with method `SpatialPolygonsDataFrame`:

```
> 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] "r1" "r2"
  .. .. ..$ : 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; this can be computed as the diagonal of the bounding box, 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 small blocks). 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.

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:

```
> (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 be 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.

```
> obj.k <- function(scheme, f) {
+     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) 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. Here we arbitrarily set it 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. We start with a random discretization of the study area, using the `spsample` 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, we set the random number seed with the `set.seed` function,

```
> 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
```

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

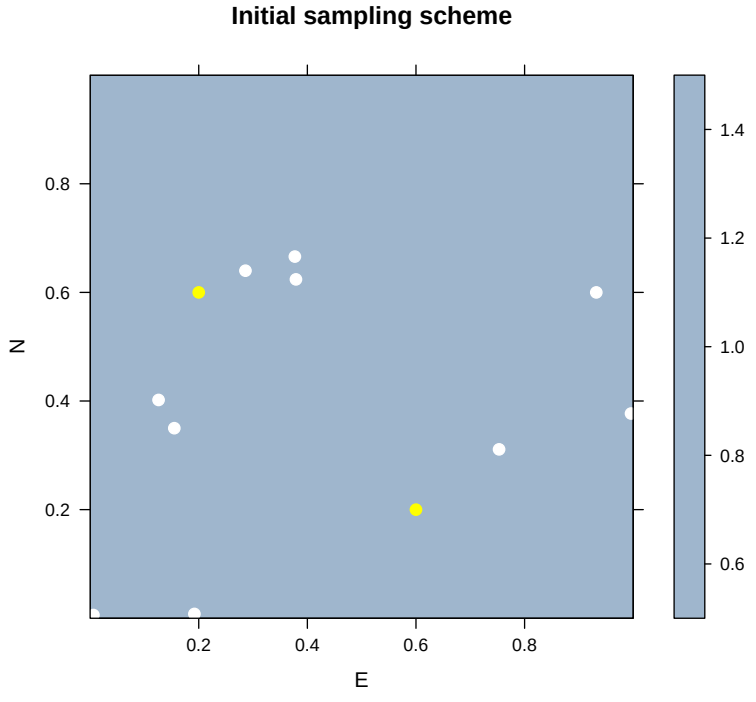
Q2 : *Why are the actual data values at these locations not necessary for SSA?* *Jump to A2* •

Also, a list of the fixed points (e.g. from a previous study); here we just define two 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
```

We can see 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, using the mean kriging variance (MEAN_OK) as our criterion:

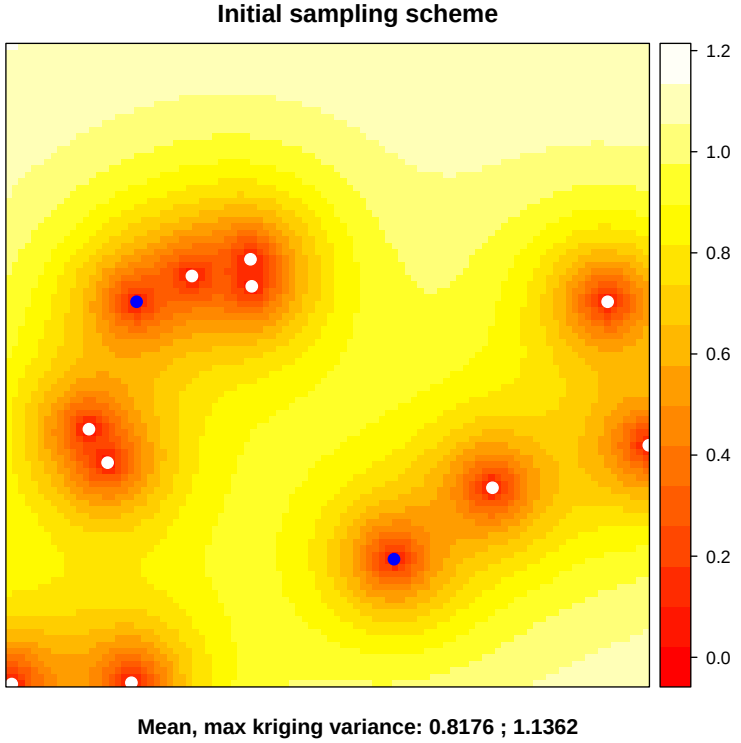
```
> (fitness <- obj.k(rbind(fix.pts, scheme), mean))
```

```
[using ordinary kriging]
[1] 0.81764
```

Q3 : *What are the units of this number? What does it measure?* *Jump to A3* •

We 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)

[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("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: Note the use of the `gridded` and `fullgrid` methods to 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:

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

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]:

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

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]. It can be calculated from this probability, normalized by the logarithm of the average increase in fitness when all increases are accepted [3, Eqn. 7]; this inverts Equation 3:

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

where we have fixed the probability p_0 at 0.8, an empirical value.

To compute this average, we shift points and evaluate the fitness 100 times, and average the improvements over the starting fitness. The maximum shift in both x and y is set at $\sqrt{2}/2$ of the diagonal distance across the bounding box.

```
> max.shift <- round(max.dist * sqrt(2)/2, prec)
> n <- 100
> sum.delta.fit <- 0
> for (i in 1: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(overlay(p, pt.new)))
+             break
+     }
```

```
+     scheme.try <- scheme
+     scheme.try@coords[pt, ] <- c(xnew, ynew)
+     fitness.new <- obj.k(rbind(fix.pts, scheme.try),
+         mean)
+     sum.delta.fit <- sum.delta.fit + (fitness.new - fitness)
+ }
> mean.delta.fit <- sum.delta.fit/n
> rm(max.shift, n, i, pt, pt.old, xnew, ynew, scheme.try,
+     fitness.new)
```

> **Note:** One hundred (100) steps is of course a small sample of possible changes, but it should give a reasonable estimate of the fitness of initial changes.

The average change in fitness after one random move from the random starting scheme is:

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

```
[1] "Average change in fitness: -0.0057"
```

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

The starting temperature is then the average change in fitness, normalized by the logarithm of the desired probabilty of accepting a poorer scheme (here, 0.8).

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

```
[1] 0.025514
```

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

So 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; here we set it to 1% reduction of the previous value.

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.

We set 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:

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

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
```

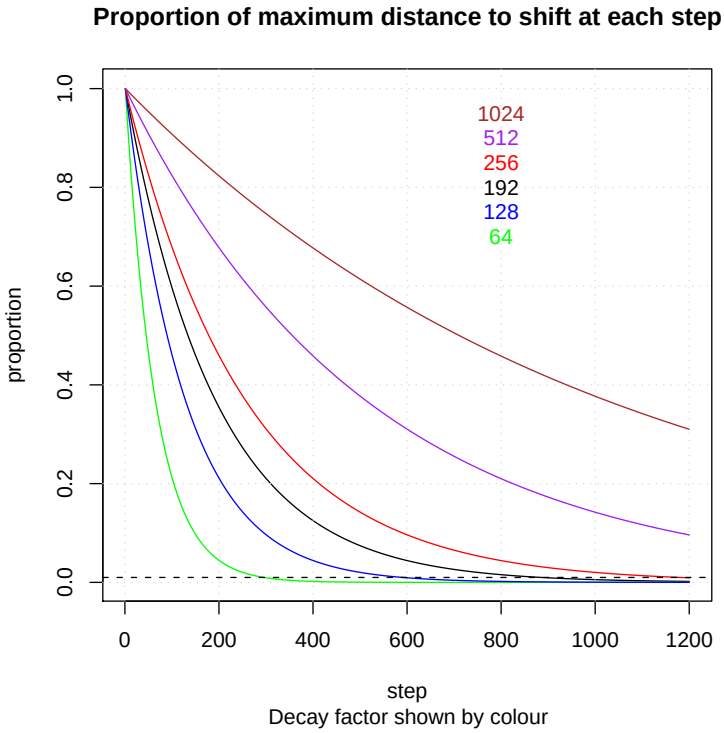
for constants 256, 512, and 128 respectively.

> **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 in the "sphere of influence" of other points, rather than adjusting a point that is more-or-less in the right place.

We visualize this decay function; this shows that the chosen factor (256) will be moving points very little after about step 1000; we can 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
```

```
> 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, 0.7, "64", col = "green")
> text(800, 0.75, "128", col = "blue")
> text(800, 0.8, "192")
> text(800, 0.85, "256", col = "red")
> text(800, 0.9, "512", col = "purple")
> text(800, 0.95, "1024", col = "brown")
> abline(h = 0.01, lty = 2)
```



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 $\pm\sqrt{2}/4$ of the diagonal distance across the bounding box. For a square bounding box, this is just another way to write half of the linear dimension (side) of the box. This is also an empirical factor, which could be reduced if there were many points. Also, with a smaller value points could still move across the box, but in several steps.

```
> (max.shift <- round((max.dist * sqrt(2)/4) * exp(-(step -
+     1)/256), prec))

[1] 0.5
```

Since we are on the first step, this will just equal 0.5; 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

The next step is to select a point to move; this is a random selection, using the `sample` function, from the list of points:

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

```
[1] 2
```

We save the coordinates of the selected point to plot the shift:

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

We next 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. We ensure that the shifted point is within the study area with the `overlay` method, which returns the polygon number containing a point, or `NA` if it is outside any polygon. We can 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:

```
> 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(overlay(p, pt.new)))
+         break
+ }
```

We can now 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))
```

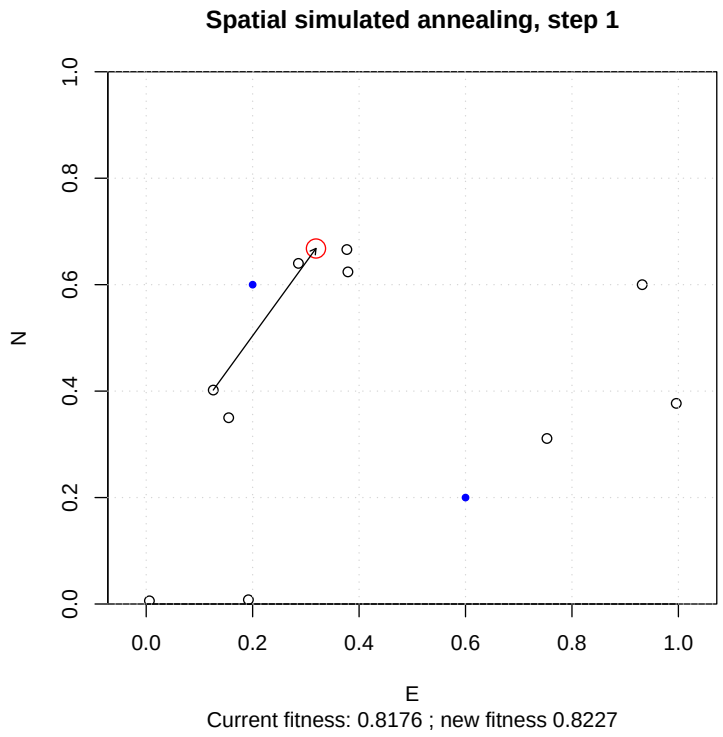
```
[using ordinary kriging]
[1] 0.8227
```

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. Here we 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 why the proposed scheme has a higher 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:

```
> 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] "Worse scheme, rejected"
```

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 200 steps, although we will not have reached an optimum. This will still take a few minutes; we use the `system.time` function to see how much.

```
> n.steps <- 200
> fits <- matrix(nrow = n.steps, ncol = 4)
> colnames(fits) <- c("actual", "proposed", "max.shift",
+     "temperature")
```

Now the main loop:

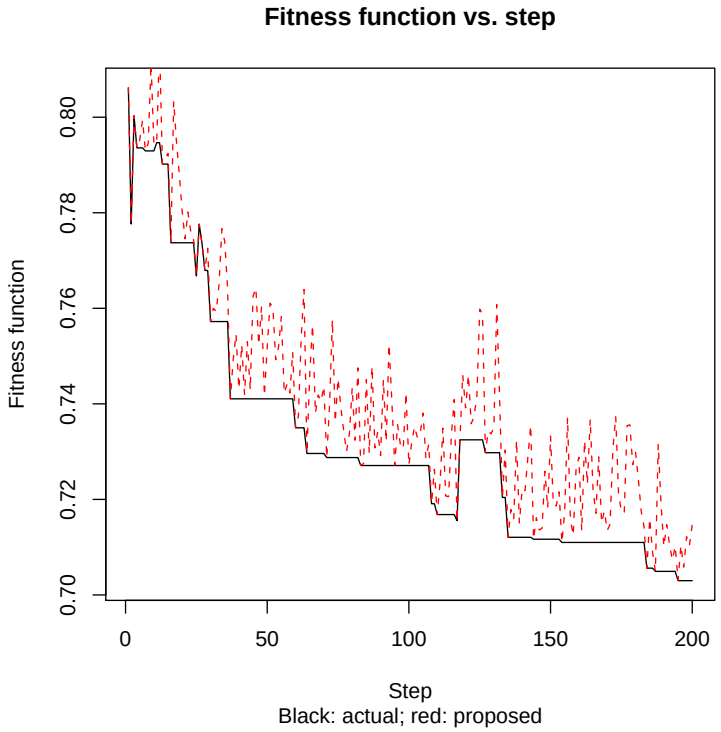
```
> (system.time(for (step in 1:n.steps) {
+     max.shift <- max.dist * 0.5 * round(exp(-(step -
+         1)/256), prec)
+     metr.temp <- metr.temp * alpha
+     pt <- sample(1:n.pts, 1)
+     pt.old <- coordinates(scheme)[pt, ]
+     repeat {
+         xnew <- round(pt.old[1] + runif(1, min = -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(overlay(p, pt.new)))
+             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
+ }))
```

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:

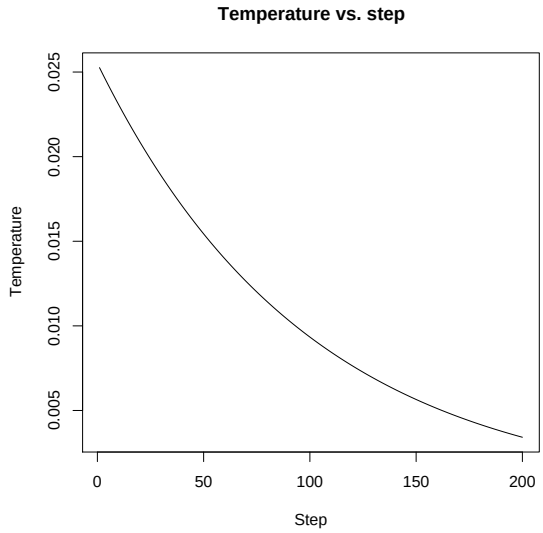
```
> plot(1:n.steps, fits[, "actual"], xlab = "Step", ylab = "Fitness function",
+     type = "l", main = "Fitness function vs. step", sub = "Black: actual; red: proposed")
> lines(1:n.steps, fits[, "proposed"], type = "l", col = "red",
+     lty = 2, pch = 3)
```



It's clear that the scheme in general improves, with local jumps to poorer fitness to avoid a local optimum.

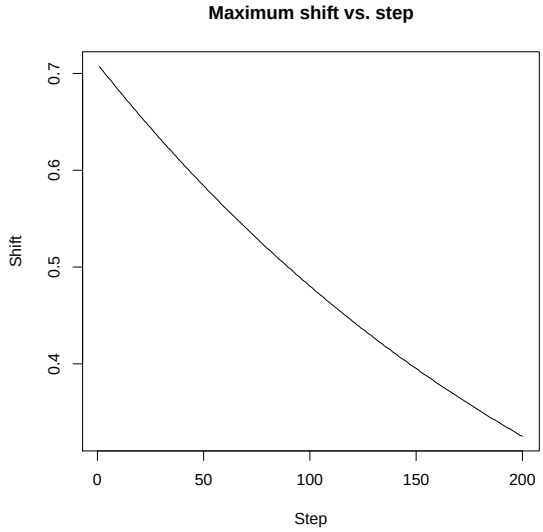
We can plot the evolution of the temperature:

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



The lowering temperature affects the maximum shift; we can graph this:

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

[1] "Maximum shift after 200 : 0.3253"
```



Q6 : *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 A6* •

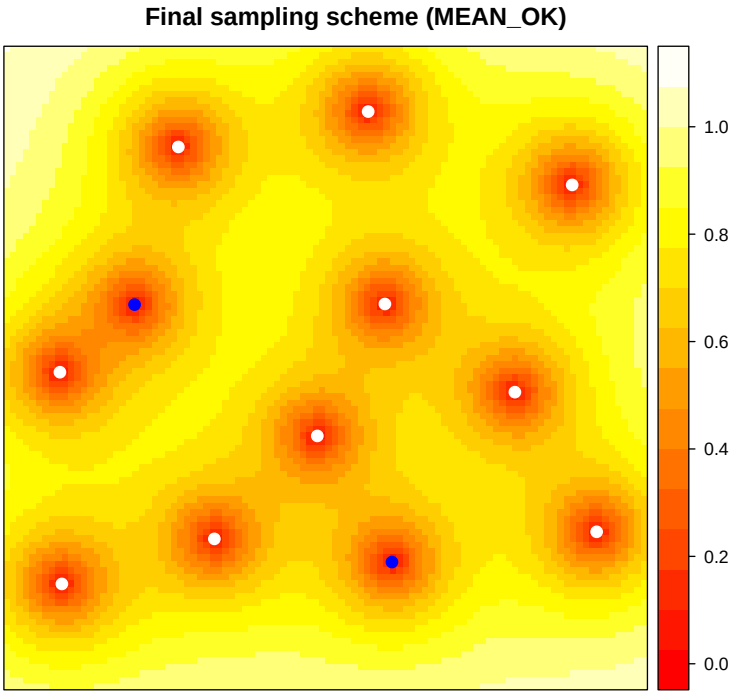
We display the final scheme, with the optimality criterion (kriging variance) as background:

```
> print(scheme)
```

```
      coordinates z
1  (0.087, 0.166) 0
2  (0.324, 0.236) 0
3  (0.484, 0.396) 0
4   (0.88, 0.786) 0
5    (0.563, 0.9) 0
6  (0.268, 0.845) 0
7  (0.791, 0.464) 0
8  (0.084, 0.495) 0
9  (0.589, 0.601) 0
10 (0.918, 0.247) 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)))
```



Q7 : *How much has this configuration changed from the original random configuration? Does it appear optimal? Why or why not?* *Jump to A7* •

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.*

Note that [8] 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.* *Return to Q1* •

A2 : *The fitness function only depends on the kriging prediction variance, and this does not depends 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 a dummy.* *Return to Q3* •

A4 : *The original fitness was* 0.8179*; the fitness of the proposed new scheme is* 0.8230*, which is worse. Of course, with a different random number, the proposed scheme might be better.* *Return to Q4* •

A5 : *The shifted point is very close to three other points and is not in an "unoccupied" region. The original point is close to only one other point and is on the convex hull (outside of the point set) so should reduce the kriging prediction variance to its outside.* *Return to Q5* •

A6 : *The maximum shift now allowed is 0.3253 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 Q6* •

A7 : *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 Q7* •

3 SSA in practice: an "optimal" scheme

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.

We set up the main loop as a **function** with some required arguments and some options; this allows us to call it for different scenarios without changing the code.

The **arguments** are:

`s.init` : the initial sampling scheme, as a `SpatialPointsDataFrame` including a (dummy) variable named `z`;

`s.fix.pts` : any fixed points, also as a `SpatialPointsDataFrame` including a (dummy) variable named `z`, otherwise `NULL`

`s.area` : the (set of) polygons comprising the study area, as a `SpatialPolygonsDataFrame`;

`s.res` : resolution of the discretization of `s.area` for computing the fitness by kriging;

`s.model` : the variogram model, as produced by the `vgm` function;

`s.f` : the fitness criterion function, default `mean`;

`s.decay` : the distance decay factor, default `256`;

`s.alpha` : the cooling parameter, default `0.99`;

`eq.steps` : stopping criterion: how many steps of unchanged fitness, default 200;

The function **returns** the final scheme, as a `SpatialPointsDataFrame`.

```
> ssa.ok <- function(s.init, s.fix.pts, s.area, s.res,
+     s.model, s.f = mean, s.alpha = 0.99, s.decay = 256,
+     eq.steps = 200) {
+     obj.k <- function(pts, f) {
+         k <- krige(z ~ 1, loc = pts, model = s.model,
+             newdata = grid)
+         return(f(k$var1.var))
+     }
+     metr.temp.init <- function(n.steps) {
+         s <- s.init
+         max.shift <- max.dist * sqrt(2)/4
+         sum.fit <- 0
+         for (i in 1:n.steps) {
+             pt <- sample(1:n.pts, 1)
+             pt.old <- coordinates(scheme)[pt, ]
+             repeat {
+                 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)
+                 if (!is.na(overlay(s.area, pt.new)))
+                   break
+             }
+             s.try <- s
+             s.try@coords[pt, ] <- c(xnew, ynew)
+             sum.fit <- sum.fit + obj.k(rbind(s.fix.pts,
+                 s.try), s.f)
+         }
```

```
+         return(((sum.fit/n.steps) - fitness)/log(0.8))
+     }
+     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)
+     grid <- spsample(s.area, n = s.res^2, type = "regular")
+     gridded(grid) = T
+     fullgrid(grid) = T
+     fitness <- obj.k(rbind(s.fix.pts, s.init), s.f)
+     metr.temp <- metr.temp.init(50)
+     print(paste("Initial Metropolis temperature:", round(metr.temp,
+         4)))
+     fits <- matrix(nrow = 0, ncol = 4)
+     colnames(fits) <- c("actual", "proposed", "max.shift",
+         "temperature")
+     step <- 0
+     equal.steps <- 0
+     s <- s.init
+     repeat {
+         max.shift <- (max.dist * sqrt(2)/4) * exp(-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
+             if (!is.na(overlay(p, pt.new)))
+                 break
+         }
+         step <- step + 1
+         s.try <- s
+         s.try@coords[pt, ] <- c(xnew, ynew)
+         fitness.new <- obj.k(rbind(s.fix.pts, s.try),
+             s.f)
+         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)
+         }
+         fits <- rbind(fits, c(fitness, fitness.new, max.shift,
+             metr.temp))
+         if (equal.steps >= eq.steps)
+             break
+     }
+     plot(1:length(fits[, 1]), fits[, "actual"], xlab = "Step",
+         ylab = "Fitness function", type = "l", main = "Fitness function vs. step",
```

```
+         sub = "Black: actual; red: proposed")
+     lines(1:length(fits[, 1]), fits[, "proposed"], type = "l",
+         col = "red", lty = 2, pch = 3)
+     return(s)
+ }
```

Again, we set up so your results look the same as the ones here, although you are welcome to try other starting schemes and fixed points.

First, the study area:

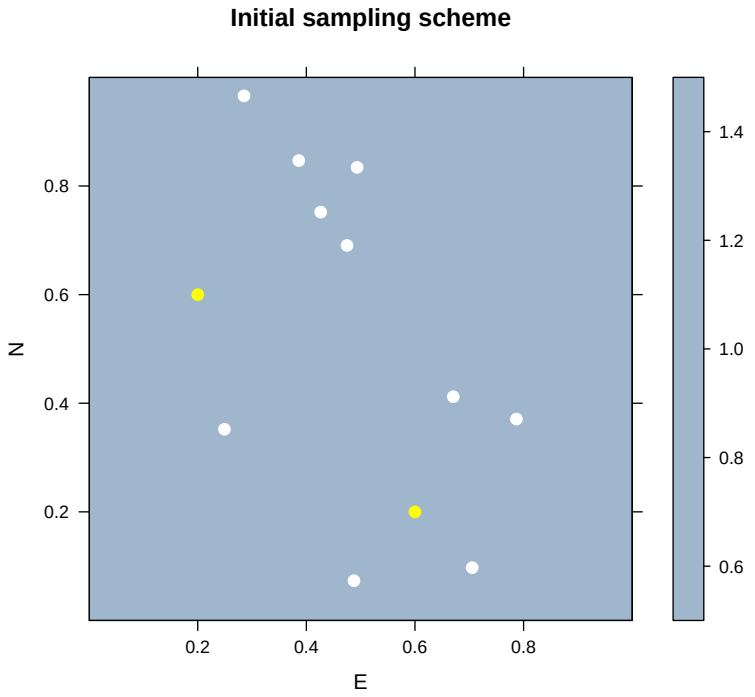
```
> p <- SpatialPolygonsDataFrame(SpatialPolygons(list(Polygons(list(Polygon(matrix(c(0,
+     0, 0, 1, 1, 1, 1, 0, 0, 0), byrow = T, nrow = 5,
+     ncol = 2))), ID = 1))), data = data.frame(id = 1))
```

Then, the fixed points and initial random sampling scheme:

```
> fix.pts <- data.frame(x = c(0.2, 0.6), y = c(0.6, 0.2),
+     z = 0)
> coordinates(fix.pts) <- ~x + y
> prec <- 4
> n.pts <- 10
> set.seed(619)
> scheme <- SpatialPointsDataFrame(spsample(p, n = n.pts,
+     type = "random"), data.frame(z = rep(0, n.pts)))
> scheme@coords <- round(scheme@coords, prec)
```

We plot the starting scheme:

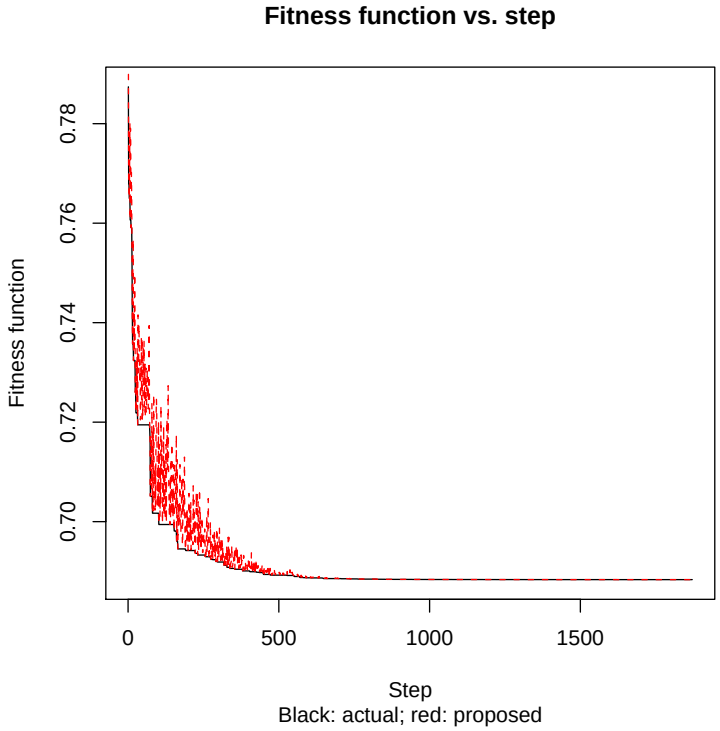
```
> print(spplot(p, xlab = "E", ylab = "N", col.regions = "slategray3",
+     main = "Initial sampling scheme", sub = "Fixed: yellow; Moveable: white",
+     scales = list(draw = T), sp.layout = list(list("sp.points",
+         scheme, col = "white", pch = 20, cex = 1.6),
+         list("sp.points",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



We first try to optimize this scheme, with the fixed points, using the mean kriging variance as the optimization criterion, stopping after 120 steps without improvement. This will take quite some time.

Note that 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.

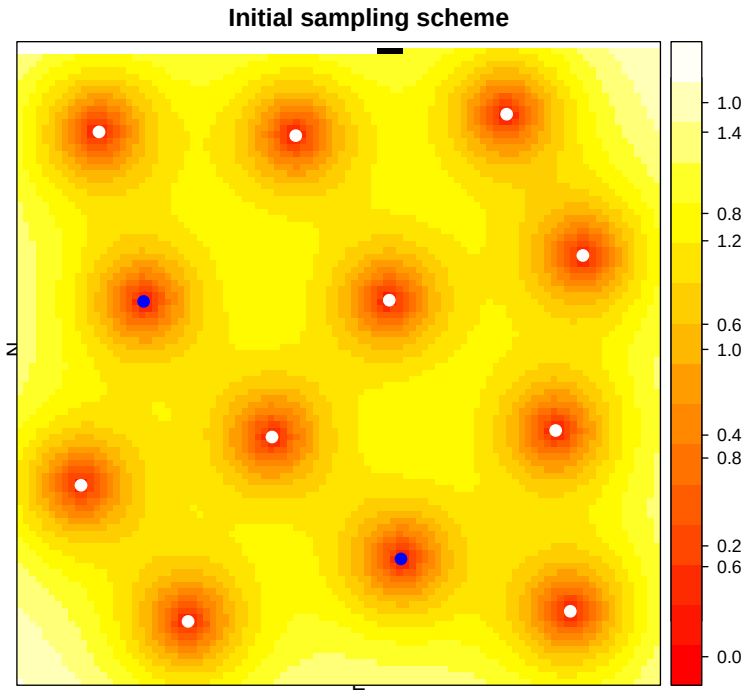
```
> max.dist <- sqrt(((bbox(p)[1, "max"] - bbox(p)[1, "min"])^2) +
+     ((bbox(p)[2, "max"] - bbox(p)[2, "min"])^2))
> vm <- vgm(psill = 1, model = "Exp", range = max.dist/(3 *
+     3), 0)
> system.time(s.final <- ssa.ok(s.init = scheme, s.fix.pts = fix.pts,
+     s.area = p, s.res = 100, s.model = vm, s.f = mean,
+     s.alpha = 0.99, s.decay = 192, eq.steps = 120))
```



We show and print the final configuration:

```
> print(s.final)

        coordinates z
1  (0.8631, 0.1186) 0
2  (0.1307, 0.8635) 0
3  (0.8403, 0.3995) 0
4  (0.3995, 0.3894) 0
5   (0.5817, 0.602) 0
6  (0.8826, 0.6715) 0
7   (0.7641, 0.891) 0
8  (0.1027, 0.3144) 0
9  (0.4364, 0.8577) 0
10 (0.2691, 0.1029) 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)

[using ordinary kriging]

> 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)
```



Fixed: yellow; Moveable: white

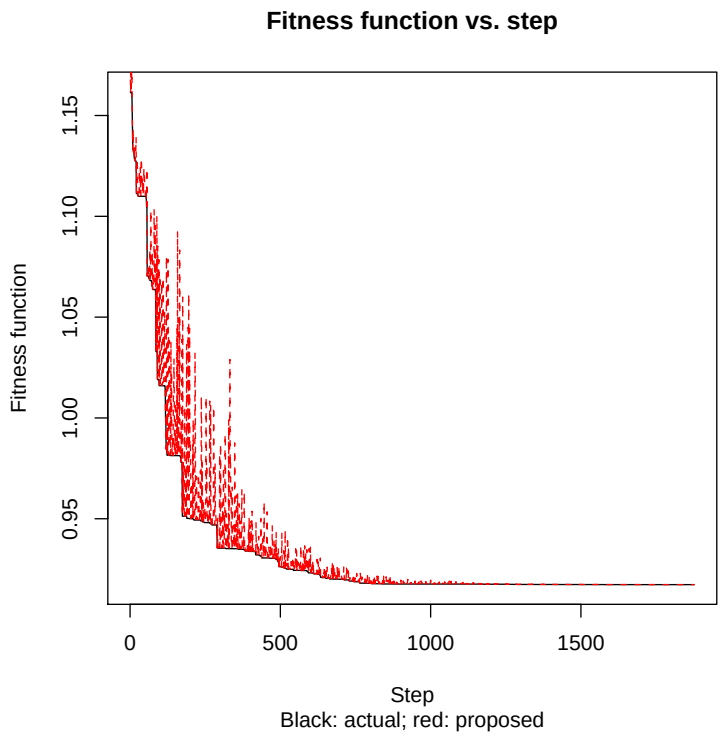
In §3.0.2 we will compare this result to that obtained from another starting configuration, so now we save the result.

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

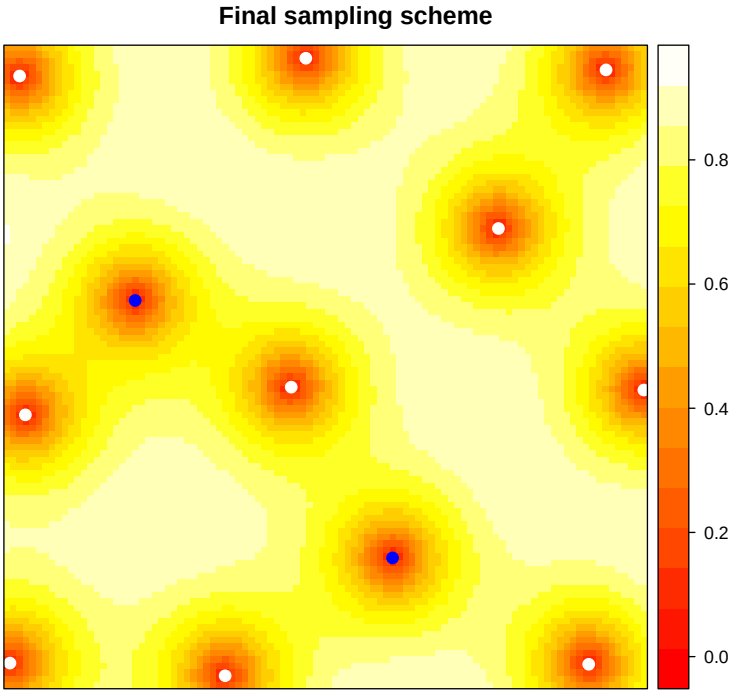
3.0.1 Another optimization criterion

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.

```
> s.final <- ssa.ok(s.init = scheme, s.fix.pts = fix.pts,
+     s.area = p, s.res = 100, s.model = vm, s.f = max,
+     s.alpha = 0.99, s.decay = 192, eq.steps = 120)
```




```
> print(s.final)

        coordinates z
1  (0.4425, 0.4654) 0
2  (0.0054, 0.0366) 0
3  (0.9048, 0.0347) 0
4  (0.3398, 0.0169) 0
5  (0.4652, 0.9764) 0
6  (0.9905, 0.4609) 0
7  (0.0204, 0.9488) 0
8  (0.7645, 0.7119) 0
9  (0.9317, 0.9583) 0
10 (0.0294, 0.4223) 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)

[using ordinary kriging]

> 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)
```

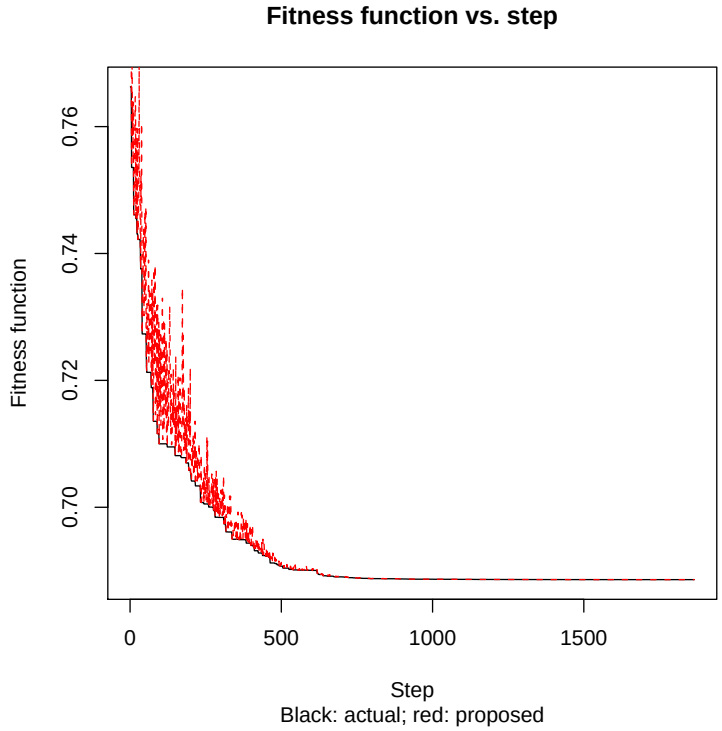


Q8 : *How does this final scheme (criterion MAX_OK) differ from the previous final scheme (criterion MEAN_OK)? Explain the reasons for the difference.* *Jump to A8* •

3.0.2 Another starting configuration

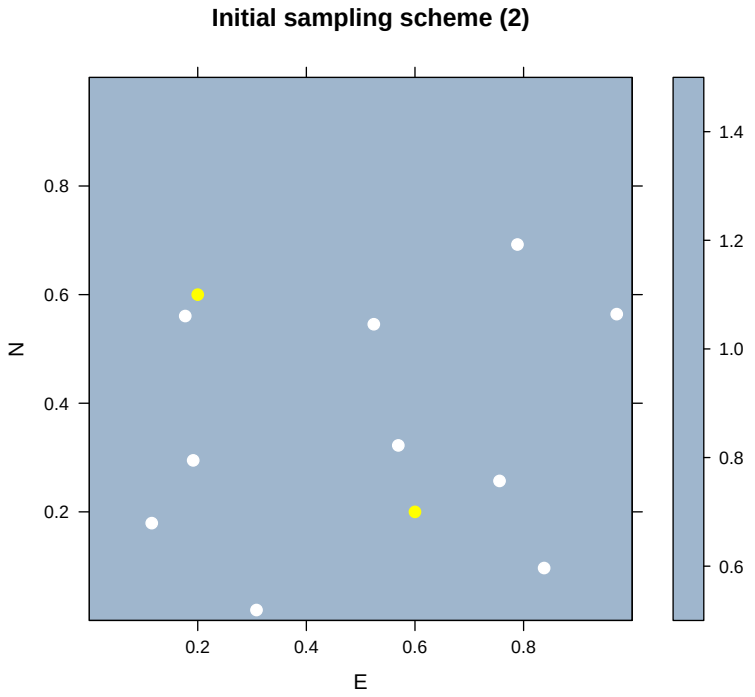
If we have reached an optimimum, it should be the same from any starting configuration. We try again with the **mean** (MEAN_OK) kriging variance, but from another initial scheme; again we use `set.seed` so your results will be the same.

```
> set.seed(7523)
> scheme <- SpatialPointsDataFrame(spsample(p, n = n.pts,
+     type = "random"), data.frame(z = rep(0, n.pts)))
> scheme@coords <- round(scheme@coords, prec)
> s.final <- ssa.ok(s.init = scheme, s.fix.pts = fix.pts,
+     s.area = p, s.res = 100, s.model = vm, s.f = mean,
+     s.alpha = 0.99, s.decay = 192, eq.steps = 120)
```



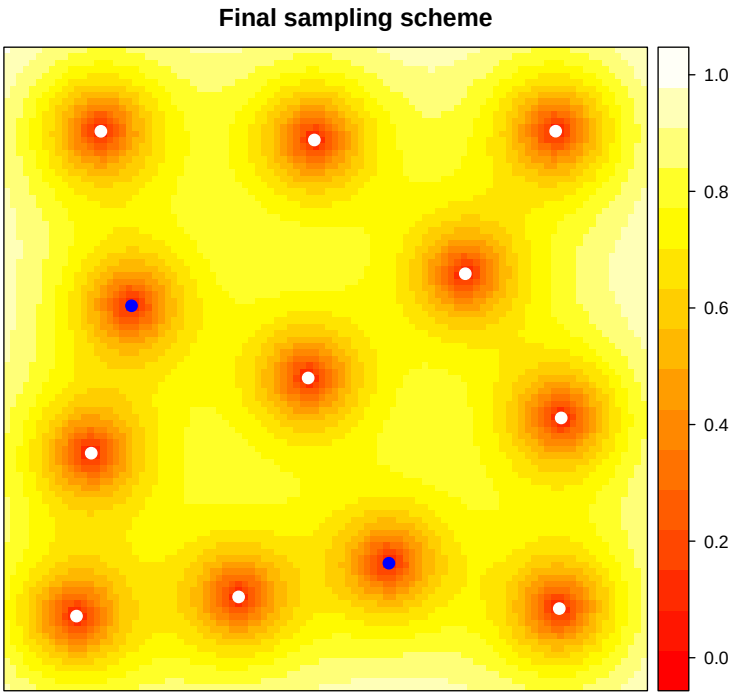
As before, we plot the starting scheme:

```
> 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))))
```



Fixed: yellow; Moveable: white

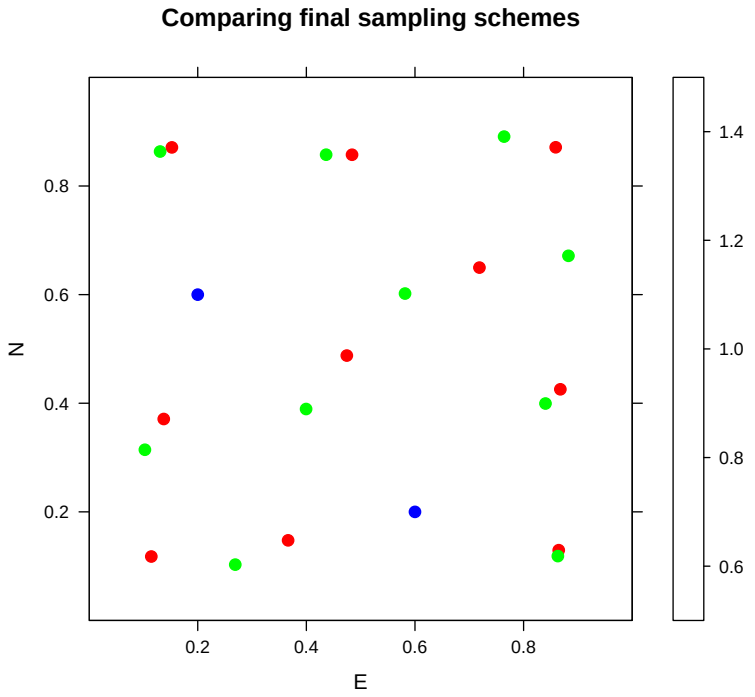
```
> print(s.final)

        coordinates z
1  (0.1145, 0.1178) 0
2  (0.7187, 0.6498) 0
3   (0.1374, 0.371) 0
4  (0.8677, 0.4257) 0
5  (0.8647, 0.1295) 0
6  (0.4841, 0.8576) 0
7  (0.3664, 0.1476) 0
8  (0.4745, 0.4878) 0
9  (0.1524, 0.8711) 0
10 (0.8591, 0.8713) 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)

[using ordinary kriging]

> 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)
```



Q9 : *Is there a difference between the two final configurations? If so, what does that imply about our procedures?* *Jump to A9* •

We can visualize the difference between the final schemes:

```
> print(spplot(p, xlab = "E", ylab = "N", col.regions = "white",
+     main = "Comparing final sampling schemes", sub = "Fixed: blue; Moveable: green (scheme
+     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))))
```



Fixed: blue; Moveable: green (scheme 1), red (scheme 2)

We now clean up from this section:

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

3.1 Optional exercises

You can experiment with the optimization in many ways:

- Change the variogram model form and/or range: what happens with a very short-range variogram?
- Change the stopping criterion: increase or decrease the number of steps with unchanged fitness before stopping (argument `eq.steps`);
- Change the number of points (recompute `n.pts` and then the starting scheme; does the solution converge earlier or later with more points?
- Change the fixed points;
- Change the resolution of the discretization grid; how coarse can it be without affecting the solution?
- Change some constants: (1) the proportional decay in the temperature `alpha` (must be < 1, was set above to 0.99); (2) the decay factor for maximum distance, `decay` (was set above to 192); (3) the proportion of the bounding box to use for the initial maximum shift distance (was set above to $\pm\sqrt{2}/4$)). These will all affect the speed and "noise" of the solution.

3.2 Answers

A8 : *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 Q8* •

A9 : *Yes, they are different. Ideally, the optimum should be the same. Some parameters of the SSA should be adjusted.* *Return to Q9* •

---

4 Self-test

For this exercise we do not have a set self-test. Instead, you should 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-range variogram?
- Change the stopping criterion: increase or decrease the number of steps with unchanged fitness before stopping (argument `eq.steps`);
- 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 ponts?
- Change the resolution of the discretization grid; how coarse can it be without affecting the solution?
- Change some constants: (1) the proportional decay in the temperature `alpha` (must be < 1, was set above to 0.99); (2) the decay factor for maximum distance, `decay` (was set above to 192); (3) the proportion of the bounding box to use for the initial maximum shift distance (was set above to $\pm\sqrt{2}/4$)). These will all affect the speed and "noise" of the solution. How do these changes affect the solution?

References

[1] Roger S. Bivand, Edzer 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. 2, 10

[4] S. Kirpatrick. Optimization by simulated annealing: quantitative studies. *Journal of Statistical Physics*, 34:975–986, 1984. 10

[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

[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

[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. The influence of variogram parameters on optimal sampling schemes for mapping by kriging. *Geoderma*, 97(3-4):223–236, 2000. 1, 2, 4, 20

[9] J W van Groenigen, A Stein, and R Zuurbier. Optimization of environmental sampling using interactive gis. *Soil Technology*, 10(2):83–97, 1997. 2

[10] Jan-Willem van Groenigen. Sampling strategies for effective variogram estimation. In Jan-Willem van Groenigen, editor, *Constrained optimisation of spatial sampling*, ITC Publication 65, pages 105–124. ITC, Enschede, NL, 1999. 2

[11] Jan-Willem van Groenigen. *Constrained optimisation of spatial sampling*. ITC, Enschede, NL, 1999. 2

[12] R Webster and M A Oliver. *Geostatistics for environmental scientists*. Wiley & Sons, Chichester, 2001. 2

[13] 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), 4
break operator, 14

data.frame, 6

fullgrid (package:sp), 5, 9

gridded (package:sp), 5, 9

is.na, 14

krige (package:gstat), 7

list, 4

max, 6
mean, 6, 20

overlay (package:sp), 14

Polygon (sp class), 4
Polygons (sp class), 4

rep, 7
repeat operator, 14
runif, 14

sample, 13
set.seed, 6, 27
SpatialGrid (sp class), 5
SpatialGridDataFrame (sp class), 9
SpatialPixels (sp class), 5
SpatialPoints (sp class), 5, 6
SpatialPointsDataFrame (sp class), 6, 7, 20, 21
SpatialPolygons (sp class), 3
SpatialPolygonsDataFrame (package:sp), 4
SpatialPolygonsDataFrame (sp class), 4, 20
spplot (package:sp), 4
spsample (package:sp), 5, 6
system.time, 16

vgm (package:gstat), 20