OPEN SPAT

Partenariat erasmus plus

Spatial Data with R

(working document - not for public use)

M. Baragatti, Y. Brostaux, J. Cadima, M. Campagnolo, B. Fontez

funded by

UEF

May 2019

Contents

Chapter 1

Introduction

Standard statistical methods that are studied in introductory courses assume the independence of sample observations. For example, confidence intervals or hypothesis tests for the population mean μ of some variable X are based on the assumption that a random sample $(X_1, X_2, ..., X_n)$ of n *independent* observations of that variable is available. Likewise, in a simple linear regression of some response variable Y on a predictor (explanatory variable) X, the standard model assumes that we have n pairs of observations $\{(x_i, Y_i)\}_{i=1}^{n}$, where the n observations of the response variable Y are *independent* random variables, such that $Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$. In this case, the independence of the observations results from the assumption that the *random errors* ϵ_i are *independent* deviations from the underlying straight line with equation $y = \beta_0 + \beta_1 x$.

Independence considerably simplifies statistical analyses. It is quite often a valid assumption, at least as a first approximation to the study of a problem. But there are many instances in which this assumption is clearly not appropriate. One such instance is when data values are observed over time, and observations that are nearby in time are more similar than those made at points in time that are further apart, as is the case with measurements of air temperature in a given location, at 10-minute intervals. Assuming independence in such *time series* measurements would lead us astray if the more standard statistical techniques were used, in ways that will be illustrated below.

In time series, there is a one-dimensional (time) dependence or *temporal autocorrelation*, that must be taken into account. Likewise, in *spatial data*, there is *spatial dependence*, or *spatial autocorrelation*, of the observations that must be taken into consideration. Sometimes, this spatial dependence may be one-dimensional, as would be the case if some water quality measurements were made at different locations along the course of a river. But usually, spatial

data has two-dimensional (sometimes 3-dimensional) dependence. Consider, for example, air temperature measurements taken at different locations on a spatial grid. It is to be expected that observations at points that are close to each other will be more similar than observations at points that are further apart. This *spatial autocorrelation* violates the assumption of independence of the observations, with consequences that impact the statistical tools needed to study such data.

Loosely speaking, *spatial data* vary over some spatial coordinate system, with spatial autocorrelation to a degree that cannot be ignored. More formally, we can talk about a *random spatial process* (variable) Z, in a space $\mathcal{S}$, $\{Z(s), s \in \mathcal{S}\}$, where s denotes a location in space S. Some sort of *model for the spatial autocorrelation* must be specified, if the effects of spatial autocorrelation are to be studied.

As in standard statistical methods, a variable Z may be of different types:

- Z may be a fully *numerical* variable, such as air temperature.
- Z may be a *categorical* variable, as would be the case if $Z(s)$ indicated types of land use over a given region $\mathcal{S}$.
- Z may be an *ordinal* variable, if its values can be ordered, but not on a fully numerical scale. For example, assume that $Z(s)$ measures the intensity of some disease affecting crops in a region $\mathcal{S}$, on a scale of observable effects with k ordered categories, where category 2 indicates a more severe incidence of the pathology than category 1, but where it does not make sense to say that the incidence is "twice as big".
- Z may also be a *binary* variable, that indicates some dichotomy (absence/presence of some characteristic; alive/dead; male/female; etc.); binary variables share some properties of numerical variables, and some properties of categorical variables.

Besides this sort of classification of variables (that is also relevant in a classical statistical setting), it is also possible to classify spatial data into categories that are directly related to the spatial nature of the variables. Cressie [1], classified spatial data $\{Z(s), s \in \mathcal{S}\}$ as belonging to one of three categories:

Geostatistical data, in which $\mathcal{S}$ is a two-dimensional (or three-dimensional) region, over which s varies continuously. For any point $s \in \mathcal{S}$, a value $Z(s)$ exists, even if it is unknown. Consider, for example, that Z represents air temperature over some region $\mathcal{S}$ of the earth's surface. A common problem for such settings is that of *interpolation*,

that is, based on an available set of observations $\{Z(s_{ij})\}_{i,j}$, to obtain estimates for the values of Z in other, unobserved, locations of region $\mathcal{S}$. Several methods dealing with this problem will be addressed in Chapter 6, among them *kriging*.

Lattice (areal) data, in which the nature of the data only makes sense if we consider $\mathcal{S}$ as a collection of *polygons* or *cells*, distributed over space $\mathcal{S}$. Consider variable Z indicating the surface area of countries, or municipalities: the variable's values only make sense for a given area as a whole (hence the expression *areal data*) and they do not vary continuously within the given polygons, or cells. Sometimes the polygons are represented by an individual point within them (usually a central point for each polygon or cell), but this does not change the areal nature of variable Z. If the only spatial reference that is known are these representative or *label points*, it may be helpful to create a lattice of polygons that provide a spatial representation of the full areas, in what is also known as a *tessellation*.

Point data, are data defined by the locations of points in space (such as the location of cities in a region, or of trees in a wooded area) and for which the main topic of interest is the study of the *point patterns* that they define.

This text focuses mostly on geostatistical data.

Chapter 2 begins by discussing why we should worry about the existence of autocorrelation in data. In particular, this section discusses the effects of autocorrelation in time, on standard statistical methods.

Chapter 3 discusses how the R statistical software deals with spatial data, highlighting the standard structures for spatial data, and some important packages and functions to manipulate spatial data in R.

Chapter 4 discusses two-dimensional spatial autocorrelation and introduces key concepts, such as bubble plots, spatial weights matrices, Moran's I and Geary's c coefficients, the semi-variogram and the correlogram, as well as tools when two or more spatial variables are involved.

Chapter 5 studies various regression models for spatially autocorrelated data.

Chapter 6 explains how to do an interpolation map with R (using Inverse Distance Weight or Kriging approaches) and gives a brief introduction to kriging and co-kriging.

Appendix A gives the code used in the simulation.

Appendix B provides information about Coordinate Reference Systems.

Appendix E has exercises for this part of the material.

Appendix E.4.4.3 gives a few key bibliographical references.

Chapter 2

The Effects of Autocorrelation on Standard Statistical Analyses

In order to understand the effects that autocorrelation may have on the results of standard statistical techniques, we will consider (as in Plant [2], 2012) the simplest of all statistical inference problems, regarding the *population mean* μ of some numerical variable Y.

2.1 The classical setting of independent observations

We recall the classical setting for inference regarding a population mean μ. It is assumed that there is a random sample $(Y_1, Y_2, ..., Y_n)$ of n *independent* observations of variable Y. With simple random sampling, each element of the sample Y_i will have a probability distribution that is equivalent to the frequency distribution of Y in the population. The *expected value* of each element Y_i is therefore equal to the population mean μ and the variance of each Y_i will be the *population variance* σ^2, that is,

$$E[Y_i] \;=\; \mu \qquad , \qquad V[Y_i] \;=\; \sigma^2 \tag{2.1}$$

A common assumption is that the population distribution is Normal (Gaussian), in which case all elements of the random sample will have the common distribution $Y_i \sim \mathcal{N}(\mu, \sigma^2)$. This is equivalent to specifying the following *model* for the sample elements:

$$\left\{ \begin{array}{rcl} Y_i & = & \mu + \epsilon_i \\ \epsilon_i & \sim & \mathcal{N}(0, \sigma^2) \qquad (i.i.d.) \end{array} \right. , \tag{2.2}$$

where *i.i.d.* stands for *independent and identically distributed*, indicating that the *random errors* ϵ_i are assumed to be independent.

The standard *estimator* for the population mean μ is the *sample mean*,

$$\overline{Y} \;=\; \frac{1}{n}\sum_{i=1}^{n} Y_i \;. \tag{2.3}$$

As is well known, the expected value and variance of the sample mean are (for independent observations) given by:

$$E[\overline{Y}] \;=\; \mu \qquad , \qquad V[\overline{Y}] \;=\; \frac{\sigma^2}{n} \tag{2.4}$$

The Central Limit Theorem guarantees that, under fairly general conditions, and even for non-Normal populations, we have asymptotically (that is, approximately, for large samples):

$$\frac{\overline{Y}-\mu}{\sqrt{\frac{\sigma^2}{n}}} \;\sim\; \mathcal{N}(0,1) \;. \tag{2.5}$$

The above result underpins inference on μ when the population variance σ^2 is known. This is not usually the case, and the *sample variance* is then used as an *unbiased* estimator of the population variance σ^2 (that is $E[S^2]=\sigma^2$). It is defined as:

$$S^2 \;=\; \frac{1}{n-1}\sum_{i=1}^{n}(Y_i-\overline{Y})^2 \;. \tag{2.6}$$

Although under somewhat more restrictive conditions, replacing σ^2 with its estimator S^2 in (2.5) produces a Student-t distribution that underlies the standard inferential results for μ when σ^2 is unknown:

$$\frac{\overline{Y}-\mu}{\sqrt{\frac{S^2}{n}}} \;\sim\; t_{n-1} \;. \tag{2.7}$$

2.2 The effect of one-dimensional autocorrelation: the AR(1) model

In order to understand the effects on the classical inference of having a sample of n *autocorrelated* observations Y_i, we must first specify a *model* for the autocorrelation. We will assume a simple model for one-dimensional (in time) autocorrelation, that assumes that the deviations from the mean, in the first equation of model (2.2), are no longer independent but rather depend on the previous deviations. The specific model considered is known as a

first-order autoregressive, or AR(1), error model. For $i=1,...,n$:

$$\begin{cases} Y_i &= \mu + \eta_i \\ \eta_i &= \lambda\eta_{i-1} + \epsilon_i \qquad \text{with} \qquad \eta_0 = 0 \\ \epsilon_i &\sim \mathcal{N}(0,\sigma^2) \qquad (i.i.d.)\ , \end{cases} \tag{2.8}$$

The parameter λ specifies the intensity of the autocorrelation along time. If $\lambda=0$, model (2.8) reverts back to model (2.2) with independent errors. When $\lambda>0$ we speak of *positive autocorrelation* since a positive deviation at time i–1 ($\eta_{i-1}>0$) would more likely be followed by another positive deviation at time i and a negative deviation at time i–1 ($\eta_{i-1}<0$) would also be more likely to be followed by a negative deviation at time i. Thus, an observation Y_{i-1} above (below) the mean will more likely be followed by an observation Y_i again above (below) the mean. On the other hand, for $\lambda<0$ we speak of a *negative autocorrelation*: positive deviations would more likely be followed by negative deviations and vice-versa. Negative autocorrelation is less frequent and we will assume $\lambda>0$. In addition, it is to be expected that the effect of a deviation will only be partially felt at a subsequent time point, and that this effect will wear out over time, so we will further assume that $\lambda<1$.

By iterating over previous times in the second equation of model (2.8), it is possible to re-write each observation of Y_i as a function of all previous independent error terms ϵ_j ($j \leq i$):

$$Y_i = \mu + \lambda^{i-1}\,\epsilon_1 + \lambda^{i-2}\,\epsilon_2 + \lambda^{i-3}\,\epsilon_3 + ... + \lambda^2\,\epsilon_{i-2} + \lambda\,\epsilon_{i-1} + \epsilon_i \tag{2.9}$$

$$\Leftrightarrow \quad Y_i = \mu + \sum_{j=1}^{i} \lambda^{i-j}\,\epsilon_j\ . \tag{2.10}$$

This expression, using only the independent errors ϵ_i, ensures easier deductions, and so we re-write the AR(1) deviations model in the following, equivalent, way:

$$\begin{cases} Y_i &= \mu + \sum_{j=1}^{i} \lambda^{i-j}\,\epsilon_j \\ \epsilon_i &\sim \mathcal{N}(0,\sigma^2) \qquad (i.i.d.)\ . \end{cases} \tag{2.11}$$

There are advantages in using vector notation. Denoting a vector of i independent random errors by $\boldsymbol{\epsilon}_i = (\epsilon_1, \epsilon_2, \epsilon_3, ..., \epsilon_{i-1}, \epsilon_i)^t$ and the vector of powers of λ by $\boldsymbol{\lambda}_i = (\lambda^{i-1}, \lambda^{i-2}, \lambda^{i-3}, ..., \lambda, 1)^t$, we can re-write the AR(1) model equation as:

$$Y_i = \mu + \boldsymbol{\lambda}_i{}^t\,\boldsymbol{\epsilon}_i \tag{2.12}$$

2.2.1 Properties of individual observations

First consequences of autocorrelation now become apparent, for any value of λ. The expected value of each element Y_i in the random sample is still the population mean μ:

$$E[Y_i] = E\left[\mu + \sum_{j=1}^{i} \lambda^{i-j}\,\epsilon_j\right] = \mu + \sum_{j=i}^{i} \lambda^{i-j}\,\underbrace{E[\epsilon_j]}_{=0} = \mu\ .$$

But the variance is now different for each element in the sample. From equations (2.11) we have:

$$\begin{aligned} V[Y_i] &= V\left[\mu + \sum_{j=1}^{i} \lambda^{i-j}\,\epsilon_j\right] = \sum_{j=i}^{i} (\lambda^{i-j})^2\,\underbrace{V[\epsilon_j]}_{=\sigma^2} \\ &= \sigma^2\left[(\lambda^2)^{i-1} + (\lambda^2)^{i-2} + (\lambda^2)^{i-3} + \ldots + (\lambda^2)^2 + (\lambda^2) + 1\right]. \end{aligned}$$

Using the expression for the sum of a geometric progression of ratio λ^2, we obtain the following expression, which replaces equation (2.1) from the independent error model:

$$V[Y_i] = \sigma^2\left(\frac{1-\lambda^{2i}}{1-\lambda^2}\right)\ . \tag{2.13}$$

This expression implies several differences in relation to the independent error model (2.2):

- each sample element Y_i now has a different variance;
- *assuming* $0 < \lambda < 1$, we have $V[Y_i] > \sigma^2$ for all $i > 1$, and the larger λ, the larger this variance becomes, for any given i.

It should also be stressed that, again *assuming* $0 < \lambda < 1$, after a sufficiently large initial *transient period* (i large) it is safe to approximate $\lambda^{2i} \approx 0$, and so $V[Y_i] \approx \frac{\sigma^2}{1-\lambda^2}$. Thus, with this model, after an initial transience, the variance of individual observations becomes (approximately) constant, so that we can speak of both *stationary* mean and variance.

As would be expected from a model with autocorrelation, the sample elements Y_i are no longer independent. This can be confirmed by computing the covariances and correlations between different sample elements, which will also be useful for later calculations:

$$Cov[Y_i, Y_j] = Cov\left[\mu + \sum_{k=1}^{i} \lambda^{i-k}\,\epsilon_k\ ,\ \mu + \sum_{m=1}^{j} \lambda^{j-m}\,\epsilon_m\right] = \sum_{k=1}^{i}\sum_{m=1}^{j} \lambda^{i-k}\lambda^{j-m}\,Cov[\epsilon_k, \epsilon_m]$$

Since the error terms $\{\epsilon_i\}$ are independent, only terms for which $k = m$ are non-zero, and in such cases $Cov[\epsilon_k, \epsilon_m] = V[\epsilon_k] = \sigma^2$. Therefore, the double sum is in reality a single summation, with as many terms as $\min\{i, j\}$. *Assuming* $i > j$, we get, from the formula for the sum of a geometric progression (this time with ratio $\frac{1}{\lambda^2}$):

$$Cov[Y_i, Y_j] = \sigma^2 \sum_{k=1}^{j} \lambda^{i+j-2k} = \sigma^2 \lambda^{i+j} \sum_{k=1}^{j} \left(\lambda^{-2}\right)^k = \sigma^2 \lambda^{i-j} . \frac{1-\lambda^{2j}}{1-\lambda^2} = \lambda^{i-j} V[Y_j] \quad (2.14)$$

Expression (2.14) tells us that there are non-zero covariances for any pair of different sample observations. For $0 < \lambda < 1$, these covariances decrease with the difference $i-j$, that is, with the time gap between the observations. As would be expected, $Cov[Y_i, Y_j]$ grows with λ.

From equation (2.14) the correlation coefficient between different observations, $r_{ij} = Cor[Y_i, Y_j]$ (again, for $i > j$) is given by:

$$r_{ij} = \frac{Cov[Y_i, Y_j]}{\sqrt{V[Y_i]\, V[Y_j]}} = \lambda^{i-j} \sqrt{\frac{V[Y_j]}{V[Y_i]}} = \lambda^{i-j} \sqrt{\frac{1-\lambda^{2j}}{1-\lambda^{2i}}} \quad (2.15)$$

After a sufficiently long initial transience, we can assume $\lambda^{2j} \approx 0$ (hence, necessarily $\lambda^{2i} \approx 0$), this simplifies to $r_{ij} \approx \lambda^{i-j}$. The correlation between different observations, Y_i and Y_j, therefore decreases as the time gap $i-j$ between observations grows. For sufficiently large differences $i-j$, the correlation becomes negligible.

Table 2.1 compares the main results, for the independent error and the AR(1) error models, highlighting the latter's characteristics *for large i (after an initial **transient** period).*

2.2.2 Properties of the sample mean

We now turn our attention to the sample mean $\overline{Y}$. We consider a general case, in which a sample of size n is drawn, following model (2.8), but after a transient period of t iterations. The random sample is the following random vector:

$$\mathbf{Y} = (Y_{t+1}, Y_{t+2}, Y_{t+3}, ..., Y_{t+(n-1)}, Y_{t+n})^t . \quad (2.16)$$

The case of no transient period arises as a specific instance, where $t = 0$. On the other hand, *assuming that the sample was taken after a long initial transience*, we are assuming that, for

	Independence	AR(1)
$E[Y_i]$	μ	μ
$V[Y_i]$	σ^2	$\sigma^2 \left(\frac{1-\lambda^{2i}}{1-\lambda^2}\right) \rightarrow \frac{\sigma^2}{1-\lambda^2}$
$Cov[Y_i, Y_j]$ (for $i > j$)	0	$\lambda^{i-j} V[Y_j] \rightarrow \lambda^{i-j} \frac{\sigma^2}{1-\lambda^2}$
r_{ij} (for $i > j$)	0	$\lambda^{i-j} \sqrt{\frac{V[Y_j]}{V[Y_i]}} \rightarrow \lambda^{i-j}$

Table 2.1: Results for the independent, and AR(1), error models. The arrows indicate post-transience results (large $j < i$). For $0 < \lambda < 1$, the AR(1) model is *stationary* in the *mean* and (*after transience*) in the *variance*. Correlations decrease with the time lags $i - j$.

any sample elements Y_{t+i} and Y_{t+j}:

$$\begin{aligned} Y_{t+i} &\sim \mathcal{N}\left(\mu, \frac{\sigma^2}{1-\lambda^2}\right) \quad , \\ Cov(Y_{t+i}, Y_{t+j}) &\approx \frac{\sigma^2}{1-\lambda^2} \lambda^{i-j} \quad ; \qquad (2.17) \\ r_{Y_{t+i}, Y_{t+j}} &\approx \lambda^{i-j} \, . \qquad (2.18) \end{aligned}$$

Again, calculations are simpler if we write the sample mean $\overline{Y}$ as a linear (affine) combination of the independent random errors ϵ_j. A direct substitution of the exact expressions in (2.11) gives:

$$\begin{aligned} \overline{Y} &= \frac{1}{n} \sum_{i=1}^{n} Y_{t+i} = \mu + \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{t+i} \lambda^{t+i-j} \epsilon_j \\ \Leftrightarrow \quad \overline{Y} &= \mu + \frac{1}{n} \left[\left(\frac{1-\lambda^n}{1-\lambda} \right) \sum_{j=1}^{t} \lambda^{(t+1)-j} \epsilon_j + \frac{1}{1-\lambda} \sum_{k=1}^{n} [1 - \lambda^{n-(k-1)}] \epsilon_{t+k} \right] . \qquad (2.19) \end{aligned}$$

From this expression it is easy to see that $E[\overline{Y}] = \mu$ (since for all i, $E[\epsilon_i] = 0$). Laborious, but straightforward, algebra gives an exact expression for the variance of $\overline{Y}$ under the AR(1)

error term:

$$V[\overline{Y}] = \frac{\sigma^2}{n^2}\left[\left(\frac{1-\lambda^i}{1-\lambda}\right)^2 \frac{\lambda^2}{1-\lambda^2}(1-\lambda^{2t}) + \sum_{i=1}^{n}\left(\frac{1-\lambda^i}{1-\lambda}\right)^2\right] \quad (2.20)$$

$$= \frac{\sigma^2}{n^2(1-\lambda)^2}\left[(1-\lambda^n)^2\frac{\lambda^2}{1-\lambda^2}(1-\lambda^{2t}) + \sum_{i=1}^{n}(1-\lambda^i)^2\right] \quad (2.21)$$

For any $\boldsymbol{\lambda \in]0,1[}$, the first term inside the square brackets of equation (2.20) (which only exists if there is a transient period $t > 0$) is non-negative, and all (but one) terms in the summation are necessarily greater than 1 (equal, for $i = 1$), and therefore the factor in the square brackets is greater than n. Thus, for any sample size n and any transient period t, we have:

$$V[\overline{Y}] > \frac{\sigma^2}{n}\,. \quad (2.22)$$

Hence, *the variance of the sample mean with the independent error model,* $\frac{\sigma^2}{n}$*, is smaller than the true variance of* $\overline{Y}$ *(given by 2.20), for the AR(1) autocorrelation model.* This underestimation is not very relevant for small values of the autocorrelation parameter λ (λ close to zero), but as λ increases, it can become quite significant. For example, the value of (2.20) is between 3 and 4 times as large as $\frac{\sigma^2}{n}$ if $\lambda = 0.5$, for any sample size greater than 5. For $\lambda = 0.75$ (and $t = 0$), even for a sample size as small as $n = 5$, the ratio of expression (2.20) to $\frac{\sigma^2}{n}$ is already greater than 5, and for larger sample sizes it grows to become 16 times as big.

Although the above result is for an AR(1) model, the underestimation of $V[\overline{Y}]$ by the expression $\frac{\sigma^2}{n}$ is a general feature when autocorrelation is present. It can be thought of as reflecting the fact that, *in the presence of autocorrelation, a sample of size n has, in reality, less than n independent sources of information.*

For large samples, collected after large transient periods (with $\lambda^{2t} \approx 0$), the variance of the sample mean converges to $\frac{\sigma^2}{n\,(1-\lambda)^2}$. In fact, from expression (2.21), and assuming both $\lambda^{2t} \approx 0$ and $\lambda^n \approx 0$, we have:

$$V[\overline{Y}] \approx \frac{\sigma^2}{n^2(1-\lambda)^2}\left[\frac{\lambda^2}{1-\lambda^2} + \left(\sum_{i=1}^{n}(1-2\lambda^i+\lambda^{2i})\right)\right]$$

$$\approx \frac{\sigma^2}{n\,(1-\lambda)^2}\left[1 + \frac{2}{n}\frac{\lambda^2}{1-\lambda^2} - \frac{2}{n}\frac{\lambda}{1-\lambda}\right]$$

So, for large sample size n, we have:

$$\frac{V[\overline{Y}]}{\frac{\sigma^2}{n\,(1-\lambda)^2}} \;\approx\; 1 + \underbrace{\frac{2}{n}\,\frac{\lambda^2}{1-\lambda^2} - \frac{2}{n}\,\frac{\lambda}{1-\lambda}}_{\approx 0} \;\approx\; 1 \qquad \Leftrightarrow \qquad V[\overline{Y}] \;\approx\; \frac{\sigma^2}{n\,(1-\lambda)^2} \tag{2.23}$$

A useful concept is that of *effective sample size*, n_ϵ, which is defined as the value for which:

$$V[\overline{Y}] \;=\; \frac{\sigma^2}{n_\epsilon} \qquad \Leftrightarrow \qquad n_\epsilon \;=\; \frac{\sigma^2}{V[\overline{Y}]}\;. \tag{2.24}$$

The effective sample size suggests the 'real' size of our sample, in terms of independent observations. After initial transience and for large sample size n, equation (2.23) indicates that the effective sample size converges to $n_\epsilon \approx n(1-\lambda)^2$, as can be seen in Table 2.2, assuming a large transience. *For large n, the effective sample size n_ϵ converges to $n(1-\lambda)^2$* even in the absence of transience.

n	λ										
	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
10	9.82	8.27	6.66	5.20	3.87	2.68	1.63	0.78	0.23	0.03	0.00
50	49.02	40.66	32.26	24.78	18.25	12.67	8.03	4.35	1.67	0.21	0.00
100	98.03	81.16	64.25	49.28	36.25	25.17	16.03	8.85	3.64	0.60	0.00
1000	980.12	810.16	640.25	490.28	360.25	250.17	160.03	89.84	39.61	9.37	0.01
10000	9801.02	8100.16	6400.25	4900.28	3600.25	2500.17	1600.03	899.84	399.61	99.33	0.51

Table 2.2: Table with the effective sample size n_ϵ in an autocorrelated AR(1) process, for various values of true sample size n and of the global autocorrelation strength λ ($0<\lambda<1$), with a large transient period ($t = 10000$). For large true sample size n, the effective sample size converges to $n_\epsilon = n(1-\lambda)^2$.

2.2.3 Properties of the sample variance

The problem of estimating the variance of $\overline{Y}$ when the independent-sample expression $\frac{\sigma^2}{n}$ is used, is compounded by the fact that *the sample variance* S^2 (which is needed to estimate the unknown σ^2) has an *overestimation bias*. In fact,

$$S^2 \;=\; \frac{1}{n-1}\sum_{i=1}^{n}(Y_{t+i}-\overline{Y})^2 \;=\; \frac{1}{n-1}\left[\sum_{i=1}^{n} Y_{t+i}^2 - n\overline{Y}^2\right]$$

$$\Rightarrow \quad E[S^2] \;=\; \frac{1}{n-1}\sum_{i=1}^{n} E[Y_{t+i}^2] \;-\; \frac{n}{n-1}\,E[\overline{Y}^2]$$

Given the general property, for any random variable X, that $E[X^2] = V[X] + E^2[X]$ and since both Y_{t+i} and $\overline{Y}$ share a common mean μ, we have:

$$E[S^2] \quad = \quad \frac{1}{n-1} \sum_{i=1}^{n} V[Y_{t+i}] \; - \; \frac{n}{n-1} V[\overline{Y}] \tag{2.25}$$

Assuming an initial transience for which $\lambda^t \approx 0$, and using expressions (2.13) and (2.21), the expected value of S^2 becomes:

$$E[S^2] \quad = \quad \frac{\sigma^2}{1-\lambda^2} \frac{n}{n-1} \left[1 - \frac{1+\lambda}{n(1-\lambda)} + \frac{2\lambda(1-\lambda^n)}{n^2(1-\lambda)^2} \right] \tag{2.26}$$

For *large samples*, $E[S^2]$ is approximately:

$$E[S^2] \quad \approx \quad \frac{\sigma^2}{1-\lambda^2} \quad > \quad \sigma^2 \; . \tag{2.27}$$

Thus, an unbiased (asymptotic, after transience) estimator of σ^2 with the AR(1) model is:

$$\hat{\sigma}^2 = (1-\lambda)\, S^2 \; . \tag{2.28}$$

2.2.4 Simulations

We illustrate the above results with simulations of the model (2.8). An `R` code for these simulations is given in Appendix A.

The simulation code was initially run with the following parameters: sample size $n = 10\,000$; a transient period of one thousand iterations; population mean zero ($\mu = 0$); and common error variance $\sigma^2 = 1$. Given the fairly long transient period, the process can be considered stationary. Various values of the autocorrelation parameter λ were used, as indicated in Table 2.3 and, for each λ, 10 thousand repetitions (`times`$= 10000$) were considered, giving the means and variances in the Table. The true expected value for the mean of the sample means is $E[\overline{Y}] = \mu = 0$. The variances of the sample means are very close to the true asymptotic value (2.23), $V[\overline{Y}] = \frac{\sigma^2}{n(1-\lambda)^2} = \frac{0.0001}{(1-\lambda)^2}$ and are, in all cases, greater than the variance of $\overline{Y}$ for independent samples ($\frac{\sigma^2}{n} = 0.0001$). Likewise, the mean of the sample variances is always very close to the asymptotic value $E[S^2] = \frac{\sigma^2}{1-\lambda^2} = \frac{1}{1-\lambda^2}$, and are always greater than $\sigma^2 = 1$, which would the expected value with an independent sample. As is to be expected, the deviation from the independent sample values becomes greater, as the autocorrelation parameter λ grows.

λ	$\overline{Y}$		S^2	
	mean	variance	mean	variance
0.01	0.00002	0.00010	1.00002	0.00020
0.10	0.00024	0.00012	1.01007	0.00020
0.20	-0.00010	0.00015	1.04132	0.00024
0.30	0.00006	0.00021	1.09849	0.00029
0.40	-0.00006	0.00028	1.19016	0.00040
0.50	-0.00013	0.00040	1.33294	0.00059
0.60	-0.00020	0.00063	1.56233	0.00103
0.70	0.00035	0.00111	1.96007	0.00228
0.80	0.00004	0.00256	2.77648	0.00708
0.90	0.00032	0.01014	5.24928	0.05194
0.99	-0.01354	0.98780	49.21952	48.29015

Table 2.3: The means and variances of the sample means $\overline{Y}$ and the sample variances S^2, obtained form 10 000 repetitions of simulations of model (2.8). In all 10 thousand repetitions, samples of size $n = 10\,000$ were considered, after a transient period of 1000 iterations. The population mean was taken to be $\mu = 0$ and the error variance $\sigma^2 = 1$. For independent samples, we would expect $E[\overline{Y}] = 0$, $V[\overline{Y}] = 0.0001$ and $E[S^2] = 1$.

A second simulation, based on the AR(1) model and the `R` function in Appendix A, computed 10 000 samples of size $n = 1000$, assuming $\lambda = 0.7$, $\mu = 10$, $\sigma = 3$, and 1000 transient iterations.

Figure 2.1 shows the histogram of the 10 000 sample means $\overline{y}$ that resulted. The red curve gives the theoretical distribution results for an independence model: $\overline{Y} \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right)$. The blue curve is the asymptotic equivalent under the AR(1) model, after transience: $\overline{Y} \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{(1-\lambda)^2\, n}\right)$. It is clear that assuming independence when we are in the presence of autocorrelation (with $\lambda = 0.7$) seriously underestimates the sampling variability of $\overline{Y}$.

Figure 2.2 shows the distribution of the 10 000 sample variances s^2 (in black and white) and of the unbiased (asymptotic, post-transience) estimates given in (2.28), $(1 - \lambda^2)\, s^2$ (in red). Considering that the true variance in this simulation was $\sigma^2 = 9$, the scale of the bias associated with the standard, independence-based, estimator (S^2) is evident. It can also be seen that the sampling variance of $(1 - \lambda^2)\, S^2$ is smaller than that of S^2, by a factor of $(1 - \lambda^2)^2$.

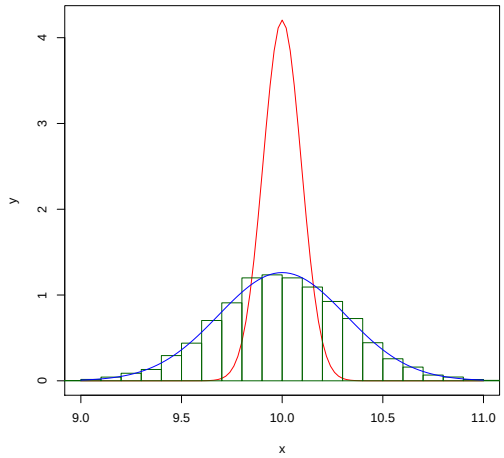


Figure 2.1: Histogram of the distribution of $\overline{y}$, for 10 000 repetitions of size $n = 1000$ samples, in an AR(1) model. Parameters: $\mu = 10$, $\sigma = 3$, $\lambda = 0.7$. Red curve: distribution of $\overline{Y}$ under independence. Blue curve: asymptotic equivalent under AR(1).

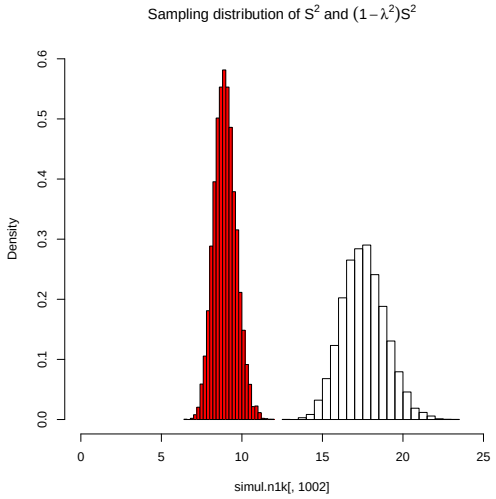


Figure 2.2: Histograms of the distributions of s^2 (in black and white) and $(1-\lambda^2)\,s^2$ (in red), for 10 000 repetitions of size $n=1000$ samples, in an AR(1) model. Parameters: $\mu=10$, $\sigma=3$, $\lambda=0.7$. The true variance is $\sigma^2=9$, and the severe bias of the standard estimator S^2 is evident.

2.2.5 Implications

The implications of these results for the classical inference on a population mean μ are now considered. Assuming an independent sample of size n, $(Y_1, Y_2, ..., Y_n)$, the standard confidence interval (CI) for μ, when the population variance σ^2 is also known, is based on the well-known distributional result $\frac{\overline{Y}-\mu}{\frac{\sigma}{\sqrt{n}}} \sim \mathcal{N}(0,1)$, and is given by:

$$\left] \overline{y} - z_{\alpha/2} \frac{\sigma}{\sqrt{n}} \ , \ \overline{y} + z_{\alpha/2} \frac{\sigma}{\sqrt{n}} \right[. \tag{2.29}$$

But, as was seen in equation (2.23), the large-sample variance of $\overline{Y}$ with the AR(1) model is actually $V[\overline{Y}] \approx \frac{\sigma^2}{(1-\lambda)^2 n}$. The Normality of $\frac{\overline{Y}-\mu}{\frac{\sigma}{(1-\lambda)\sqrt{n}}}$ holds with the AR(1) model, and so the appropriate $(1-\alpha) \times 100\%$ CI would be:

$$\left] \overline{y} - z_{\alpha/2} \frac{\sigma}{(1-\lambda)\sqrt{n}} \ , \ \overline{y} + z_{\alpha/2} \frac{\sigma}{(1-\lambda)\sqrt{n}} \right[. \tag{2.30}$$

This confidence interval is $\frac{1}{1-\lambda}$ times larger than the standard, independence-based, confidence interval.

Consider again the second simulation mentioned in Subsection 2.2.4. The normality QQ-plot (drawn with R's `qqnorm` function) for the 10 000 simulated values of the ratio $\frac{\overline{Y}-\mu}{\frac{\sigma}{\sqrt{n}}}$ is given on the left of Figure 2.3. It suggests that the Normality assumption is appropriate for this quantity, even with the AR(1) model. The confidence interval (2.30) is therefore adequate, but is $\frac{1}{1-\lambda} = \frac{10}{3} = 3\frac{1}{3}$ times as large as the conventional, independence-based CI (2.29). Thus, the standard confidence intervals (and standard hypothesis tests and p-values) will be incorrect in a setting with AR(1) autocorrelation.

As an illustration of the effects, consider the first of the ten thousand repetitions in the simulation that has just been mentioned, for which $\overline{y} = 10.1985884$. The standard 95% CI from equation (2.29) is therefore $]10.01625, 10.38453[$ and does not include the true population mean $\mu = 10$. The more appropriate CI in equation (2.30) is $]9.578793, 10.81838[$ and includes the true value of μ. In the 10 000 samples of the simulation, only 44.5% of the (nominally) 95% independence-based confidence intervals actually contained the true population mean $\mu = 10$, so the use of formula (2.29) for a (supposedly) 95% confidence interval would, in fact, produce something akin to a 45% confidence interval. Using the asymptotic AR(1) 95% confidence intervals (formula 2.30), 94.95% of the 10 000 intervals contained the true population mean $\mu = 10$ (as would be expected with a true 95% confidence interval).

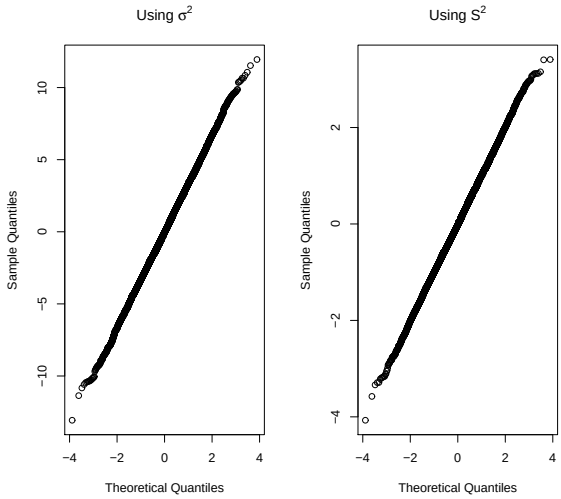


Figure 2.3: The Normality QQ-plots of the quantities underlying the Confidence Intervals for the population mean μ. The plots are based on 10 000 repetitions of samples of size $n = 1000$, after 1000 transient iterations, and assuming $\mu = 10$ and $\sigma^2 = 9$. On the left, the QQ-plot for $\frac{\overline{Y}-\mu}{\frac{\sigma}{\sqrt{n}}}$, and on the right, the QQ-plot for $\frac{\overline{Y}-\mu}{\sqrt{\frac{1+\lambda}{1-\lambda}}\frac{S}{\sqrt{n}}}$.

In a real application, it is unlikely that the true population variance is known. The classical result in such a situation involves replacing the unknown population variance σ^2 with its unbiased (for an independent sample) estimator S^2. The standard CI assumes Normal populations and is based on the result $\frac{\overline{Y}-\mu}{\frac{S}{\sqrt{n}}} \sim t_{n-1}$. For large independent samples, the Student's t distribution is well approximated by a standard Normal distribution and we can safely use the CI:

$$\left] \overline{y} - z_{\alpha/2}\frac{s}{\sqrt{n}} \ , \ \overline{y} + z_{\alpha/2}\frac{s}{\sqrt{n}} \right[. \tag{2.31}$$

But, as was seen in equation (2.27), with a large AR(1) sample, the true expected value of

S^2 is not σ^2, but $\frac{\sigma^2}{1-\lambda^2}$. Thus, an unbiased estimator of σ^2 is, for this model, $\hat{\sigma}^2 = (1-\lambda^2)\,S^2$. The corresponding unbiased estimator of $V[\overline{Y}] = \frac{\sigma^2}{n\,(1-\lambda)^2}$ is therefore

$$\widehat{V[\overline{Y}]} \;=\; \frac{(1-\lambda^2)\,S^2}{n\,(1-\lambda)^2} \;=\; \frac{1+\lambda}{1-\lambda}\,\frac{S^2}{n}\;. \tag{2.32}$$

Again assuming that Normality holds, the appropriate CI for μ would now be:

$$\left]\; \overline{y} - z_{\alpha/2}\,\sqrt{\frac{1+\lambda}{1-\lambda}}\,\frac{s}{\sqrt{n}}\;,\; \overline{y} + z_{\alpha/2}\,\sqrt{\frac{1+\lambda}{1-\lambda}}\,\frac{s}{\sqrt{n}}\;\right[\;. \tag{2.33}$$

This confidence interval is wider than the standard CI by a factor of $\sqrt{\frac{1+\lambda}{1-\lambda}}$.

The Normality assumption for $\frac{\overline{Y}-\mu}{\sqrt{\frac{1+\lambda}{1-\lambda}}\,\frac{S}{\sqrt{n}}}$ seems to hold well, for the simulation discussed above, as can be seen in the QQ-plot on the right of Figure 2.3. The fact that $E[S^2] > \sigma^2$ somewhat compensates the inappropriate effects of using the standard CI: equation (2.31) now gives the (nominally) 95% confidence interval $]9.932917, 10.46426[$. But this is still an inadequate CI. Out of the 10 000 samples in the simulation, only 58.77% of the "95% confidence" intervals given by formula (2.31) actually contain the true population mean $\mu = 10$. The more appropriate 95% confidence interval, given by formula (2.33), for the first sample is $]9.566163, 10.83101[$. It is more than twice as wide as the conventional one, since $\sqrt{\frac{1+\lambda}{1-\lambda}} = \sqrt{\frac{1.7}{0.3}} = 2.380476$. With these 95% (asymptotic) confidence intervals, the proportion of the 10 000 samples with intervals containing $\mu = 10$ rises to 94.80%.

The underlying lesson is that autocorrelation, when it exists, should be taken into account in any statistical analysis.

Chapter 3

Geographic Data Sets in R

In this chapter, we will look at two data sets that are going to be explored later in the book. The first data set represent *Aragonez* grape yields in Portugal. We read the original tabular data and create spatial georeferenced R objects of class `sf` to represent that data set (Section 3.1). We will produce a point spatial data set `Aragonez3763`, and explore the neighborhood relations between its elements (Section 3.2). We will also derive two new areal data sets `Aragonez3763Vor` and `Aragonez3763Grid`, where the spatial geometry if 2-dimensional. While `Aragonez3763Vor` (Section 3.3) covers the whole area of the vineyard, `AragonezGrid` (Section 3.5) represents the geographic area of influence of each group of plants as a regular grid, with void grid cells for missing data. Those data sets will be used later in Section 4.4. The second data set represent corn yields in an experiment near *Las Rosas*, in Argentina. In Section 3.7 explore the data and add new preditor variables that are derived from an ancillary digital elevation model. Later in the book (Section 4 and Section 5), those preditors will be used in spatial repression models for the yield. Throughout this chapter, we will discuss the main data structures in R that support geographic data.

In fact, R is an extremely powerful tool to access and analyze geographic data. If one is familiar with R, the major challenge is to understand the specificities of geographic data, and in particular the data structures that are necessary to represent those data sets. While for most standard statistical analysis, data can just be organized as a *table* (an object of class `data.frame` in R), this is in general insufficient for geographic data. In particular, data structures for geographic data must permit the representation of complex shapes in space. Moreover, geographic data sets have a specific location over the surface of the Earth, given by coordinates associated to the data which need to be interpreted in the appropriate coordinate reference system (CRS for short). Therefore, geographic data sets must always

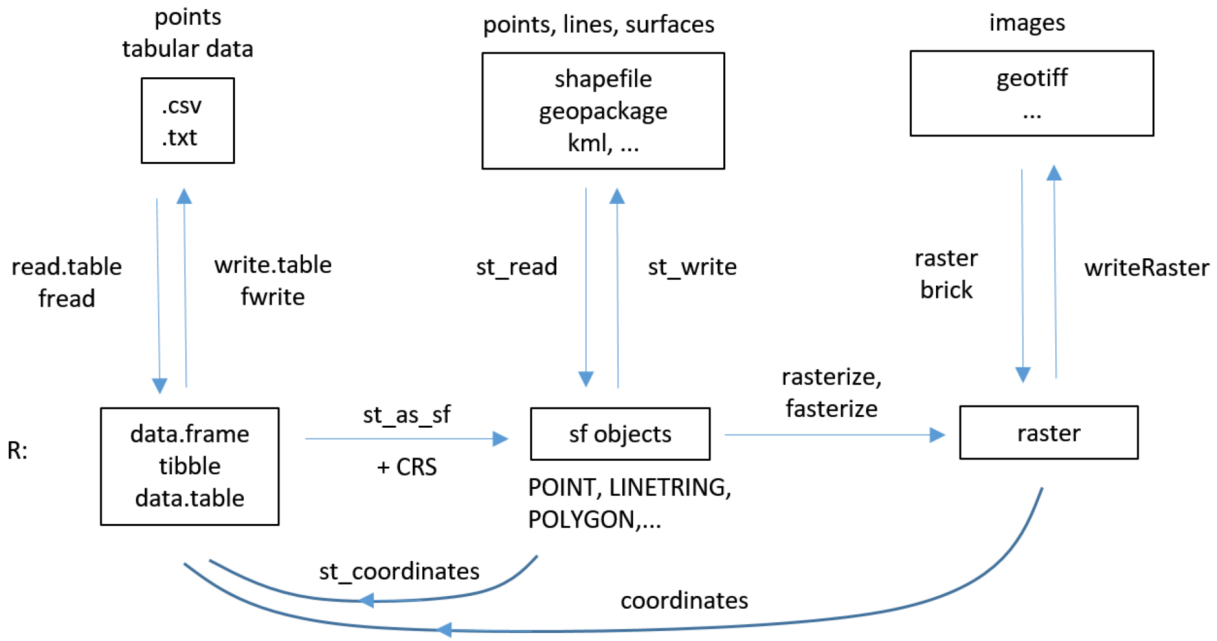


Figure 3.1: Read and write main data formats for spatial data in R

include coordinates and the corresponding CRS.

There are two basic families of data structures for geographic data: *vector* and *raster*. The vector data stucture represents spatial features which can be 0-dimensional (points), 1-dimensional (polygonal lines) or 2-dimensional (polygonal surfaces). Each feature can have several attributes which are represented in a attribute table (where each row corresponds to a feature, and each column corresponds to an attribute). For instance, this is a convenient data structure to represent territorial units within some region of the world, where each unit has a code and a name. The vector data structure is also convenient to represent a watershed system, where each geographic feature represents a stream and might have attributes like "stream name" and "water quality". In this case, each feature can be represented by a single polygonal line, or a group of polygonal lines. The raster format corresponds to *images*. It is a regular array of pixels, where each pixel represents some contiguous location over the surface of the Earth and has an associated numerical value. This is the natural data structure to represent, for instance, a satellite image of a given region, or a map of surface temperatures.

Besides vector and raster data structures, simple tables are often used to represent spatial information. Figure 3.1 describes the major functions in R to read and write files for each one of those data formats.

A very complete and easy to follow on-line resource is the eBook Geocomputation with R

which is also available as [3]. Package `sp` was the main package to process vector spatial data for a number of years, but it is being gradually abandoned in favour of the more recent package `sf` (for Simple Features for R) which is described in `https://r-spatial.github.io/sf/` or [4]. For *raster* data, package `raster` [5] is still widely used. One of the strong points of `raster` is that it allows to efficiently manage memory, which is crucial for large data sets as images typically are.

Since the R "Spatial Analysis" community is very dynamic, new packages are released frequently. Therefore, it is important to know which tools are available at any time. The site The Comprehensive R Archive Network's task view "Analysis of Spatial Data", by Roger Bivand is an excellent and very compact introduction that provides a general view of what is happening in this field. All R packages that are going to be used in this section are mentioned and put in context in that task view.

The major packages we rely on are `sf`, `raster`, and `mapview` to interactively visualize geographic data, but we will need a few other packages for more specific tasks.

```
library(raster)
library(sf)
library(mapview)
library(RANN) # fast nearest neighbors
library(interp) # linear spatial interpolation
library(spdep) #  spatial data analysis, includes knn2nb
library(tidyverse) # packages for data science, includes %>%
```

Some spatial analysis packages still require objects of the older vector class `sp`, so when necessary, `sf` objects are converted into `sp` objects using general purpose function `as` or `as_Spatial` from package `sf`. Spatial objects can be converted back to `sf` in an a similar way or with the `sf` dedicated function `st_as_sf()`

```
nc <- st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)
nc_sp <- as(nc, Class = "Spatial") # convert from sf to sp
nc_sp <- as_Spatial(nc) # convert from sf to sp
nc_sf <- st_as_sf(nc_sp, "sf") # convert from sp to sf
```

For users familiar with the `sp` package, it is worth to note that `sf` encompasses not only the capabilities of `sp`, but also of packages `rgeos` for spatial analysis and `rgdal` for reading/writing data in different file formats.

3.1 Vector data sets: a simple example with point data

Aragonez is the Portuguese name of a variety of grapes that is also called Tinta Roriz. This variety is most frequently known by its Spanish name, Tempranillo. A field trial to measure genotype yields was carried out under the supervision of Instituto Superior de Agronomia of the University of Lisbon, in Reguengos de Monsaraz, in the Évora district of Southern Portugal. A vineyard trellis was set up, with wires running on an approximate North-South direction, and which are henceforth referred to as columns.

In each column, groups of three plants were taken to represent a cell, thereby creating a rectangular grid with 40 columns and 26 rows (see illustration in Section 3.4). The rows are numbered 2 to 27 from North to South, since the bordering rows were considered a "transient" part, not included in the dataset. The 40 columns (numbered 4 to 43 from West to East since, again, bordering columns were left out of the data set) were 2.25 m apart. In each column, the centre of each grid cell (i.e, of the "rows") are separated by 3.75 m. Each grid cell is therefore a small rectangular region with three vines, whose yield produced a single observation for the data set, in kg of grapes per plant. There are fewer than $26 \times 40 = 1040$ observations since, for various reasons, some of the cells have missing values.

In Section 3.4 we will discuss how those measurements along rows and columns of the vineyard can be converted into longitudes and latitudes. For simplification, we consider at this point that latitudes and longitudes are available, as well as information about `genotype`, `block`, vineyard `column`, vineyard `row`, `yield` (kg/plant) and coordinates `colm` and `rowm` in meters, along columns and rows of the vineyard.

```
Aragonez<-read.table(file.path(getwd(),"datasets","Aragonez.txt"),header=TRUE)
head(Aragonez,3)
```

```
  genotype block col row colm  rowm yield       lon      lat
1    RZ717    B1   4   2    0 93.75 2.417 -7.516431 38.44193
2   RZ1158    B1   4   9    0 67.50 2.724 -7.516291 38.44172
3   RZ1325    B1   4   6    0 78.75 2.647 -7.516351 38.44181
```

To create a spatial object of class `sf` from a table we just need to use function `sf::st_as_sf` (most functions' names from package `sf` start with `st_` for "spatiotemporal") and indicate which columns of `Aragonez` should be used as coordinates, and the corresponding coordinate reference system. For this data set, coordinates are longitude and latitude over the global WGS84

CRS, with EPSG code 4326.[1]

```
AragonezSF<-st_as_sf(Aragonez, coords=c("lon","lat"),crs=4326)
```

R package `mapview` allows interactive visualizations of spatial data with or without background maps. The basic visualization function is called `mapview` and can be used to display `AragonezSF`.

```
mapviewOptions(basemaps="Esri.WorldImagery")
mapview(AragonezSF, cex=2,zcol="yield",lwd=0)
```

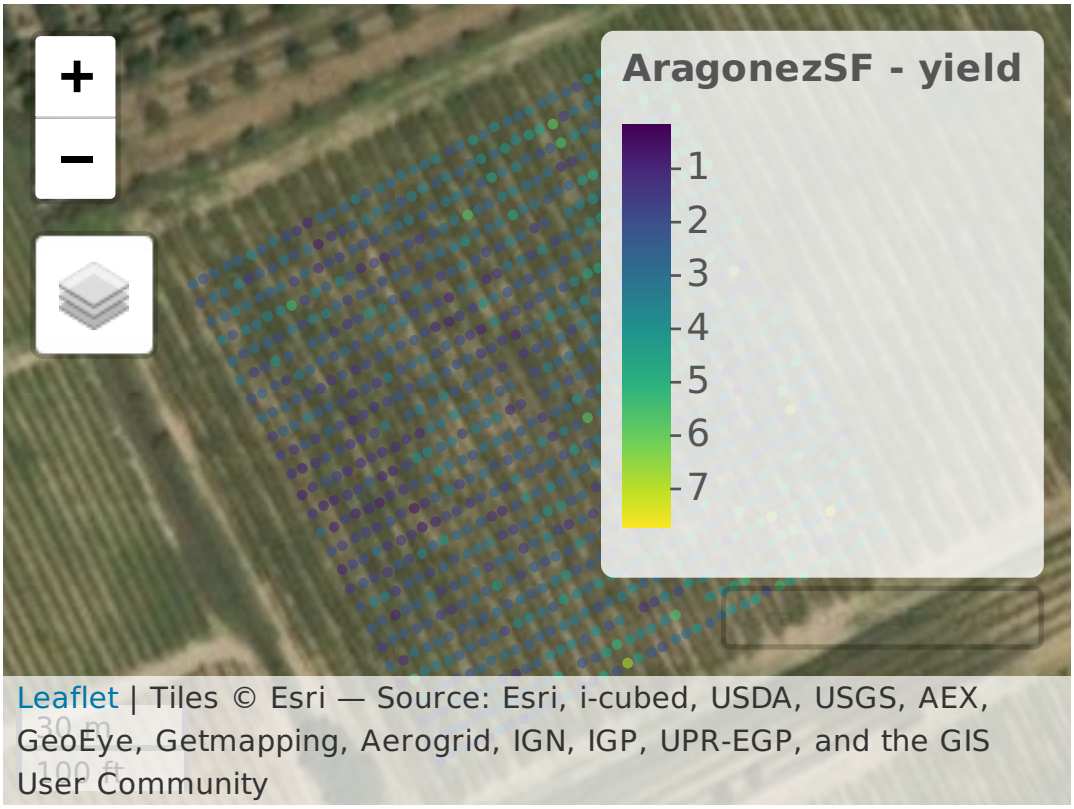


Figure 3.2: Mapview `sf` object. The option `basemaps` indicates the background map and `zcol` specifies the variable to be plotted.

[1] Throughout the book, `packageName::functionName` is going to be used to indicate functions which are not part of the base core of functions of R. Although syntax `packageName::functionName` is correct in R code, for simplicity, R commands within chunks of code will often indicate just the function, without the package name. In those cases, the context should make it clear which package is being used.

Note that one can edit data interactively over `mapview` maps (for instance, to create new features) with `mapedit::editMap`.

Since geographic coordinates (longitude and latitude) are not adequate to measure distances, angles and areas, it is recommended to work over a cartographic CRS suitable for the study area. For Continental Portugal, the official CRS is named ETRS89-TM-PT06 and has EPSG code 3763. To reproject `AragonezSF` we use function `sf::st_transform`.

```
Aragonez3763<-st_transform(AragonezSF,crs=3763)
```

Finally, `Aragonez3763` can be exported as *shapefile*, *geopackage*, *kml* or other vector format with `st_write`. Available drivers can be listed with `st_drivers()`.

```
st_write(Aragonez3763, "Aragonez3763.shp")
```

Additionally, it can be converted into an alternative R object. For instance, later in this text, packages like `spdep` accept as input spatial objects of the older class `sp`. As discussed earlier, we can create a `SpatialPointsDataFrame` object from a `sf` object with `sf::as_Spatial`.

```
AragonezPoints <- as_Spatial(Aragonez3763) # creates sp object
```

3.2 Determining neighbors in spatial point data sets

The concept of *neighborhood* is crucial to analyse spatial data and will be at the heart of the statistical techniques that are going to be discussed later in this book. Essentially, one wants to determine, for each data feature (e.g. a POINT in the Aragonez data set), which are its neighbors. In Section 4.4, several convenient functions from package `spdep` which determine neighbors are going to be discussed. Here, we will see how this can be done from scratch with `sp` functions and how the neighborhood can be represented as an R object compatible with the spatial data analysis functions of package `spdep`.

The main object to be defined is a matrix, where each row represents a feature (there will be 1019 rows for the Aragonez data set) and the k-th column holds the index (an integer) of a neighbor.

If the data set is not loo large, one can simply compute Euclidian or great circle distances for all pairs of features, and select for instance just the pairs of features with distances lower

than a given threshold. Function `sf::st_distance` returns an object of class `units`, since the units depend on the CRS, but this can be coerced to `numeric` by `as.numeric`.

The example below shows how to create a matrix `nn` of neighbor indices starting with all distances between pairs of features. In this example, matrix `nn` is dense and it is built according to the following rule: if j-th feature is a neighbor of the i-th feature, then `nn[i,j]` is `j`, otherwise `nn[i,j]` is `NA`.

```
D <- st_distance(Aragonez3763) # matrix N*N
nn<-col(D)  #  assume that all features are neighbors
nn[as.numeric(D)>4]<-NA # NA is assigned to non neighbors
head(nn[,1:10],3)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1   NA   NA   NA   NA    6   NA   NA   NA    NA
[2,]   NA    2   NA    4   NA   NA   NA    8   NA    NA
[3,]   NA   NA    3   NA   NA   NA   NA   NA   NA    10
```

Matrix `nn` above indicates that feature 1 is a neighbor of feature 1 and feature 6, feature 2 is a neighbor of features 2, 4 and 8, and so on. We can now build an R object compatible with the spatial data analysis functions of package `spdep` to be discussed in Section 4.4. It has to be of class `knn` and it has to be a list with the following components: the matrix `nn`, the number of features `np`, the number `k` of columns of `nn`, `dimension=2` since there are two coordinates, and the matrix of coordinates `x`.

```
xy<-st_coordinates(Aragonez3763)
NN<-structure(list(nn=nn, np=nrow(nn), k=ncol(nn), dimension=2, x=xy), class="knn")
```

Finally, the topology of the neighborhood can be checked visually with the following command.

```
plot(spdep::knn2nb(NN),coord=NN$x)
```

Alternatively to `st_distance`, one can use `sf::st_is_within_distance` which returns a logical matrix, and proceed similarly to define `nn`.

```
D <- st_is_within_distance(Aragonez3763,dist=4,sparse=FALSE) # logical N*N
nn<-col(D)#  assume that all features are neighbors
nn[!D]<-NA # NA is assigned to non neighbors
```

For large data sets the construction above is not convenient since it requires a very large matrix `D`. Alternatively, one should use function `RANN::nn2` to create a more compact matrix `nn`. Similarly, the output of `RANN::nn2` can be converted into an object of class `knn`, which can be used as an input for package `spdep`. In the example below, we restrict the number of neighbors to `k=30` and the distance between neighbors to `radius=5`. Since `nn2` returns 0 when the k-th neighbor is not defined, we need to replace 0 by `NA` as required by `spdep`.

```
xy <- st_coordinates(Aragonez3763)
nn<-nn2( xy , k=30, searchtype="radius",radius=4)$nn.idx
nn[nn==0]<-NA
head(nn[,1:10],3)
```

Several simple functions from package `spdep`, which will be discussed later in Section 4.4, can be used to easily generate the neighborhoods discussed above. However, one my want to define a neighborhood specifically designed for the data at hand. For instance, let's suppose that only groups of plants which either belong to the same vineyard column and are at most 4 meters apart, or belong to distinct vineyard columns but are at most 3 meters apart, should be neighbors. We can adapt the techniques described above and design a matrix `nn` according to that rule.

If we rely on function `RANN::nn2`, then we explore the fact that the output of `nn2` is a list with two components: `$nn.idx` is the matriz of indices of neighbors (as seen earlier), and `$nn.dists` is the corresponding matrix of distances. Firstly, we apply `RANN::nn2` to obtain those two matrices (we consider up to 30 neighbors, which is more than sufficient given the spatial distribution of our data set).

```
nn<-nn2( xy , k=30)$nn.idx  # matrix 1019*30 of indices
d<-nn2( xy , k=30)$nn.dists # matrix 1019*30 of distances
```

Then, we extract the attribute of interest: in our case it is the vineyard column number for each feature we call `idxcol`. With this, we can select which neighbors do not fulfill the condition set above (i.e. at most 4 meters within the same column and at most 3 meters for

distinct columns), and we can assign NA to those neighbors. Note that nn[idxcol] is the vineyard column index of every neighbor in nn, so idxcol[nn]==idxcol is TRUE when both the feature and its neighbor belong to the same vineyard column and FALSE otherwise.

```
idxcol<-Aragonez3763$col
nn[(idxcol[nn]==idxcol & d >4) | (idxcol[nn]!=idxcol & d>3)]<-NA
head(nn[,1:10],3)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1   28    6   NA   NA   NA   NA   NA   NA    NA
[2,]    2   29    4    8   NA   NA   NA   NA   NA    NA
[3,]    3   34   10   11   NA   NA   NA   NA   NA    NA
```

The resulting matrix indicates that the neighbors of feature 1 are features 1, 28 and 6, the neighbors of feature 2 are features 2, 29, 4 and 8, and so on. Finally, the resulting neighborhood can be converted into an spdep object and plotted with spplot.

```
NN<-structure(list(nn=nn, np=nrow(nn), k=ncol(nn), dimension=2, x=xy), class="knn")
plot(spdep::knn2nb(NN),coord=NN$x)
```

3.3 Vector data sets: multiple geometries

A sf object has a table structure. The attribute table of the data is extended with a special column called *geometry*. The remaining columns of the table are feature's attributes, and each row of the table describes one feature. The main simple feature geometry types are POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON and GEOMETRYCOLLECTION as the mixed type. The geometry of features in a sp object can be extracted with function st_geometry, which returns a list of the geometries of all features (the list is an object of class sfc, for simple feature geometry list-column).

In general, the input for spatial data analysis can be either points or areas (areal data), where common boundaries between features determine the neighbors. In Section 3.1, the simplest geometry (POINT, where each feature corresponds to a single point) has been illustrated with the Aragonez3763 example. Now, we will represent the data set as a collection of polygons.

Firstly, from `Aragonez3763` we derive an object with POLYGON geometry, which is going to be the convex hull of the set of points. The first step is to group all points in one single feature, which can be done with an unary union performed by `sf::st_union` with a single input, which returns in this case a MULTIPOINT geometry type. Notice that this operation eliminates all attributes of `Aragonez3763` since the attribute table collapses to one single row with no other columns besides `geometry`. In fact, `AragonezMP` just holds the geometry and it is of class `sfc` (simple feature geometry list-column).

```
AragonezMP<-st_union(Aragonez3763) # MULTIPOINT sfc object (with 1019  points)
class(AragonezMP)


[1] "sfc_MULTIPOINT" "sfc"
```

Next, we create the convex hull of the single feature, which has POLYGON geometry, and the buffer of the convex hull with `st_buffer`, which is of POLYGON geometry as well, considering a buffer distance of 3 meters.

```
AragonezCHull<-st_convex_hull(AragonezMP) # class sfc with one POLYGON
AragonezBuffer<- st_buffer(AragonezCHull, dist=3) # class sfc with one POLYGON
```

To complete this section, we create an areal version of the Aragonez data set by associating to each group of plants its *Voronoi polygon*. Function `sf::st_voronoi` is used to create the Voronoi polygons for the 1019 points in `AragonezMP`. Finally, since `sf::st_voronoi` outputs an object of geometric type GEOMETRYCOLLECTION, function `st_cast()` is needed to simplify the geometry to POLYGON.

To make the code easier to read, the next instruction uses `magrittr`'s pipe operator (package `magrittr` is part of the set of pakages called `tidyverse` which was loaded at the beginning of this section). This is a convenient way of applying several functions consecutively: the function after `%>%` uses as its first argument the object before `%>%` (i.e. x `%>%` f(y) is the same as f(x,y)).

```
AragonezVoronoi<-AragonezMP %>%
  st_voronoi() %>% # create Voronoi polygons
  st_cast()        # cast to  POLYGON
```

Voronoi polygons returned by `sf::st_voronoi` covers all the extent of the data set (which could be identified with `sf::st_bbox`). If we want to limit the extent of boundary polygons, we can use `AragonezBuffer` to clip `AragonezVoronoi` and reduce the size of the boundary Voronoi polygons.

```
AragonezVoronoiBuffer<-st_intersection(AragonezVoronoi,AragonezBuffer)
```

We can now associate the POLYGON geometry of `AragonezVoronoiBuffer` to the original `sf` object `Aragonez3763`: we just have to replace the geometry column of Aragonez3763 (POINT geometry) by the new POLYGON geometry (Voronoi polygons) since the features are the same and the order of the features has not changed.

```
Aragonez3763Vor<-Aragonez3763 # copy sf
st_geometry(Aragonez3763Vor)<-AragonezVoronoiBuffer # replace geometry
```

Finally, we map the output in Figure 3.3, using `zcol` to indicate the variable to be displayed in the legend.

Later, in Section 3.5, we will consider the case where, instead of associating to each location a Voronoi polygon, one will associate to the groups of plants a regular grid, with missing grid cells for missing values.

To conclude this section, and analogously to Section 3.2, we briefly discuss how we can define neighbors from the areal data set `Aragonez3763Vor` according to an arbitrary user defined rule. Let's for instance consider that two Voronoi polygons are neighbors if the shortest distance between them is lower than some small tolerance `tol`. Then, we need to test if the distance between POLYGON features in `Aragonez3763Vor` is lower than `tol` and proceed as in Section 3.2.

```
tol<-1
D <- st_is_within_distance(Aragonez3763Vor,dist=tol,sparse=FALSE)# matrix N*N
nn<-col(D)  #  assume that all features are neighbors
nn[!D]<-NA  # non neighbors are NA
```

As in Section 3.2, we first create the appropriate `knn` object, and then we can plot both `Aragonez3763Vor` and the links between neighbors as described below to visualy check the result.

```
mapviewOptions(basemaps="Esri.WorldImagery")
mapview(Aragonez3763Vor,zcol="yield",alpha.regions=0.7, lwd=0.3)
```

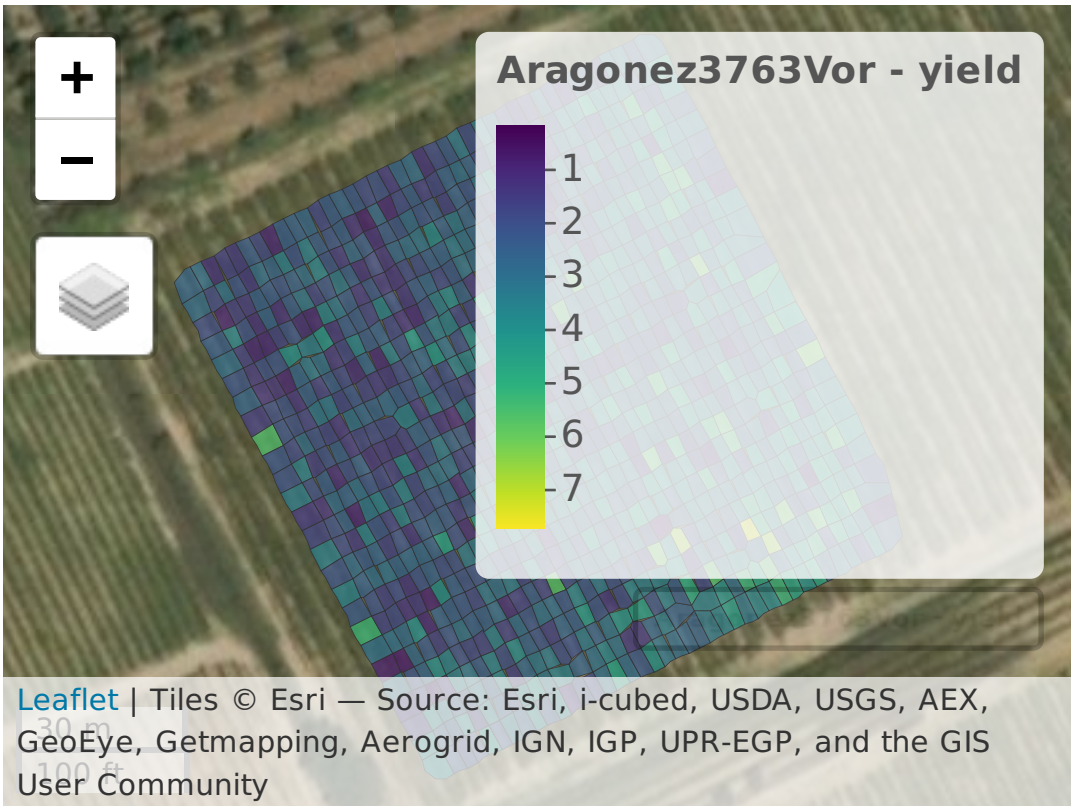


Figure 3.3: Mapview `Aragonez3763Vor`. The option indicates the background map and `alpha.regions` determines the opacity of the polygons in the map.

```
xy<-Aragonez3763Vor %>% st_centroid() %>% st_coordinates()
NNVor<-structure(list(nn=nn,np=nrow(nn),k=ncol(nn),dimension=2,x=xy),class="knn")
plot(st_geometry(Aragonez3763Vor))
plot(spdep::knn2nb(NNVor),coord=NNVor$x, add=TRUE, col="red", cex=0.5)
```

3.4 Working example: georeferencing the Aragonez data set

The Aragonez data set has been described in Section 3.1. As discussed then, the location of plants of the Aragonez data set is determined by a vineyard trellis, with wires running on an approximate North-South direction, and which are referred to as columns. In each

column, groups of three plants were taken to represent a cell, thereby creating a rectangular grid with 40 columns and 26 rows.

The vineyard's columns are approximately parallel. Figure 3.4 shows the location of each group of plants along the rows and columns of the vineyard. Moreover, it gives the longitude and latitude for three known locations in the standard WGS84 CRS. Note that longitudes and latitudes for those specific three locations could have been obtained with high precision GPS device or could have been simply extracted from georeferenced high resolution imagery over the vineyard.

The original data are the yields for each row and column, as well as other attributes that will not be considered in this exercise, and can be read from the text file "Aragonez.txt".

```
Aragonez<-read.table(file.path(getwd(),"datasets","Aragonez.txt"),header=TRUE)
Aragonez<-Aragonez[,c('genotype','block','col','row','yield')]
head(Aragonez,3)
```

```
  genotype block col row yield
1    RZ717    B1   4   2 2.417
2   RZ1158    B1   4   9 2.724
3   RZ1325    B1   4   6 2.647
```

The goal of this exercise is to georeference this data set. This is a necessary condition for combining this data set with additional geographic information and, for example, display it with `mapview` with some background image. The goal is to associate to each group of three plants their geographic coordinates. Towards this end, we consider the three locations in Figure 3.4 which have a row and column index (`row` and `col`) as well as longitude and latitude coordinates. If we suppose that a simple linear transformation is enough to transform the data, which is acceptable due to the small size and regular geometry of the plot, the system of equations that needs to be solved in order to $a_1, \ldots, c_2$ is

$$\begin{cases} x &= a_1 + b_1\,\texttt{row} + c_1\,\texttt{col} \\ y &= a_2 + b_2\,\texttt{row} + c_2\,\texttt{col}. \end{cases}$$

In matrix form, we want to solve the equation $B = AT$, with respect to T, *i.e.* $T = A^{-1}B$,

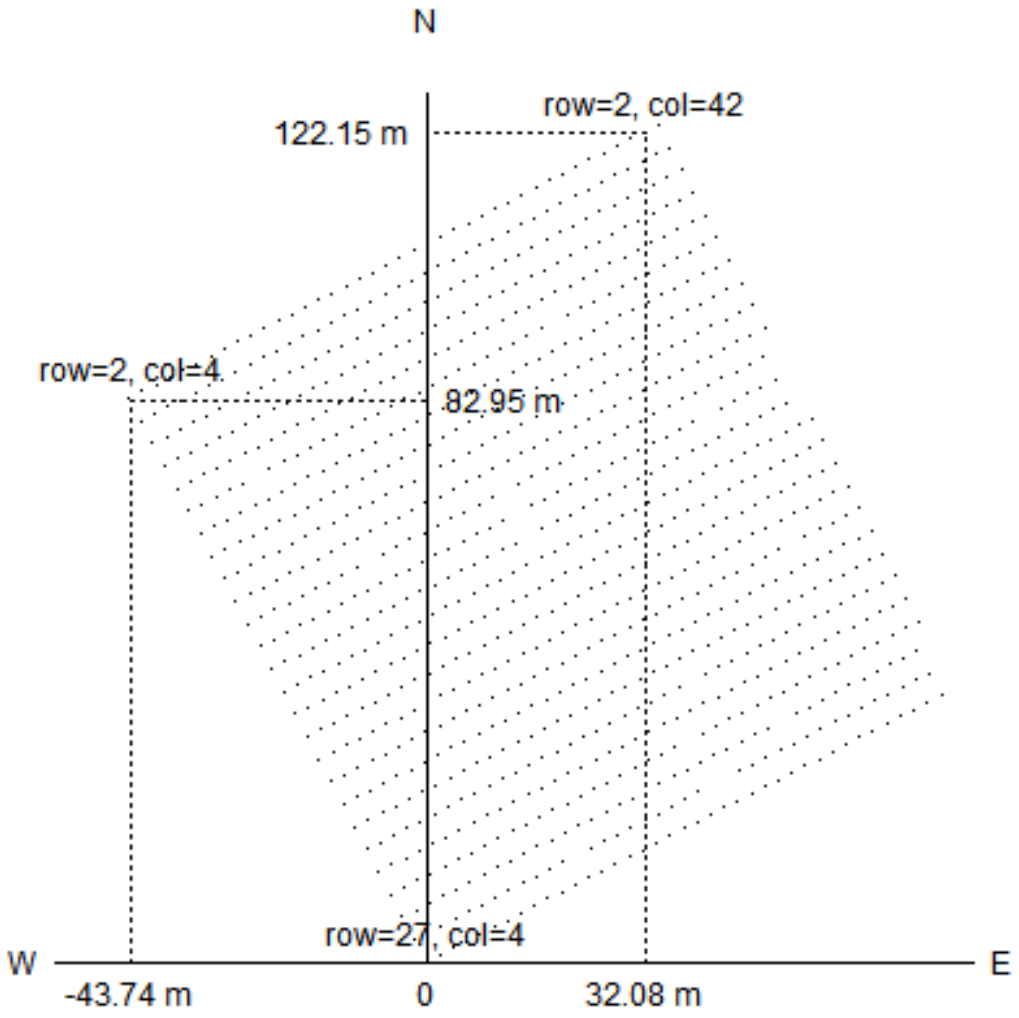


Figure 3.4: Plot of the Aragonez data set with the indication of the rows and columns of the vineyard and the WGS84 coordinates at three points. With a linear transformation, those three points are used to georeference the whole data set.

where

$$B = \begin{bmatrix} -7.515930 & 38.44118 \\ -7.516431 & 38.44193 \\ -7.515540 & 38.44229 \end{bmatrix} \quad A = \begin{bmatrix} 1 & 27 & 4 \\ 1 & 2 & 4 \\ 1 & 2 & 42 \end{bmatrix} \quad \text{and } T = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{bmatrix}$$

```
B<- rbind( c(-7.515930, 38.44118),c(-7.516431, 38.44193), c(-7.515540, 38.44229))
A<-rbind(c(1,27,4),c(1,2,4),c(1,2,42))
T<-solve(A,B)
```

Now that we know T, we can just apply the transformation T to the row and column numbers of the 1019 locations. This can done by multiplying a 1019×3 matrix – with a first column of 1's, followed by the row and column numbers –, by T to obtain a 1019×2 matrix which columns are x and y. Since matrix multiplication is done in R with the operator `%*%`, the following command returns the longitudes and latitudes for all 1019 groups of plants.

```
lonlat<-cbind(1,Aragonez[,"row"],Aragonez[,"col"])%*%T
head(lonlat,3)

          [,1]     [,2]
[1,] -7.516431 38.44193
[2,] -7.516291 38.44172
[3,] -7.516351 38.44181
```

Finally, we just need to add the longitudes and latitudes to `Aragonez` and convert it a `sf` object with `st_as_sf` as seen earlier. Since coordinates are longitude and latitude, the EPSG code is 4326.

```
Aragonez$lon<-lonlat[,1]
Aragonez$lat<-lonlat[,2]
AragonezSF<-st_as_sf(Aragonez, coords=c("lon","lat"),crs=4326)
```

The result is the georefenced data set which is depicted in Figure 3.2.

3.5 Working example: creating a regular grid for the Aragonez data set

The goal of this exercise is to create a grid for the Aragonez data set oriented along the vineyard trellis which aproximates the area of influence of each group of three plants.

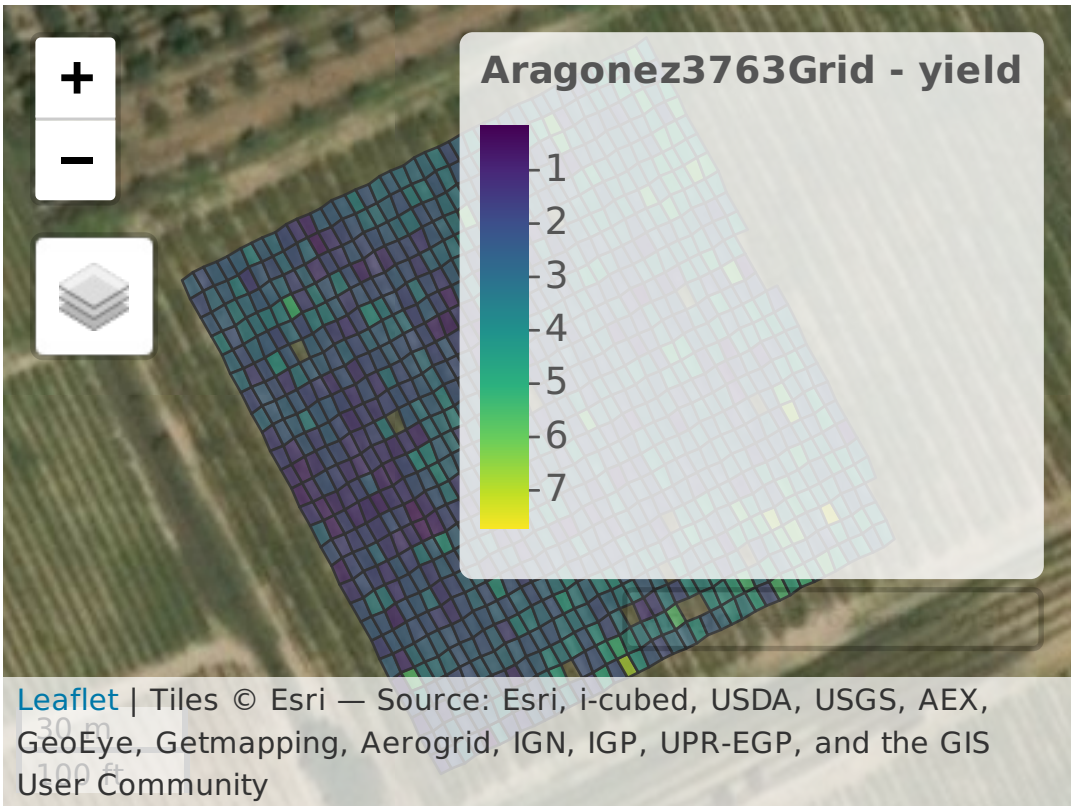


Figure 3.5: Regular grid cells for the Aragonez data set, with void cells for missing data.

```
Aragonez<-read.table(file.path(getwd(),"datasets","Aragonez.txt"),header=TRUE)
AragonezSF<-st_as_sf(Aragonez, coords=c("lon","lat"),crs=4326)
```

One possible solution for this problem uses the linear transformation T above thats convert points in the referential (row,column) into geographic coordinates. In particular, for each vineyard `row` and `col`, we can consider that the respective grid cell ranges from `row`-0.5 to `row`+0.5 and from `col`-0.5 to `col`+0.5. For instance, to the group of plants at location `row`=2 and `col`=4 (see Figure 3.4), corresponds to the following areal element, defined by its four pairs of coordinates (the first pair is repeated at the end).

```
cbind(c(1,1,1,1,1), c(1.5,2.5,2.5,1.5,1.5), c(3.5,3.5,4.5,4.5,3.5)) %*% T
```

```
          [,1]     [,2]
[1,] -7.516453 38.44194
[2,] -7.516433 38.44191
[3,] -7.516409 38.44192
[4,] -7.516429 38.44195
[5,] -7.516453 38.44194
```

To make the code below more compact, let us define a function `Tr` which returns a POLYGON geometry (with longitude and latitude coordinates) from the *row* and *column* position, using linear transformation `T`.

```
Tr<-function(df) #input is a data.frame
{
  row<-df$row;col<-df$col
  A <- cbind(c(1,1,1,1,1),
             c(row-.5,row+.5,row+.5,row-.5,row-.5),
             c(col-.5,col-.5,col+.5,col+.5,col-.5))
  return(st_polygon(list(A%*%T)))
}
```

Let's test this function over the group of plants in position (2,4). The output is a `sf` object with POLYGON geometry as expected.

```
Tr(data.frame(row=2,col=4))
```

```
POLYGON ((-7.516453 38.44194, -7.516433 38.44191, -7.516409 38.44192, -7.516429
38.44195, -7.516453 38.44194))
```

To obtain the full geometry, i.e. an object `sfc` which is a list of POLYGON, we just have to apply `Tr` to each of the 1019 rows of `Aragonez[,c("row","col")]`, and gather the results as a list. This could be achieved with a `for` cycle but it can also be done with R base function `by` applied to the `data.frame` with *row* and *col* numbers. The option `simplify=FALSE` ensures that `by` does not attempt to collapse the output list into a more compact R object.

```
L<-by(Aragonez[,c("row","col")], # data.frame
      INDICES=1:nrow(Aragonez),  # each group is a data.frame row
      FUN=Tr,                    # function to be applied to each group
      simplify=FALSE)
```

Function `by` technically returns an object of class `by`, but it can be cast to `list` with the generic R base function `as`. Since `as(L,"list")` is a list of POLYGON, it can be used to create our `sfc` object, with one POLYGON for each group of plants.

```
new.geometry<-st_as_sfc(x=as(L,"list"),crs=4326)
```

Finally, we can proceed as in Section 3.3 and make a copy of `AragonezSF`, where the POINT geometry column is replaced by the new POLYGON geometry that was created above, to obtain the desired spatial data set called `AragonezGrid`.

```
AragonezGrid<-AragonezSF
st_geometry(AragonezGrid)<-new.geometry
Aragonez3763Grid<-st_transform(AragonezGrid, crs=3763)
```

3.6 Raster geographic data sets

The `raster` package can deal with very large images since it does not need to load the whole data set in memory. If one needs to read a very large file, `raster` can be used to create the connection, and then data can be loaded by blocks of rows with `raster::getValues` and processed one block at the time.

In the example below the data set is a multilayer image with three bands (three bands of a Landsat 8 surface reflectance images over the Alentejo) and it is read with `raster::brick`. If each band was available as a separate `tif` file, `raster::stack` could be used to read the list of file names. Then, the image stack could be converted into a multiband single image with `raster::brick`. For `tif` files (or other image formats) with just one layer, reading the file can also be done with `raster::raster`. All those functions read the file metadata but do not load the actual data into memory, which is very convenient for large data sets.

```
b<-brick(file.path(getwd(),"datasets","LC82030332014151LGN00_sr_bands345.tif"))
raster::inMemory(b)

[1] FALSE

names(b)<-c("band3","band4","band5")
print(b)

class       : RasterBrick
dimensions  : 7791, 7651, 59608941, 3  (nrow, ncol, ncell, nlayers)
resolution  : 30, 30  (x, y)
extent      : 529785, 759315, 4190085, 4423815  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=29 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84
data source : /home/jcadima/Isa/Geo/OpenSpat2/datasets/LC82030332014151LGN00_sr_bands
names       : band3, band4, band5
min values  :   -20,   -88,  -205
max values  : 12661, 13515, 12429
```

For multiband images, `mapview::mapview` might be replaced by `mapview::viewRGB` to display color composites of the multiple bands. The color composite defined in the code below is a *false color* composite, since in particular the red channel (denoted by "r") of the color composite corresponds to the near infrared band (band 5) of the sensor.

In Figure 3.6, the extent of the image is computed with `raster::extent`. Then, a new narrower extent is computed with `ext/4` and the original image is cropped with `raster::crop`.

Landsat 8 surface reflectance values are not supposed to be smaller than 0 or larger than 10000, so we assign `NA` to those pixels, before computing the vegetation index `ndvi`, which is displayed with `mapview::mapview`, using colors and value intervals set by the user. Map colors are briefly discussed in Appendix C.

```
myb[myb<=0 | myb>10000]<-NA # set non valid values
ndvi<-(myb[[3]]-myb[[2]])/(myb[[3]]+myb[[2]]) # access individual bands
```

The CRS of the data set is available through function `raster::crs`. It is an object of class `CRS` and can be converted into a simple string that describes the CRS as discussed in Appendix B.

```
ext<-extent(b) # raster extent
myb<-crop(b,ext/4) # crop to a smaller extent
mapviewOptions(basemaps="CartoDB.Positron")
mapview::viewRGB(myb,r = 3, g = 2, b = 1)
```

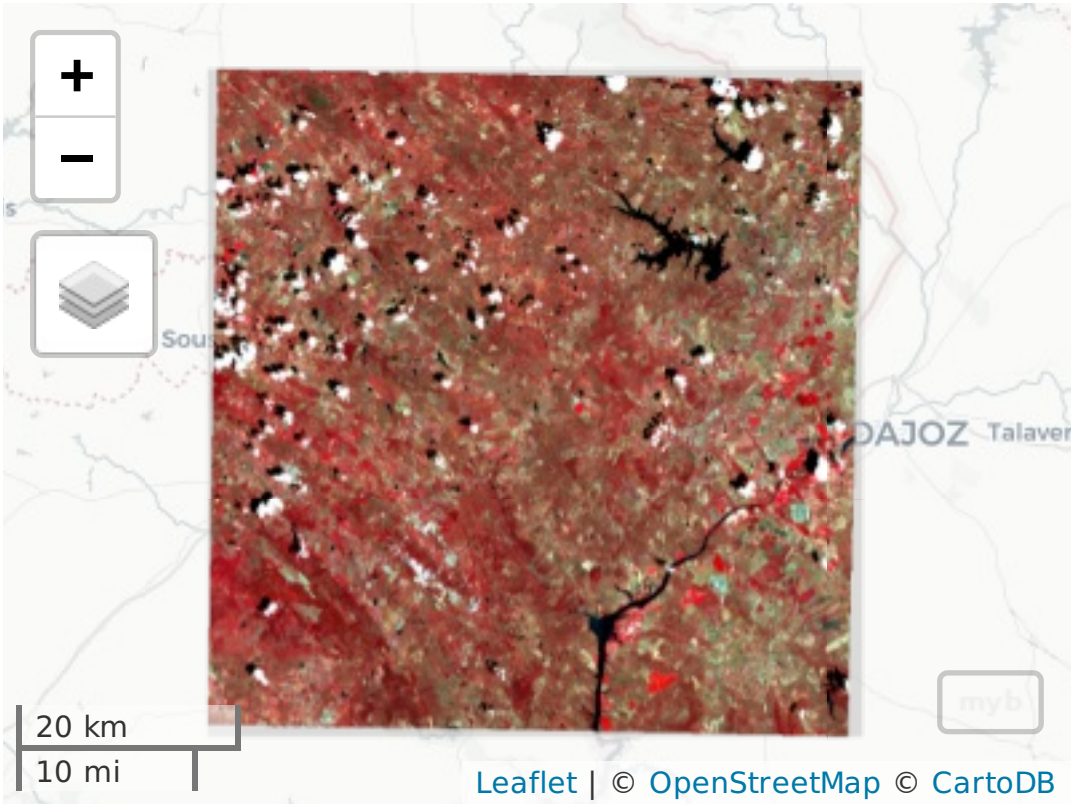


Figure 3.6: Mapview color composite of multiband images.

```
crs(b)

CRS arguments:
 +proj=utm +zone=29 +datum=WGS84 +units=m +no_defs +ellps=WGS84
+towgs84=0,0,0

print(as.character(crs(b)))

[1] "+proj=utm +zone=29 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

```
mycolors<-colorRampPalette(c(rgb(1,1,0,0.5), rgb(0,1,0,0.5)), alpha = TRUE)(10)
mapview(ndvi,col.regions=mycolors,at=c(0.1,.3,.5,.7,1))
```

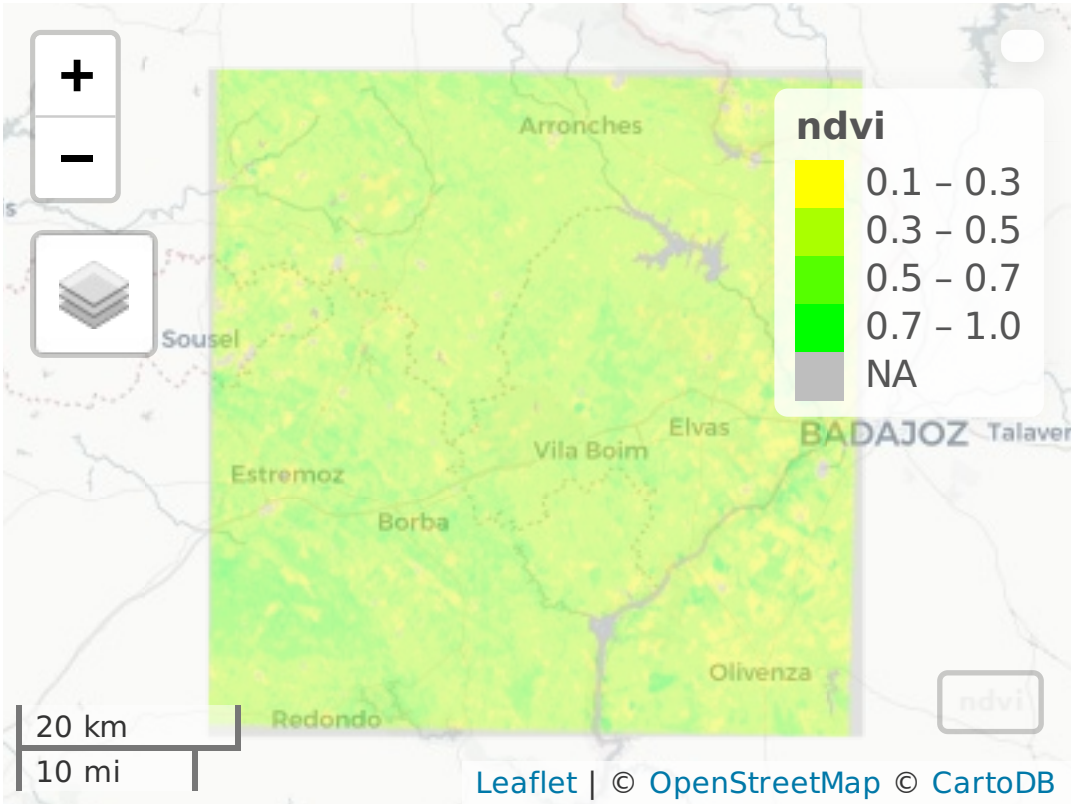


Figure 3.7: The `ndvi` map, with explicit definition of the legend intervals and colors.

This PROJ string describes the CRS. The map projection for `b` is universal transverse Mercator at zone 29, and the reference datum is WGS84. The coordinate units are meters. Finally, `+ellps` indicates the ellipsoid, and `+towgs84`, which is unnecessary for this particular CRS, contains in general parameters for datum transformation. As discussed in Appendix B, this CRS can also be referred to by its `epsg` code. In fact, the vast majority of usual CRS have an EPSG code, which can be found in spatialreference.org.

```
utm.29<-"+init=epsg:32629" # string that can be interpreted as a CRS
```

Pixel values can be extracted (and loaded into memory) with function `raster::values` and coordinates can be extracted with `raster::coordinates`. Function `raster::rasterToPoints` extract both and removes pixels with `NA` values. All those functions return vectors or matrices.

```
head(values(ndvi),3)

[1] 0.3966480 0.4031805 0.4117297

head(values(myb),3)

     band3 band4 band5
[1,]   709   972  2250
[2,]   708   957  2250
[3,]   732   988  2371

head(coordinates(myb),3)

          x       y
[1,] 615870 4336140
[2,] 615900 4336140
[3,] 615930 4336140

head(rasterToPoints(myb),3)

          x       y band3 band4 band5
[1,] 615870 4336140   709   972  2250
[2,] 615900 4336140   708   957  2250
[3,] 615930 4336140   732   988  2371
```

Let's consider the problem of determining the location with the highest NDVI value. To address the question, it is more convenient to work with coordinates of the pixel centers and with pixel values instead of using the full raster data structure. Since `rasterToPoints` returns a three-column matrix with coordinates and values for all pixels, it is very easy to determine the row of that matrix which has the highest NDVI value. For instance, one can use function `which.max` over the `ndvi` values (*i.e.* the third column of `xyz` below) that gives us the position of the maximum. Then , we just have to select that row from `xyz`, which gives us the coordinates x, y of the pixel where that maximum occurs.

```
xyz<-rasterToPoints(ndvi)
xyz[which.max(xyz[,3]),] # x,y and maximun ndvi value


           x            y        layer
6.702000e+05 4.305450e+06 9.458685e-01
```

3.7 Working example: Las Rosas

In this section, we will read data from a corn field experiment in Las Rosas, Argentina. We will also read topographic data for the same region and combine the experiment data with relief data derived from the topography. The ultimate goal is to model the yield at each location from the set of predictors. The statistical models and methods to address the problem will be discussed and applied in Section 4 and Section 5. Here, we are going to read the initial data set and expand it with relief data, to create the full spatial R object needed for the statistical analysis.

3.7.1 Read experiment data and gather elevation data

The *Las Rosas* data set [6] contains measurements of corn yield over a controlled plot in Argentina. Measurements are made over an almost regular grid and are approximately 71 cm apart. Besides yield, *Las Rosas* data set also includes the amount of nitrogen fertilizer that is applied in each location. For the experiment described in [6], 6 different levels of nitrogen fertizer (0 , 39 kg/ha, 50.6 kg/ha , 75.4 kg/ha, 99.8 kg/ha and 124.6 kg/ha) were applied along the rows of the field. The basic set of information consists of four variables measured at 1704 locations:

1. `YIELD`, which is the yield of corn, has been converted to kg/ha;
2. `N`, which is the amount of nitrogen fertilizer, also expressed in kg/ha;
3. `LONGITUDE`, the location longitude in degrees
4. `LATITUDE`, the location latitude in degrees

The data are available in the the following file:

```
X<-read.table(file.path(getwd(),"datasets","rosas2001predN-kg-ha.txt"),header=TRUE)
head(X,3)

     YIELD     N LONGITUDE  LATITUDE
1 4224.759 124.6 -63.84857 -33.04995
2 4308.220 124.6 -63.84850 -33.04998
3 4300.509 124.6 -63.84843 -33.05000

dim(X)

[1] 1704    4
```

An interesting fact is that, overall, variables `YIELD` and `N` have very low correlation.

```
cor(X$YIELD,X$N)

[1] 0.07880061
```

To be able to examine the geographic context the observations, we convert `data.frame X` into a `sf` POINT geometry object. The coordinate reference system (WGS84) is indicated as EPSG code as discussed in Appendix B.

```
X4326<-st_as_sf(X, coords=c("LONGITUDE","LATITUDE"),crs=4326)
head(X4326,3)

Simple feature collection with 3 features and 2 fields
geometry type:  POINT
dimension:      XY
bbox:           xmin: -63.84857 ymin: -33.05 xmax: -63.84843 ymax: -33.04995
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
     YIELD     N                    geometry
1 4224.759 124.6 POINT (-63.84857 -33.04995)
2 4308.220 124.6  POINT (-63.8485 -33.04998)
3 4300.509 124.6    POINT (-63.84843 -33.05)
```

The data set can now be plotted with `mapview::mapview` with high resolution imagery background.

```
mapviewOptions(basemaps="Esri.WorldImagery")
mapview(X4326,zcol="YIELD",legend=TRUE, cex=1.5, lwd=0)
```

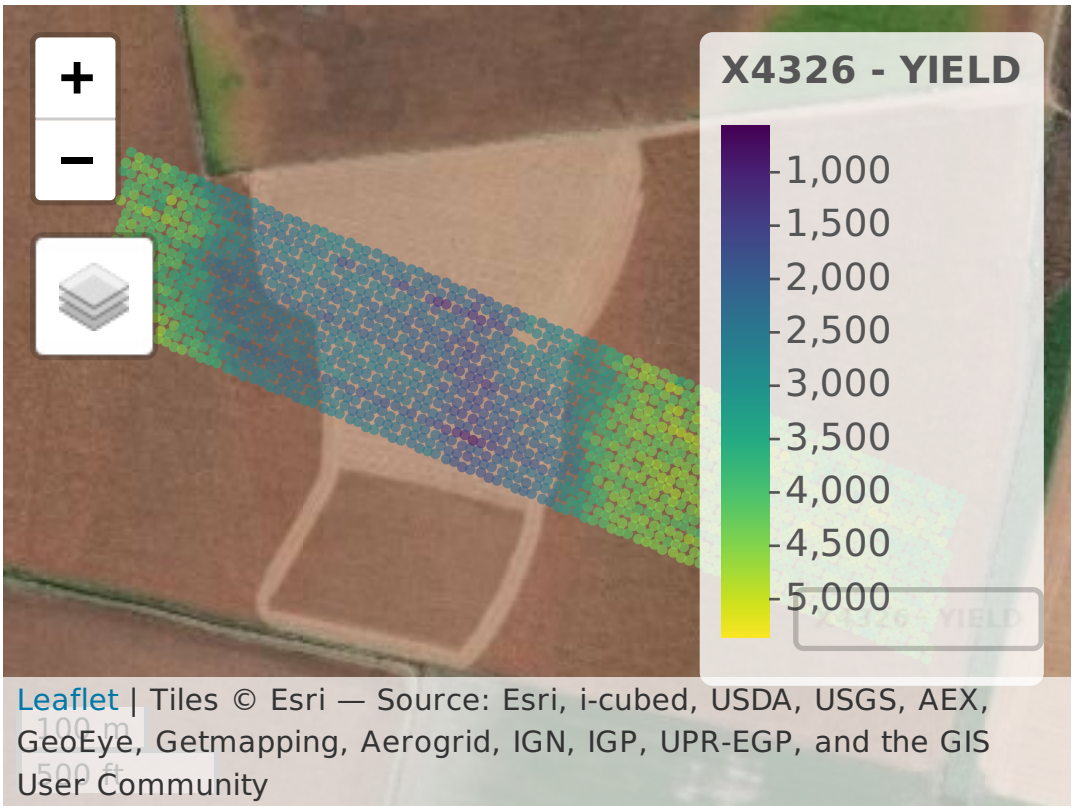


Figure 3.8: The Las Rosas data set in Argentina.

This data set is located in Argentina, around coordinates LONG= -63.84521 and LAT=-33.05053. The examination of the plot suggests that elevation varies within its boundaries. Furthermore, it is clear that yield is somewhat correlated with elevation. Therefore, we should estimate the elevation for each observation, and possibly derive new useful variables that describe the relief, in addition to `N`, to model corn yield.

Elevation `SRTM` data for the $1^o \times 1^o$ tile where the data set lies can be downloaded, unzipped and imported through the following R commands. Note that the `if` condition tests if the file `S34W064.hgt` already exists in the working directory.

```
if (!("S34W064.hgt" %in% list.files(path=file.path("datasets")))) {
urlzip<-"http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/South_America/S34W064.hgt.zip"
download.file(url=urlzip,destfile=file.path("datasets","S34W064.hgt.zip"),mode="wb")
unzip(zipfile=file.path("datasets","S34W064.hgt.zip")) }
srtm<-raster(file.path("datasets","S34W064.hgt"))
```

The resulting object is of class `RasterLayer` and contains the elevation measurements for each 3 arc-second pixel (the resolution is therefore approximately 90 m in the North-South direction). Note that `srtm` coordinates are longitude and latitude (CRS EPSG:4326).

Alternatively, one could use the SRTM1 data set (with a finer 1 arc-second spatial resolution) that can be downloaded from Earth Explorer.

```
srtm1<-raster(file.path("datasets","s34_w064_1arc_v3.tif"))
```

3.7.2 Deriving variables that describe the relief from elevation data

The goal of the current section is to derive new relevant variables from elevation for the 1704 locations in the data set `X`.

Since `srtm` contains a full $1^o \times 1^o$, much larger than the actual plot of interest, let us first crop the image to the extent of the *Las Rosas* data set. We use the `%>%` pipe operator to make code simpler to understand. The overall goal is to determine a buffer (say, of 500 m) around the points. To that end, we need to reproject the data to a local cartographic coordinate reference system with coordinates in meters (here we use the UTM CRS for Argentina). Then we proceed as in Section 3.3 to determine the buffer. Finally we reproject the result back to longitude and latitude coordinates and we extract the extent with `st_bbox` and convert it to a vector with 4 values.

```
utm20s<-"+proj=utm +zone=20 +south +ellps=WGS84 +datum=WGS84 +units=m +no_defs"
ext<-X4326 %>%
  st_transform(crs=utm20s) %>%  # reproject to x/y
  st_union() %>%                # create MULTIPOINT object
  st_convex_hull() %>%          # create POLYGON convex hull
  st_buffer(500) %>%            # create POLYGON buffer
  st_transform(crs=4326) %>%    # reproject back to lon/lat
```

```
  st_bbox() %>%                    # determine extension
  as.vector()                      # vector xmin ymin xmax ymax
```

Finally, we apply `raster::crop` to `srtm`. However, this function requires the extent input as *xmin, xmax, ymin, ymax* so we need to re-order the components of `ext`.

```
elev<-crop(srtm,y=ext[c(1,3,2,4)])
```

Figure 3.9 depicts both elevation and yield variables for the Las Rosas site.

```
mapview(elev,legend=TRUE)+mapview(X4326, zcol="YIELD",cex=2,lwd=0)
```

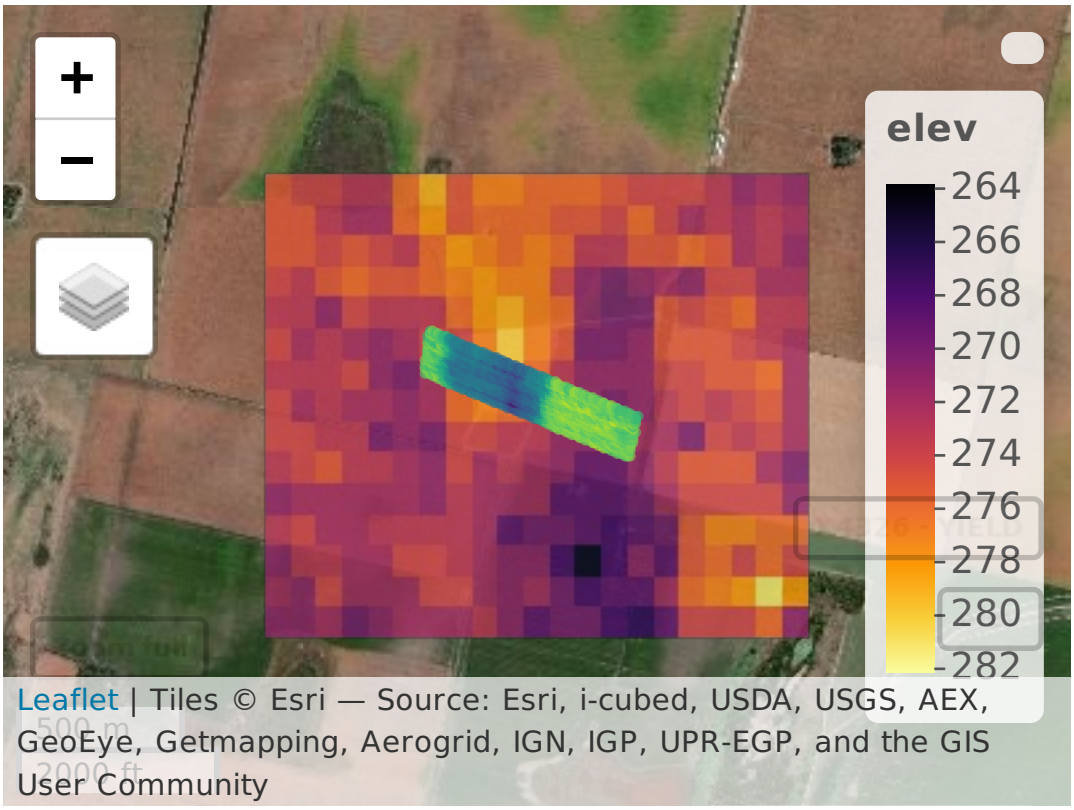


Figure 3.9: The Las Rosas data set in Argentina.

From `elev`, new relief variables can be derived. In particular, we use function `raster::terrain` to compute the slope and the aspect from `elev`. In the example below, slope is measured

in radians (0 indicate a flat surface) and the aspect is also measured in radians (0 indicates North, with the angle growing clockwise).

Function `raster::focal` allow us to apply an arbitrary linear filter and define its kernel. Essentially, a linear filter replaces the pixel value by a weighted sum of all pixels of a moving window which size and weights are determined by the kernel (note that `slope` is also the result of applying a particular linear filter). Since the variation of `YIELD` is mostly along the East-West direction, we use `raster::focal` to estimate the *derivative* along that direction, which is called `slopeX` below. The kernel, determined by argument `w`, is a 3×3 window with appropriate weights. Moreover, we suspect that the yield is related to the accumulation of water in the soil, which can be quantified by the estimate of the *second derivative* of the elevation along the East-West direction. Therefore, we also create a new variable `accu` by applying `raster::focal` to `slopeX`.

```
slope<-terrain(elev,opt="slope", unit="radians")
aspect<-terrain(elev,opt="aspect",unit="radians")
slopeX<-focal(elev,w=matrix(c(-1,-2,-1,0,0,0,1,2,1),ncol=3))
accu<-focal(slopeX,w=matrix(c(-1,-2,-1,0,0,0,1,2,1),ncol=3))
```

We stress that in general slope and aspect should not be computed over geographic (latitude/longitude) coordinates. However, function `raster::terrain` does the correct calculations when the dataset (in our case `srtm`) has a WGS84 coordinate reference system. Furthermore, since the linear filter is applied along one single direction, `slopeX` and `accu` are proportional to the actual estimated values of the *first derivative* and the *second derivative* (the proportionality constant is given by the conversion of degrees of longitude into distances on the ground). As long as we do not care about precise units for `slopeX` and `accu` we do not need to apply the conversion.

Finally, we may wonder if yield is related to the amount of radiation that each location gets. This can be measured by the cosine of the incidence angle of the radiation (0 if the radiation is normal to the surface), which is estimated by function `raster::hillShade`. To apply it, we need to choose a representative location for the sun. Since the plot is on the Southern Hemisphere, we consider that the sun direction is North (azimuth=0º) , which is defined by the argument `direction`). Let us suppose that the sun is relatively high in the sky (late spring conditions) with `angle`=60º indicating that the sun is 60º above the horizon.

```
hshade<-hillShade(slope=slope,aspect=aspect,angle=60,direction=0)
```

3.7.3 Linear interpolation of relief variables

So far, we have created a set of new variables in raster format. We would like to derive the value for each variable at each one of the 1704 locations of data set `X`. This can be achieved by spatially interpolating variables `elev`, `slope`, etc, over the locations in `X`. Since we don't have extra information about the best way of doing the interpolation, we simply perform a linear interpolation over a Delaunay triangulation with function `interp::interp`.

Function `raster::rasterToPoints` returns a matrix with columns longitude, latitude and elevation extracted from the `RasterLayer elev`.

Function `interp::interp` interpolates `z` values over coordinates given by `longs` e `lats`. This returns a vector that can be added to the attribute table of `X`. Option `output = "points"` ensures that interpolated values are computed precisely at the locations given by `longs` and `lats` and not over a grid.

```
interpolate<-function(r,longs,lats)
{
  rtp<-rasterToPoints(r)
  interp(x=rtp[,1],y=rtp[,2],z=rtp[,3],
         xo=longs,yo=lats,output = "points",duplicate="mean")$z
}
```

Function `interpolation` can now be applied to interpolate values of the raster data sets `elev`, `slope`, ..., to the 1704 locations in the Las Rosas data set. The interpolated values are added to `data.frame X4326` as new columns.

```
longs<-st_coordinates(X4326)[,1]
lats<-st_coordinates(X4326)[,2]
X4326$elev<-interpolate(r=elev,longs,lats)
X4326$slope<-interpolate(r=slope,longs,lats)
X4326$slopeX<-interpolate(r=slopeX,longs,lats)
X4326$accu<-interpolate(r=accu,longs,lats)
X4326$aspect<-interpolate(r=aspect,longs,lats)
```

```
X4326$hshade<-interpolate(r=hshade,longs,lats)
```

As a result, `X4326` has now 6 additional variables which could be summarized with the following command.

```
summary(X4326)
```

To examine how those variables vary over the study area, one can use `mapview::sync` which allows us to zoom in a syncronized manner over a set of images. Doing this shows that the pattern of `elev`, in particular, is closely related to the pattern of `YIELD`, which is an indication that `elev` is a relevant predictor for the yield.

```
m1<-mapview(X4326,zcol="YIELD",cex=0.7,lwd=0,legend=TRUE)
m2<-mapview(X4326,zcol="N",cex=0.7,lwd=0,legend=TRUE)
m3<-mapview(X4326,zcol="elev",cex=0.7,lwd=0,legend=TRUE)
m4<-mapview(X4326,zcol="slope",cex=0.7,lwd=0,legend=TRUE)
sync(m1,m2,m3,m4)
```

3.7.4 Tables and spatial data sets for statistical analysis

If we want to extract just the attribute table from `X4326`, to replace the original `data.frame X` with columns `YIELD`, `N`, `LONGITUDE` and `LATITUDE`, we can proceed as follows. First we make a copy of `X4326`, which is a `sf` object with geometry column, and then we drop the geometry column. Since a `sf` spatial object is a `data.frame`, we end up also with a `data.frame` with the remaining (non geometry) columns.

```
X<-X4326
st_geometry(X)<-NULL
```

Interestingly, the correlations between `YIELD` and `slope` or `elev`, are much stronger than the correlation between `YIELD` and the amount of nitrogen fertilizar `N`. Figure 3.10, depicts the relation between `elev` and `YIELD`.

```
round(cor(X),3)
```

```
        YIELD      N   elev  slope slopeX   accu aspect hshade
YIELD   1.000  0.079 -0.881 -0.627 -0.107  0.889 -0.144  0.378
N       0.079  1.000 -0.022  0.008  0.003 -0.001  0.002 -0.043
elev   -0.881 -0.022  1.000  0.584  0.123 -0.954  0.108 -0.306
slope  -0.627  0.008  0.584  1.000 -0.051 -0.525  0.033 -0.368
slopeX -0.107  0.003  0.123 -0.051  1.000  0.016  0.965  0.708
accu    0.889 -0.001 -0.954 -0.525  0.016  1.000  0.015  0.424
aspect -0.144  0.002  0.108  0.033  0.965  0.015  1.000  0.613
hshade  0.378 -0.043 -0.306 -0.368  0.708  0.424  0.613  1.000
```

```
plot(YIELD~elev, data=X, xlab="elevation (m)", ylab="yield (kg/ha)")
```

Figure 3.10: Plot of the yield again the variable `elev`, which shows a trend. Overall, when the elevation increases, the yield tend to decrease, possibly due to the availabily of water in the soil.

At this point we have essentially all the information that is needed for subsequent statistical analysis of the data set in Section 4 and Section 5. To be able to apply sound spatial statistical techniques to the data set we may want to reproject the geographic coordinates into cartographic coordinates that express correctly relative distances.

We consider the local cartographic coordinate reference system (UTM zone 20 South) that was used in Section 3.7.2 and we reproject `X` into that CRS. Furthermore, we add new cartographic coordinates `x,y` to the attribute table so those became easily available as inputs of autocorrelation models.

```
utm20s<-"+proj=utm +zone=20 +south +ellps=WGS84 +datum=WGS84 +units=m +no_defs"
Xsfutm<-st_transform(X4326,crs=utm20s) # sf object
```

To create the corresponding `SpatialPointsDataFrame` `sp` object, one uses `sf::as_Spatial`.

```
Xutm<-as_Spatial(Xsfutm) # sp object
```

3.8 Overview: common functions of packages raster and sf

To read **geotiff** files and other formats one can use `raster::raster`, `raster::brick` and `raster::stack` and to create a new file from a R raster object, the function is `raster::writeRaster`. Coordinate reference systems can be identified or set with `raster::projection`. Spatial resolution is returned by `raster::res` and the extension of a raster object by `raster::extent`. To project a raster onto a new CRS one can use `raster::projectRaster` but `gdalUtils::gdalwarp` is more flexible an efficient, and to get pixel coordinates one uses `raster::coordinates()` that returns a matrix. This can also be done with `raster::rasterToPoints`. `RasterLayer` or `RasterBrick` pixel values are returned by `raster::values`. To extract pixel values at given locations one can use `raster::extract` and to crop a `raster` object using an `sp` object one can use function `raster::crop`. Functions `raster::merge` and `raster::mosaic` are used to mosaic rasters together and return a single `raster` object. Digital elevation models can be explored to derive slope, aspect and hillshading with `raster::terrain` and `raster::hillShade`, but more general linear and non linear filtering techniques can be applied with `focal`.

For vectorial `sf` objects the major functions that were discussed were `sf::st_read` and `sf::st_write` for input/output. Coordinate reference systems for `sf` objects are retrieved or set with `sf::st_crs`. Extension is returned by `sf::st_bbox`. To re-project a data set to a new CRS, one uses `sf::st_transform`. For `sf` objects, function `sf::st_coordinates` returns vertices' coordinates. It also returns indices of the features, parts and rings to which the vertices belong, according to the complexity of the geometry. The geometry of a `sf` spatial data set is returned or set with `sf::st_geometry`. Various function for spatial data analysis were discussed like `st_cast`, `st_union`, `st_buffer`, `st_intersection` or `st_voronoi`. In general, all functions from the GIS simple feature norm (e.g. `st_area`, `st_centroid`, `st_is_valid`, ...) are available under the `sf` package.

Function `raster::rasterize` can be used to convert vector data structures (objects `sf`) into rasters but it is not very efficient. One much faster alternative for POLYGON geometry is `fasterize::fasterize`.

Chapter 4

Tools for Spatial Autocorrelation

We now turn our attention to basic tools to inspect the existence of spatial autocorrelation (Section 4.3), describe the way in which it operates (Section 4.4), measure its intensity (Sections 4.5 and 4.6) and model it (Section 4.7). These concepts are illustrated with the Aragonez dataset (Section 4.1) and a meteorological data set (Subsection 4.8.2).

Spatial autocorrelation is not always easy to identify. One important reason for this is that it may be confused with the existence of some kind of underlying trend in the data which, once removed, would leave deviations (residuals) where spatial autocorrelation is no longer important This issue will be addressed in Section 5.5 and, subsequently, in Chapter 5. The borderline between what is an underlying trend and what is true spatial autocorrelation is often hazy.

Besides the `R` packages discussed previously, a few additional `R` packages will be needed in this Chapter. We begin by loading them (they must have been previously installed on your platform).

```
library(gstat)
library(geoR)
```

4.1 Inspecting the Aragonez yields

Consider again the Aragonez dataset, which was introduced and geo-referenced in Chapter 3. We load the (geo-referenced) object of class `sf` that was created (`AragonezSF`), and inspect it with the `head` `R` command.

```
load(file.path(getwd(), "datasets", "Aragonez.RData"))
head(AragonezSF)  # the first six lines of the AragonezSF object


Simple feature collection with 6 features and 7 fields
geometry type:  POINT
dimension:      XY
bbox:           xmin: -7.516431 ymin: 38.44163 xmax: -7.516231 ymax: 38.44193
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
  genotype block col row colm  rowm yield                   geometry
1    RZ717    B1   4   2    0 93.75 2.417 POINT (-7.516431 38.44193)
2   RZ1158    B1   4   9    0 67.50 2.724 POINT (-7.516291 38.44172)
3   RZ1325    B1   4   6    0 78.75 2.647 POINT (-7.516351 38.44181)
4   RZ3313    B1   4   8    0 71.25 1.543 POINT (-7.516311 38.44175)
5   RZ3603    B1   4  12    0 56.25 0.865 POINT (-7.516231 38.44163)
6   RZ3604    B1   4   3    0 90.00 1.659  POINT (-7.516411 38.4419)
```

The dataset was originally collected to study the yields of different genotypes, the names of which are given in the data frame's first variable (the factor `genotype`), but this information will be ignored for our purposes, and different genotypes will (unwisely) be equated with repetitions. Likewise, we ignore the experimental design, which divided the field trial into 4 different blocks, whose names are the second variable (the factor `block`) in `AragonezSF`. This is not a problem, since the field was divided into four blocks as a 2×2 matrix, and therefore the rows and columns of the rectangular grid provide even more detailed information regarding any possible terrain effect that the block design could capture. The data frame columns with names `col` and `row` provide the location of each cell in terms of its column and row number, respectively. These may be used as a simple form of spatial coordinates. The two subsequent data frame columns, called `colm` and `rowm`, also identify columns and rows but indicating the distance from the center of each grid cell, in meters, to the reference point, which is the southernmost point in the field. Since grid cells are rectangular, and not square, the use of the latter two variables as geographical coordinates is more appropriate than just column and row numbers, insofar as they provide information regarding the spatial distance between the label points of each observation. The next column of the data frame, `yield`, is our variable of interest: the yield (in kg/plant) for each grid cell.

An initial visual inspection of this dataset plots the yields on their spatial coordinates. We can use the `plot` method for `sf` objects to create such a plot. Figure 4.1 shows that there are spatial clusters of similar yields, with a pattern of increasing yields as we move from the left to the right on the trial field.

```
plot(AragonezSF[,"yield"], pch=16)
```

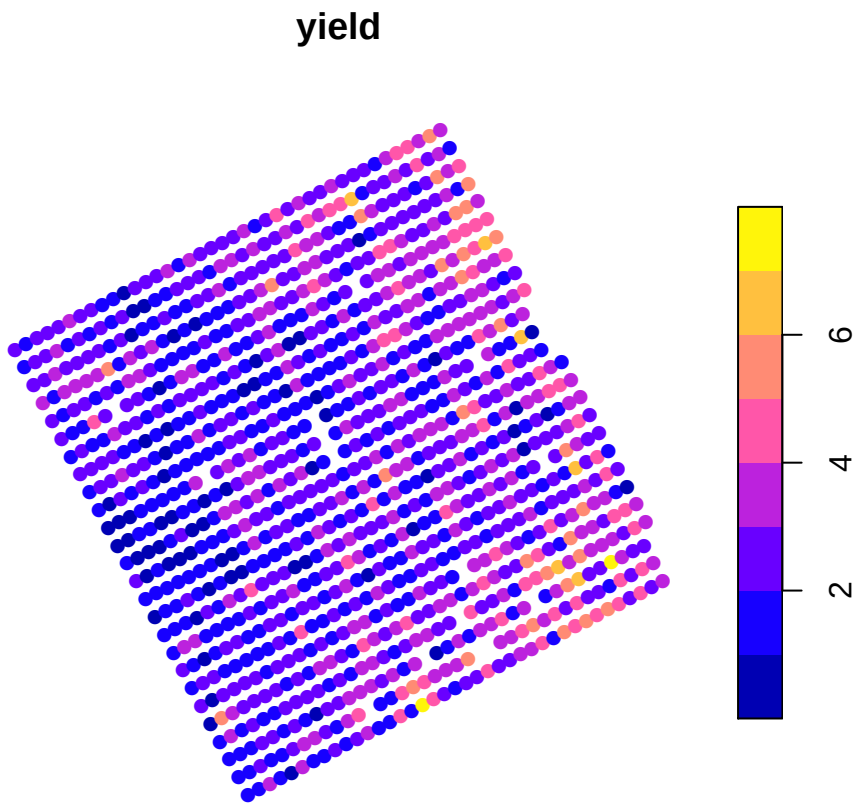


Figure 4.1: A simple plot of the Aragonez yields, produced from the `sf` object `AragonezSF`. The legend indicates yield classes (in kg/plant). Yields increase as we move from left to right on the trial field. The `pch` argument controls the *p*lot *ch*aracter, and 16 is the code for filled circles. Missing values appear as missing points in what would otherwise be a rectangular grid of points.

A similar plot can be produced from the polygon-based `sf` object `Aragonez3763Vor`, created in Chapter 3, as is shown in Figure 4.2. Voronoi tessellations occupy the entire available region, and so missing values do not show up as missing polygons, but rather as irregularly shaped polygons in the midst of the region.

Since many of the functions used in this Chapter are still only available for `sp` objects of class `SpatialPointsDataFrame`, we show in Figure 4.3 how to produce a similar plot using

```
plot(Aragonez3763Vor[,"yield"])
```

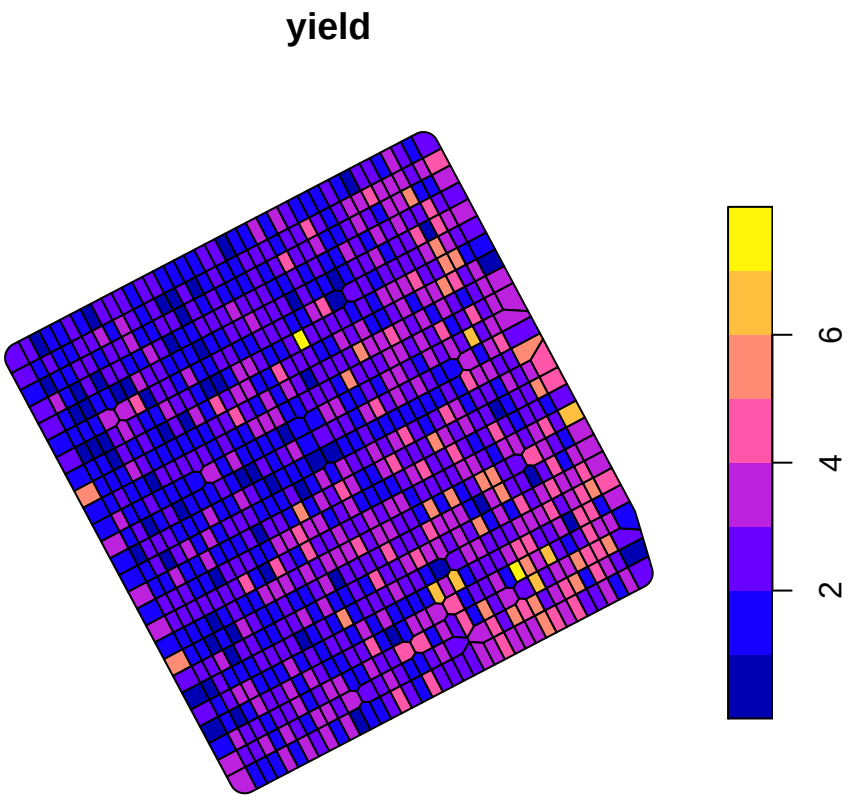


Figure 4.2: A `plot` for the Aragonez yields, based on the polygons in the `sf` object `Aragonez3763Vor`.

the `sp:spplot` function.

Despite the left-to-right pattern of increasing yields, which these Figures highlight, it is not necessarily the case that spatial autocorrelation tools are needed to model this situation. Just as in a classical simple linear regression between two variables Y and X, the underlying trend that we observe in Figure 4.3 may be described by some kind of relationship which, once removed, leaves residual variability where no (or, at least, no significant) spatial autocorrelation is observable.

We now focus on the issue of *detrending* a numerical spatial variable, such as `yield`, in order

```
AragonezPoints<-sf::as_Spatial(Aragonez3763SF)
spplot(AragonezPoints, zcol="yield", key.space="right")
```

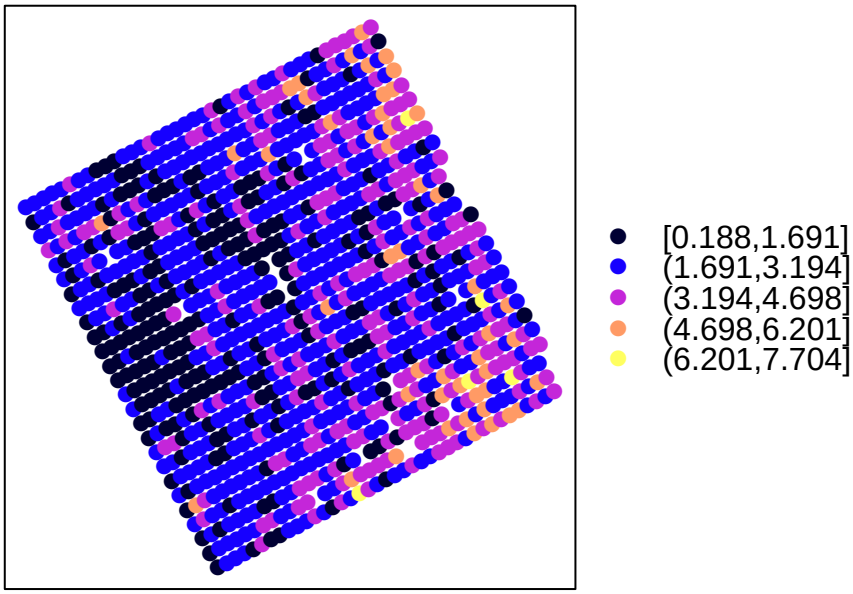


Figure 4.3: A similar plot of the Aragonez yields, generated from the `AragonezPoints` `sp` object, using the `sp::spplot` function. The legend, which indicates yield classes (in kg/plant), is placed to the right of the plot because of the `key.space` argument.

to study whether the remaining variability is affected by spatial autocorrelation.

4.2 Trends and detrending

In order to discuss trends, we will initially assume that the numerical variable of interest, Z, depends on two spatial coordinates x and y, representing the location on the $x0y$ plane of each value of the random process Z. Following Plant ([2]), we will consider a fairly general

decomposition of $Z(x, y)$ into three terms:

$$Z(x, y) \;=\; T(x, y) + \eta(x, y) + \epsilon(x, y) \;, \tag{4.1}$$

where:

- $T(x, y)$ is a deterministic (non-random) underlying spatial *trend*, which is sometimes given as $\mu + T(x, y)$, where μ is an overall mean;
- $\eta(x, y)$ is a *spatially autocorrelated random process*, describing spatially correlated deviations from the underlying trend;
- $\epsilon(x, y)$ is an uncorrelated random process, describing independent error terms.

In Chapter 5 a more general situation will be considered, where the trend T is not just a function of the spatial coordinates x and y, but a function of some other numerical predictors.

It is advisable to remove any underlying deterministic trend $T(x, y)$, that is, to *detrend* the process, in order to check whether spatial autocorrelation tools are needed or if, once a suitable trend is removed, the classical setting of independent random errors adequately models the situation.

One common approach to detrending is to fit a given type of surface by least-squares (regression) fits. Note that the assumption of independent observations is not needed when fitting a surface with the least-squares criterion (it is only necessary for subsequent inferential results), and so standard regression software can be used to fit a trend even when spatial autocorrelation exists. A general form of equation for the surface must be specified, and estimates obtained for the parameters in the surface equation. For example, a flat surface (plane) can be fitted to a data set $\{(x_i, y_i, z_i)\}_{i=1}^{n}$ with a linear regression of the variable z on the coordinates x and y:

$$z \;=\; \beta_0 + \beta_1\, x + \beta_2\, y \;. \tag{4.2}$$

A second-degree polynomial provides curvature, resulting in a paraboloid surface (which can be either an elliptic or a hyperbolic paraboloid, depending on the fitted coefficients):

$$z \;=\; \beta_0 + \beta_1\, x + \beta_2\, y + \beta_3\, x^2 + \beta_4\, y^2 + \beta_5\, xy \;. \tag{4.3}$$

These two examples of surfaces, or other surfaces defined by polynomials of higher degree, can be fitted in `R` using the `lm` command. Other curved surfaces can be considered, and fitted

in R using the command for non-linear regressions, nls, which also minimizes least-squares as a fitting criterion.

When the spatial region under observation is a regular grid, an alternative approach to fitting a trend surface is a non-parametric approach originally suggested by Tukey (Tukey 1977) called *median polish*. Arranging the observed values of Z in matrix form, successive subtractions of row and column medians are carried out until these medians become zero. The resulting detrended data can then be subtracted from the original data to obtain the trend. This approach does not require any functional form to be specified for the trend surface, which is an advantage, but it has the disadvantage of making the results less interpretable and less adaptable to other points on the $x0y$ plane.

We now turn our attention to the analysis of a detrended random process, $Z^*(x, y) = Z(x, y) - T(x, y)$.

4.2.1 Detrending the Aragonez data set

Consider the Aragonez dataset once again. Create two new variables in the data frame, by: (i) subtracting the mean yield (a constant); and (ii) removing a linear trend on the colm and rowm spatial coordinates, as indicated in equation (4.2). The appropriate commands are given below. A few comments regarding these commands:

- *new* variables in the data frame can be added by just writing their full name to the left of R's attribution sign (see the commands below);
- R's linear regression command, lm, also accepts both a SpatialPointsDataFrame object, or an sf object, as its data argument;
- all calls to the variables in the data frame should, for the sp object, more rigorously be made by invoking the @data slot of AragonezPoints(an object of class SpatialPointsDataFrame), as for example in: AragonezPoints@data$yield. But R can cope with the omission of the slot @data in this context.

```
AragonezPoints$yieldct <- AragonezPoints$yield-mean(AragonezPoints$yield)
AragonezPoints$yieldldt <-
     AragonezPoints$yield - fitted(lm(yield ~ rowm + colm , data=AragonezPoints))
# the first lines of the AragonezPoints data frame with the two new columns
head(AragonezPoints)
```

```
  genotype block col row colm  rowm yield     yieldct    yieldldt
1    RZ717    B1   4   2    0 93.75 2.417 -0.13168302  0.87742187
2   RZ1158    B1   4   9    0 67.50 2.724  0.17531698  1.08176580
3   RZ1325    B1   4   6    0 78.75 2.647  0.09831698  1.04876126
4   RZ3313    B1   4   8    0 71.25 1.543 -1.00568302 -0.08456904
5   RZ3603    B1   4  12    0 56.25 0.865 -1.68368302 -0.82122965
6   RZ3604    B1   4   3    0 90.00 1.659 -0.88968302  0.10475672
```

Linear detrending is equivalent to taking the residuals in the above mentioned linear regression, so that the second command above could be replaced by:

```
AragonezPoints$yieldldt <- residuals(lm(yield ~ rowm + colm , data=AragonezPoints))
```

Centred and linearly detrended yields were also added to the `sf` objects `Aragonez3763Vor` and `AragonezGrid`, that was created in Chapter 3.

```
Aragonez3763Vor$yieldct <- AragonezPoints$yieldct
Aragonez3763Vor$yieldldt <- AragonezPoints$yieldldt
head(Aragonez3763Vor)


Simple feature collection with 6 features and 9 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 53834.87 ymin: -135972.3 xmax: 53843.84 ymax: -135957.3
epsg (SRID):    3763
proj4string:    +proj=tmerc +lat_0=39.66825833333333 +lon_0=-8.133108333333334 +k=1 +x_0=0 +y
  genotype block col row colm  rowm yield                       geometry
1    RZ717    B1   4   2    0 93.75 2.417 POLYGON ((53837.5 -135959.4...
2   RZ1158    B1   4   9    0 67.50 2.724 POLYGON ((53839.54 -135958....
3   RZ1325    B1   4   6    0 78.75 2.647 POLYGON ((53836.11 -135965....
4   RZ3313    B1   4   8    0 71.25 1.543 POLYGON ((53841.53 -135967,...
5   RZ3603    B1   4  12    0 56.25 0.865 POLYGON ((53841.58 -135957....
6   RZ3604    B1   4   3    0 90.00 1.659 POLYGON ((53837.88 -135969,...
      yieldct    yieldldt
1 -0.13168302  0.87742187
2  0.17531698  1.08176580
3  0.09831698  1.04876126
```

```
4 -1.00568302 -0.08456904
5 -1.68368302 -0.82122965
6 -0.88968302  0.10475672


AragonezGrid$yieldct <- AragonezPoints$yieldct
AragonezGrid$yielddt <- AragonezPoints$yielddt
head(AragonezGrid)


Simple feature collection with 6 features and 9 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: -7.516453 ymin: 38.44161 xmax: -7.516209 ymax: 38.44195
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
  genotype block col row colm  rowm yield                       geometry
1    RZ717    B1   4   2    0 93.75 2.417 POLYGON ((-7.516453 38.4419...
2   RZ1158    B1   4   9    0 67.50 2.724 POLYGON ((-7.516312 38.4417...
3   RZ1325    B1   4   6    0 78.75 2.647 POLYGON ((-7.516373 38.4418...
4   RZ3313    B1   4   8    0 71.25 1.543 POLYGON ((-7.516333 38.4417...
5   RZ3603    B1   4  12    0 56.25 0.865 POLYGON ((-7.516252 38.4416...
6   RZ3604    B1   4   3    0 90.00 1.659 POLYGON ((-7.516433 38.4419...
      yieldct    yielddt
1 -0.13168302  0.87742187
2  0.17531698  1.08176580
3  0.09831698  1.04876126
4 -1.00568302 -0.08456904
5 -1.68368302 -0.82122965
6 -0.88968302  0.10475672
```

4.3 Detrended Plots

Once a spatial process has been detrended, any residual variability may, or may not, reveal spatial autocorrelation. A spatial plot of the detrended values will help to highlight the existence of any remaining spatial autocorrelation, which will then appear as clusters of above-trend (positive) values and clusters of below-trend (negative) values of similar size. If the remaining variability were independent (not spatially autocorrelated), positive and negative values would be distributed at random.

One simple visual aid are *bubble plots*. These are simply plots of the observations on the underlying spatial coordinates, but where the symbol used to represent each observation is scaled, and/or depicted with a certain colour coding, so as to provide information regarding the observed values at each point.

In `R`, the `sp::bubble` command creates bubble plots from `sp` objects. The command requires at least two arguments: (i) the name of a `SpatialPointsDataFrame` object providing the spatial coordinates of these points; and (ii) argument `zcol`, providing the name of the variable that will define the bubbles. By default, the command assumes that the variable has been detrended, and provides a two-colour code for negative, and for positive, deviations from the trend. There is also a default scaling effect, to highlight the magnitude of the data values. The `bubble` command has a number of arguments, which are described in the corresponding helpfile. Unfortunately, the `bubble` command does not, at present, accept a vector of variable names in the `zcol` argument, which would allow multiple bubble plots to be drawn side by side.

Figure 4.4 gives the bubble plot for the centred Aragonez yields (yields minus the mean yield). The bubble plot provides similar information to the previous `spplot` of (uncentred) yields: most below-average yields are concentrated on the left-hand side of the grid, with most above-average yields concentrated in the upper right and lower right corners. Both the sign and size of the deviations appears to be spatially clustered: merely centring the data cannot eliminate the spatial pattern observed on the original yields.

Figure 4.4 suggests the existence of a linear trend on the spatial coordinates, in other words a trend represented by a plane that slopes upwards as we move from left to right in the field.

The bubble plot for yields detrendend by subtracting a linear trend on the spatial coordinates is given in Figure 4.5, using the variable `yieldldt`, as defined above.

The much more irregular layout in Figure 4.5 suggests that removing a linear trend has partially broken down the pattern of similar values. But the persistence of clustered patches of values of similar sign and magnitude suggests that there is still spatial autocorrelation in the detrended data. The remaining clusters may, of course, result from an unsuitable detrending. This can also occur in the familiar case of a 2-variable scatterplot, when a linear regression is fitted to a curved relation: in this case, sequences of negative and positive residuals would be highlighting the inadequate nature of a linear trend, and not (necessarily) spatial autocorrelation. On the other extreme of the scale, there is the risk of overfitting when detrending, leaving little residual variability left to explain.

The `sp::spplot` command is a more flexible and powerful command than `sp::bubble`. It

```
bubble(AragonezPoints, zcol="yieldct")
```

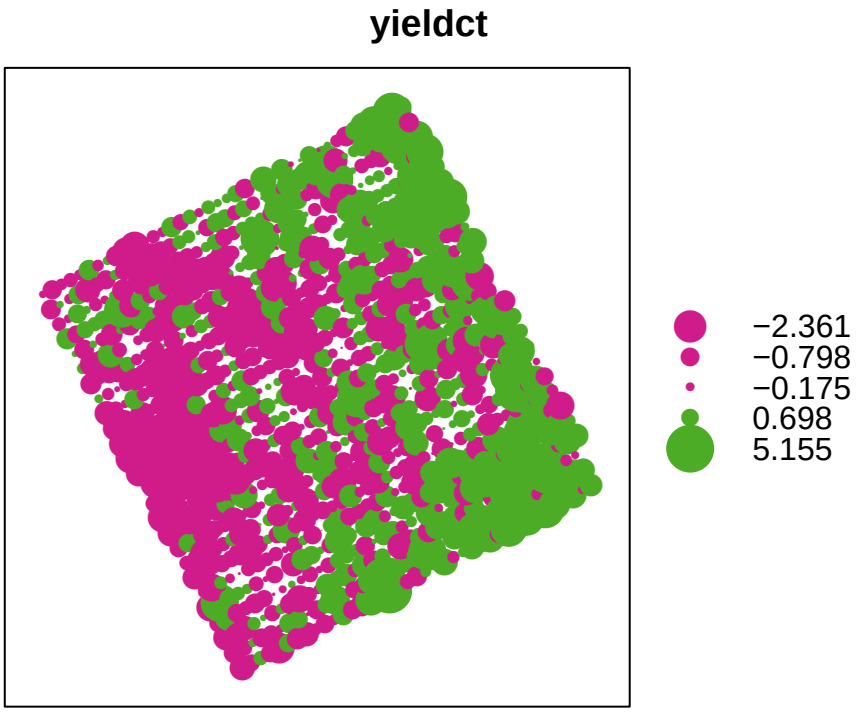


Figure 4.4: A `sp::bubble` plot for the centred Aragonez yields. The magenta points represent negative (below-average) centred yields. Green points represent above-average yields. The circles become larger as the deviation from the mean grows. The five values indicated next to the colour keys are the five values used to build boxplots (the minimum and maximum, as well as the three quartiles) and the symbols next to the values indicate the corresponding point size.

uses the `lattice` package for graphical output in the *Trellis* graphics system (Cleveland, 1993, 1994). It caters for more types of spatial data classes than `bubble`. In particular, it is a useful command to visualize spatial data of polygon type (although `bubble` also accepts `sp` objects of class `SpatialGridDataFrame`). We illustrate the use of the `spplot` command by creating an object of class `SpatialPolygonsDataFrame` for the AragonezGrid. As can be seen in the plot for linearly detrended yields, in Figure 4.6, when polygons are created in

```
bubble(AragonezPoints, zcol="yieldldt")
```

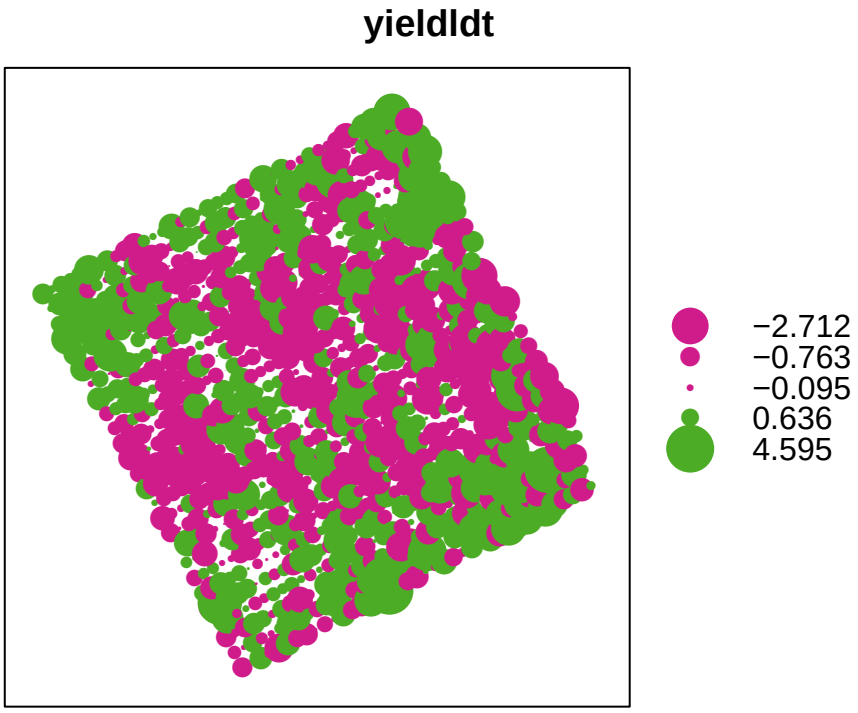


Figure 4.5: A bubble plot for the Aragonez yields, detrended with a linear regression on the x (column) and y (row) distances. The magenta points are associated with below-trend yields. Green points represent above-trend yields. The circles become larger as the deviation of the yields from the mean grows.

this way, missing values appear as empty polygons. The information provided by Figure 4.6 is essentially the same as in Figure 4.5.

In `sp::spplot`, the `zcol` argument may be a vector of variable names. When more than one variable is requested through the `zcol` argument, separate plots are given for each variable, with a common colour code for their values, as illustrated in Figure 4.7. Invoking the function in this simple way may not particularly useful, unless the different variables have comparable values. This problem can be seen in Figure 4.7, since negative yields do not exist prior to

```
AragonezPolygons <- as_Spatial(AragonezGrid)
spplot(AragonezPolygons, zcol="yieldldt")
```

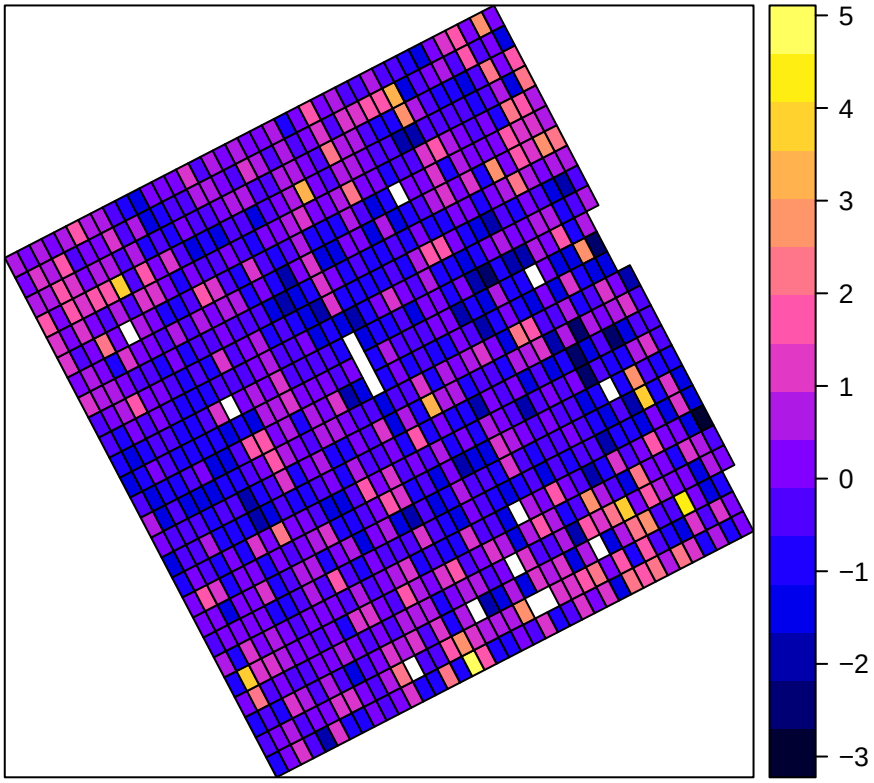


Figure 4.6: An `spplot` for the Aragonez yields, detrended with a linear regression on the x (column) and y (row) distances (in meters). When the `sp` object of class `SpatialPolygons` is created in this way, missing values appear as missing polygons.

detrending (that is, for the `yield` variable).

It is tempting to introduce measures of spatial autocorrelation straight away. But indices for spatial autocorrelation (such as Moran's I and Geary's c, which will be introduced in Section 4.5), require a discussion of the all-important issue of *spatial weights*.

```
spplot(AragonezPoints, zcol=c("yield","yieldct","yielddt"), key.space="right")
```

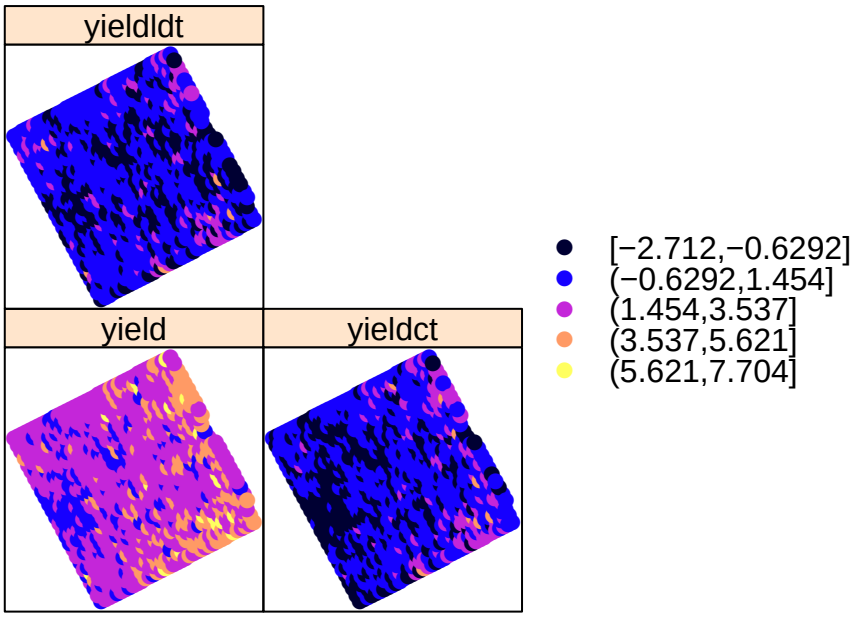


Figure 4.7: The `spplot` for the three yield variables in `AragonezPoints`.

4.4 Spatial weights and graphs

Spatial autocorrelation may be modelled in different ways. It may be assumed that it affects only the error terms, in other words, the deviations of the spatial process Z from some underlying trend. Or one may assume that spatial autocorrelation is directly impacting the spatial process Z, so that the values of Z at some location s_1 are, in part, the result of the values of Z in neighbouring locations, regardless of trend. It may also be the case that spatial behaviour of Z is affected by some other variable which, once included in the model, may account for all the observed spatial autocorrelation. These issues will be discussed in Chapter 5. But all such models share a common feature, which is the notion of *spatial*

weights. A spatial weight w_{ij} is a means of both indicating whether some observation (or error) at a spatial location s_j affects an observation (or error) at location s_i and, if so, how strong this effect is.

In order to better understand this key concept of spatial weights, we begin by relating it to the one-dimensional autocorrelated error model discussed in Chapter 2. We assume that, in equation (4.1), the trend $T(x, y)$ is given by a constant μ (or, equivalently, $Z(x, y)$ has been essentially detrended, except maybe for a constant μ) and that the values of $\eta(x, y)$ depend on the values of η at other points in the vicinity of (x, y). More specifically, we assume that each η is given by a *linear combination* of other error terms η, as follows:

$$\begin{cases} Z_i &= \mu + \eta_i \\ \eta_i &= \lambda \left(\sum_{j=1}^{n} w_{ij}\eta_j \right) + \epsilon_i \\ \epsilon_i &\sim \mathcal{N}(0, \sigma^2) \qquad (i.i.d.) \ , \end{cases} \tag{4.4}$$

where w_{ij} is a constant measuring the influence of the error term η_j, for observation Z_j, on the error term η_i, for observation Z_i. The parameter λ may be thought of as an overall measure of the intensity of spatial autocorrelation. The one-dimensional AR(1) model (2.8) is a specific instance of this model, in which the only non-zero weights occur when $j = i - 1$, in which case $w_{i,i-1} = 1$.

Model 4.4 extends model (2.8), both in that it allows for more than one observation to affect the i-th observation (which is better suited for *spatial* autocorrelation), and in that it allows for more flexibility in defining the size of those weights, that is, the intensity of those effects.

A zero spatial weight, $w_{ij} = 0$, indicates that η_j does not affect η_i, whereas non-zero weights indicate that such an effect exists. Weights greater than 1 would imply that the effect of a neighbouring observation tends to be greater than the observation itself, which is seldom the case. As a general rule, we assume that the spatial weights w_{ij} verify the condition $0 \leq w_{ij} \leq 1$.

As was seen, two different (although inter-related) issues are at stake when defining spatial weights:

- identifying which observations Z_j (or errors) affect any given observation Z_i (often called the *neighbours* of Z_i), in other words, which weights w_{ij} are non-zero; and
- specifying the intensity of those effects that do exist (in other words, the values of non-zero weights w_{ij}).

A discussion of the first of these issues is helped by the mathematical notion of a *graph*, which will be very briefly introduced in the next Subsection.

4.4.1 Graphs: some introductory concepts

A graph is a set V of *vertices* (or *points*, or *nodes*), pairs of which may be united by *edges* (or *lines*, or *arcs*). The existence of an edge between a pair of vertices is associated with some property of interest. The set of edges can be represented by E, and the graph is given by both sets: $G = (V, E)$. Figure 4.8 shows a graph with $n=9$ vertices and 12 edges.

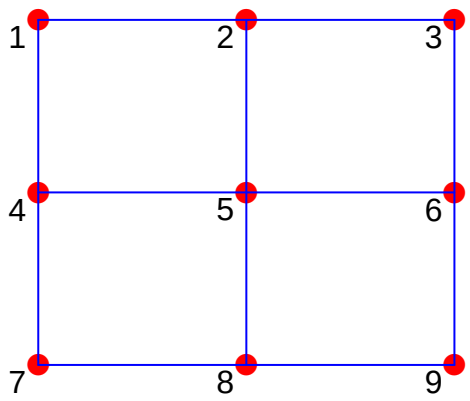


Figure 4.8: A (undirected) graph with nine vertices (numbered by row), and 12 edges.

The number of vertices is called the *order* of the graph and denoted by $|V|$. The number of edges, which is represented by $|E|$, is sometimes called the *size* of the graph.

In our context, the set of n points, or polygons, in space for which we have a set of obser-

vations $\{z_i\}_{i=1}^n$ is associated with the set of vertices, so that $n = |V|$. An edge connecting a pair of vertices, i and j, indicates that observation (vertex) j affects observation (vertex) i. In other words, there will be an edge uniting vertex i to vertex j if and only if w_{ij} is a non-zero weight in the weights matrix.

The fact that observation j affects observation i may not necessarily imply that observation i affects observation j. If this is the case, we need to specify *directed* edges: an edge with an initial vertex v_i and a terminal vertex v_j will not be the same thing as an edge with initial vertex v_j and terminal vertex v_i (which may even not exist). In this case, we speak of a *directed graph*, or *digraph*.

Consider once again the Aragonez dataset. Let us assume that any given observation (vertex) is influenced by observations (vertices) that are a distance of $4m$ or less, and that this influence is symmetric, so that an undirected edge can be established between the two vertices representing those observations in a graph. The resulting graph is given in Figure 4.9 (we will see later the `R` functions that were used to build it).

We say that a given vertex v_i is *incident* with a given edge if that edge unites v_i with another vertex v_j, in which case the edge can be identified as $e_{ij} = (v_i, v_j)$. For example, the central vertex in Figure 4.8 is incident with 4 edges. The *degree* (or *valency*) of a vertex is the number of edges incident with that vertex. Thus, vertex 5 in Figure 4.8 is of degree 4, whereas the four corner vertices (1, 3, 7, 9) are of degree 2 and all other vertices in that (very small) graph are of degree 3. For directed graphs, it is necessary to distinguish between the *in-degree* and the *out-degree* of a vertex which are the number of edges that respectively end, and begin, at that vertex.

In a graph, two vertices that are united by an edge are called *adjacent*. One way of fully specifying a graph is through its *adjacency matrix* A, in which both rows and columns are associated with the set of vertices. The matrix element a_{ij} is therefore associated with the pair of vertices v_i and v_j, and it can take two values: $a_{ij} = 1$ if v_i and v_j are adjacent (that is, if edge e_{ij} exists), and $a_{ij} = 0$ if they are not. Adjacency matrices for undirected graphs are symmetric, that is, $a_{ij} = a_{ji}$ for any i,j, or equivalently, $A^t = A$. For directed graphs, the adjacency matrix is not symmetric ($A^t \neq A$).

For applications in spatial statistics, the standard convention is that a vertex is not adjacent to itself, so that the diagonal elements in the adjacency matrix are all zero. For the example

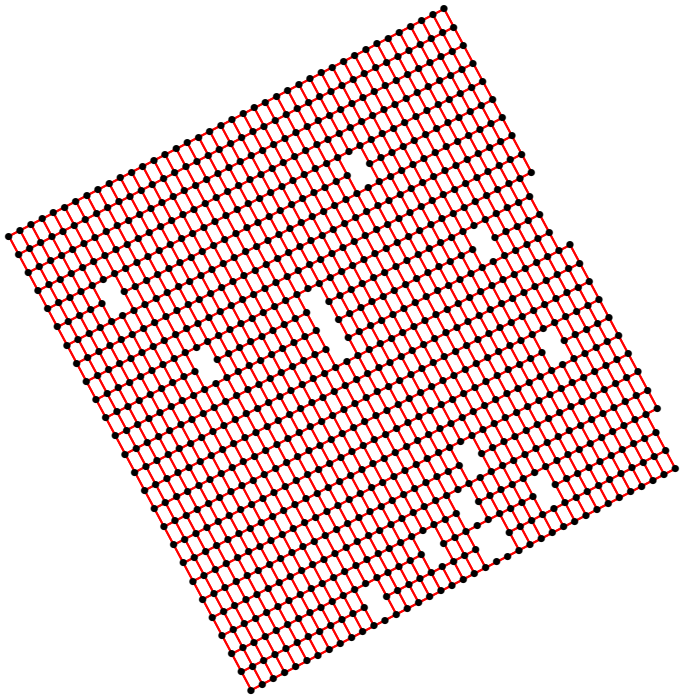


Figure 4.9: The graph of all non-zero spatial weights for the Aragonez dataset, assuming that any observation is influenced by all other observations within a radius of 4 meters.

in Figure 4.8, and numbering the nine vertices by row, the adjacency matrix is:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{4.5}$$

Since each matrix row corresponds to a vertex, the row sums of an adjacency matrix give us the degree of each vertex. Each column also corresponds to a vertex, and the columns sums also give the degree of each vertex. For undirected graphs, row sums are equal to column sums, but this is not in general the case for directed graphs. The adjacency matrix of a directed graph is still defined as having $a_{ij}=1$ if and only if the edge e_{ij} with initial vertex j and terminal vertex i exists (this is the convention that is coherent with the indices of the spatial weights w_{ij} in equation 4.4). But it is no longer necessarily true that $a_{ij}=a_{ji}$. In this case, the row sums of the adjacency matrix give the in-degree of each vertex, and column sums give out-degrees.

For graphs of very high order (the number of vertices, n, is very big) adjacency matrices become very large and very demanding in terms of storage memory. In such cases, and in particular when the number of edges is not extremely large, a more efficient way of storing the information for an adjacency matrix is in an *adjacency list*, which is a list of n *vectors*, where each vector (list object) i contains the vertex numbers adjacent to vertex i or, in the case of a directed graph, the initial vertices for any edge with terminal vertex i. For the above example, the list would have 9 objects, the first of which is the vector for vertex 1: $(2, 4)$; the second is the vector for vertex 2: $(1, 3, 5)$; and so on.

If there is a sequence of edges that begins at edge v_i and ends in edge v_j, we say that there is a *walk* between the vertices v_i and v_j, and a *path* if $v_i \neq v_j$. Thus, in Figure 4.8 there is a path between vertices 1 and 5, since there is an edge uniting vertices 1 and 2, and another uniting vertices 2 and 5. In this path, vertex 2 is called an *inner* vertex, and we say that vertices 1 and 5 are *linked*. The *length* of a path is the number of edges in that path. In the previous example, the path is of length two. Two vertices may be linked by more than one path, and in the example, there is also a path of length four connecting vertices 1 and 5, consisting of the edges e_{12}, e_{23}, e_{36}, and e_{65}. This path, like any other, may also be represented by its ordered set of vertices: $(v_1, v_2, v_3, v_6, v_5)$. The length of the shortest path connecting two vertices is called the *distance* between the two vertices (if no such path exists, the distance is set to ∞). The maximum distance in a graph is called the *diameter* of the graph.

We say graph is *connected* if all pairs of vertices are linked by a path. This is the case with the graphs in both Figure 4.8 and Figure 4.9. A graph that is not connected has *separate components* (maximal connected subgraphs), so that a path between any pair of vertices exists if, and only if, those vertices belong to the same component. In our context, this means that we are assuming that vertices that are in different components are not spatially autocorrelated. This may be an appropriate assumption in the case of, for example, physical barriers that separate the locations where different subsets of observations were made.

A *weighted graph* is a graph in which edges have weights. Weights can be used to give different strengths to the connections between vertices. In our context, weighted graphs may be used to represent the spatial weights associated with each pair of observation errors, η_i and η_j. The following Subsection addresses the issue of how these weights can be assigned.

4.4.2 Spatial weights matrices

The $n \times n$ matrix $\mathbf{W}$, whose (i, j)-th element is w_{ij} is called a *spatial weights matrix*. Spatial weights matrices play a crucial role in the analysis of spatial data.

Whenever a directed graph is needed to describe the neighbours of each observation, because an edge (v_i, v_j) does not always imply the existence of the edge (v_j, v_i), then a corresponding spatial weights matrix cannot be symmetric. However, even when an undirected graph is in order, because adjacencies are symmetric, it may still be the case that an associated weight matrix is not symmetric. This depends on the precise way in which weights are assigned to each pair of neighbours.

Before considering in more detail some specific ways of assigning spatial weights, a few general comments are in order.

1. as seen above, it makes sense to assume that $0 \leq w_{ij} \leq 1$, for all pairs (i, j), with $w_{ij} = 0$ if and only if observation (error) j does not influence observation (error) i.

2. if all non-zero weights are set equal to 1, the weight matrix $\mathbf{W}$ coincides with the adjacency matrix $\mathbf{A}$ of the graph of adjacencies.

3. it may also be argued that a constraint should be imposed on the sum of all weights describing effects on the i-th observation. For example, it may be required that the sum of all weights associated with observations that influence the i-th observation be set to 1, in other words, $\sum_{j=1}^{n} w_{ij} = 1$. One way of doing so is by assigning equal weights to all observations that influence observation i, and so $w_{ij} = \frac{1}{d_i}$, where d_i is the in-degree of vertex i in the graph of neighbours (or just degree, for an undirected graph). This is called a *row-normalized weight matrix*. On statistical grounds, it may be justified with the idea that for a vertex of (in-)degree $d_i = 4$, the contribution of each of the four individual observations that affect it is, in relative terms, not as big as would be the case if the degree of that vertex were only 1 or 2.

4. Alternatively, it may be decided to impose an overall constraint on the size of the weights, such as setting the sum of all weights to some value.

We now consider some specific rules for assigning spatial weights.

4.4.3 Distance-based weights

For *geostatistical data*, it is usually appropriate to define spatial weights matrices with weights w_{ij} given by some function of the spatial separation between observations $Z_i = Z(s_i)$ and $Z_j = Z(s_j)$, assuming that this separation can be measured on some *continuous* scale. The standard Euclidean distance between points is a common choice. In general, not all pairs of observations will have non-zero weights (although this is conceptually possible), since it may be considered appropriate to ignore spatial autocorrelation for points that are further apart than some threshold distance. In particular, we may:

- Define weights by some non-increasing function, g, of the *scalar Euclidean distance* d_{ij} between the coordinates of the points at which observations i and j were made:

$$w_{ij} \;=\; g(d_{ij}) \;. \tag{4.6}$$

  Different choices for function g allow us to control the strength of the influence of points that are at a given distance d_{ij} apart. Some frequent choices for distance functions g are:

  1. the **radial distance weight function**: a pair of observations at a distance closer than some parameter d has weight 1, and the weight for observations made further apart is zero:

$$w_{ij} \;=\; \begin{cases} 1 & , \text{ if } 0 \leq d_{ij} \leq d \\ 0 & , \text{ if } d_{ij} > d \end{cases} \tag{4.7}$$

  2. the **inverse (power) distance weights function**: weights decrease with some power of the distance. For some positive constant a:

$$w_{ij} \;=\; \frac{1}{d_{ij}^{a}} \;; \tag{4.8}$$

     This family of weight functions is often used in *interpolation* problems. The larger the power a, the less influential are points that are further away.

  3. the **exponential distance weight function**: weights decrease exponentially with distance. For some positive constant a:

$$w_{ij} \;=\; \mathrm{e}^{-a\, d_{ij}} \;. \tag{4.9}$$

Implicit in equation (4.6) is the idea that the weights depend only on the scalar distance d_{ij}, regardless of the *direction* which separates the points at which observations Z_i and Z_j were made. This assumption, which is called the **isotropy** assumption, may, or may not, be realistic.

- A more complex definition for a spatial weights matrix may assume that, for any given direction, the weights would decrease with scalar distance, but the precise way in which they would decrease would differ, for different directions. This is the **anisotropy** assumption. In this case, w_{ij} would be a function of the distance *vectors* uniting each pair (i, j) of observed points.

It may be considered appropriate to combine the use an exponential, or inverse distance, weight function with a threshold, such that w_{ij} becomes zero when d_{ij} exceeds some threshhold d. Other definitions of distance-based weights may be suggested by the specificities of any given application. For example, geographical barriers (seas, mountain ranges, etc.) that are considered to cut off any autocorrelation between observations may suggest that a rule specifying weights based on Euclidean distances be modified, so as to exclude weights for a pair of observations that lie on opposite sides of that geographical barrier. This corresponds to deleting edges in the adjacency graph.

4.4.4 Neighbours and k-th order neighbours

For *areal data*, other possible definitions of a spatial weights matrix $\mathbf{W}$ may be more appropriate. Consider a process Z observed on some spatial arrangement of polygons, where the concept of **neighbouring polygon** can be defined. It is assumed that only neighbouring polygons (cells) affect any given observation Z_i of the process. Neighbours can also be defined for values observed at *points* in space (and therefore for geostatistical data), either by defining a distance-based concept of neighbourhood, or by creating a tessellation of regions surrounding the points and using the resulting polygons to define pairs of neighbours.

A standard convention is that a polygon is not a neighbour of itself. Possibilities for the definition of neighbours include two famous conventions, the *rook's case* and the *queen's case*:

- **Rook's case**: polygons are considered neighbours if they share a common border of length greater than zero. The name *rook's case* originates from the adjacent chessboard squares to which a rook can move, as illustrated in Figure 4.10. The patchwork of cells does not have to be a rectangular grid for the definition to apply.

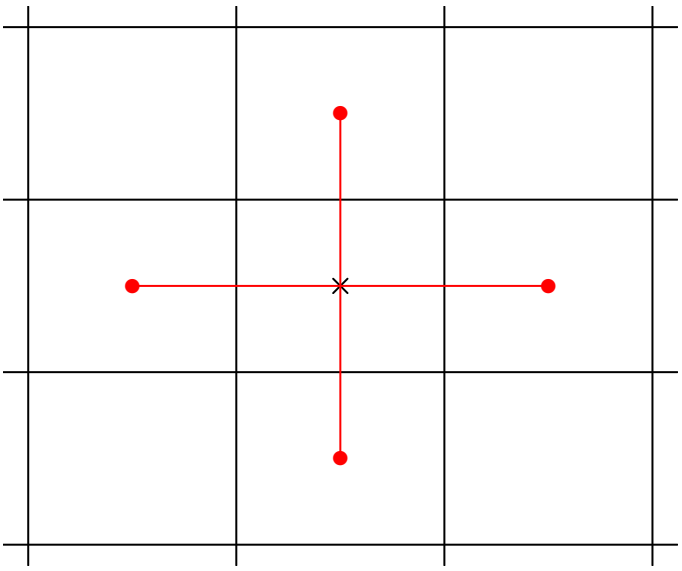


Figure 4.10: For a given observation at the location with center $\times$, *rook's case* neighbours are the cells (red circles at the center) with borders of length greater than zero.

- **Queen's case**: polygons are considered neighbours when they touch each other, even if only at a single point, that is, when they have non-empty borders. The name *queen's case* is again inspired by the possible movements of a queen on a chessboard, as illustrated in Figure 4.11.

Other definitions of neighbours are, of course, possible, and may be justified by the specific nature of a given problem.

An initial definition of neighbours may be too restrictive, since spatial autocorrelation may also be felt beyond these immediate neighbours. The concept of **k-th order neighbours** may be useful in assessing this. Once a set of neighbours for each observation has been specified, we may consider **neighbours of order** k ($k \in \mathbb{N}$) in the following way. The initially

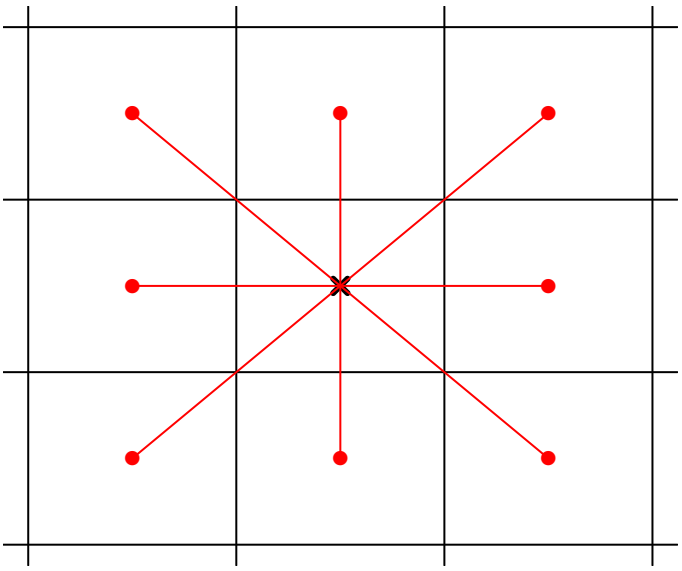


Figure 4.11: For a given observation at the location with center ×, the *queen's case* neighbours are all adjacent cells (red circles at the center), even if only adjacent at a single point

specified neighbours are considered *first*-order neighbours. The *second*-order neighbours of any given vertex are the neighbours of its neighbours (who are not first-order neighbours). *Third*-order neighbours are the neighbours of the neighbours' neighbours (who are not neighbours of order 1 or 2), and so on.

Once again, graph theory (see Subsection 4.4.1) has useful concepts to discuss this notion. For any given vertex, its second-order neighbours are the vertices that are at a distance 2 (the shortest path between them is of length two), third-order neighbours are at a distance 3 and, in general, k-th order neighbours are at a distance k. Thus, by initially specifying first-order neighbours we are introducing a possible concept of distance between observations, which is not equivalent to Euclidean distances.

In the example of Figure 4.8, vertices 3, 5 and 7 are the second-order neighbours of vertex 1 (they are at a distance two). Vertex 1 has two third-order neighbours (vertices 6 and 8) and a single fourth-order neighbour (vertex 9).

4.4.5 Defining neighbour-based weight matrices

Once the set of neighbours has been defined, the specific weights w_{ij} must be specified for each pair of neighbours. A few common options in the spatial data literature are the following:

- The **binary weights matrix** is the adjacency matrix of the graph of neighbours: $w_{ij} = 1$ if the observation at polygon/point j affects the observation at polygon/point i, and $w_{ij} = 0$ otherwise. If the associated graph is undirected, it will be a symmetric matrix. For the example in Figure 4.8, the binary weights matrix is the adjacency matrix 4.5, and therefore symmetric. Binary weights are similar to a radial distance weight function, although neighbours may be defined in ways that are not direct functions of a distance.

- The **row-normalized weights matrix**, for which all non-zero weights w_{ij} in a given row are equal, and the row sum is 1: $\sum_{j=1}^{n} w_{ij} = 1$. In a row-normalized weight matrix, for a given a pair of neighbours i and j the weight is given by $w_{ij} = \frac{1}{d_i}$, where d_i is the in-degree of vertex i in the graph of neighbours (or just degree, in the case of an undirected graph). A matrix of this kind is in general *not* symmetric: $w_{ij} \neq w_{ji}$ for some i,j, even when the adjacency matrix is symmetric. Its usage is fairly common. Again, for the rook's case of Figure 4.10, the row-normalized weights matrix would be:

$$\mathbf{W} = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{bmatrix} \tag{4.10}$$

- The **globally standardized by the mean number of edges weight matrix**, which

is the binary weights matrix (4.5) divided by $\frac{|V|}{|E|}$, where $|V|$ is the order of the graph (i.e., the number of vertices in set V) and $|E|$ is the size of the graph (i.e., the number of edges in set E). Thus, the non-zero weights (at the same positions as in the binary weights matrix) have value $\frac{|E|}{|V|}$, and they add up to $n=|V|$, the number of observations.

- The **globally standardized by the total number of edges weight matrix**, which is the binary weights matrix divided by $|E|$, the total number of edges (adjacent pairs of vertices) that were specified. In this case, all non-zero elements of $\mathbf{W}$ are $\frac{1}{|E|}$, and their sum total is 1.

4.4.6 Defining neighbours with `R` packages

The `R` package `spdep`, by numerous authors, the first of which is Roger Bivand, provides a large number of functions that assist in creating weights matrices. Creating a neighbour-based weights matrix is, as described above, a two-stage process. In a first step it is necessary to specify which observations are to be considered neighbours of any given observation Z_i (the second step involves deciding what spatial weight should be associated with each pair (i, j) of neighbours).

Figure 4.12 summarizes the various commands that will now be covered.

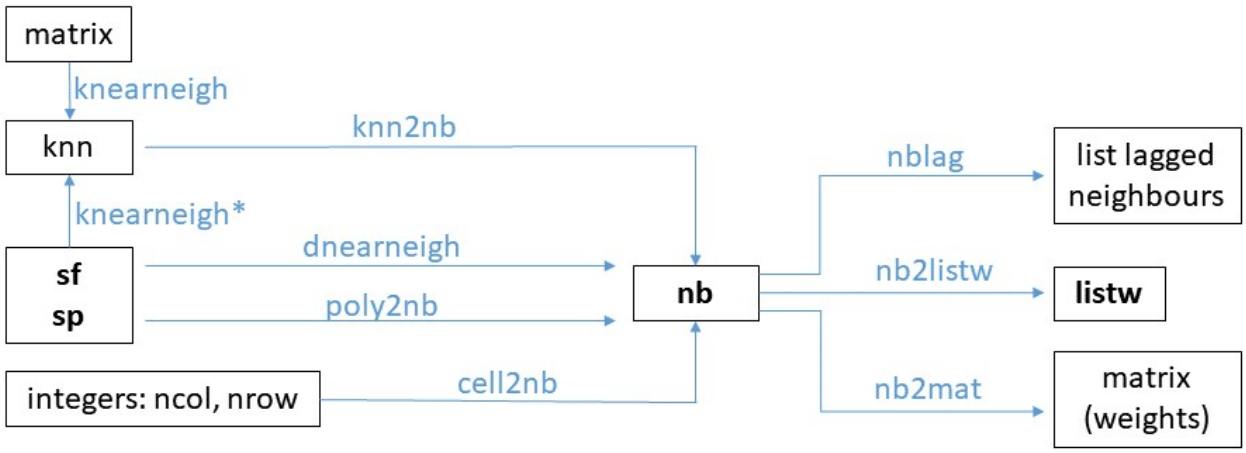


Figure 4.12: Summary of `R` commands to create and work with neighbours lists and weight matrices/lists. (*) At the moment, the `knearneigh` function has not yet been adapted to accept `sf` inputs.

Package `spdep` provides a class called `nb` for *neighbour lists*. More details about this class can be found in the `nb` vignette, which can be invoked (after loading the `spdep` package)

with the command:

```
vignette("nb")
```

Objects of class `nb` store information about which pairs of objects are to be considered neighbours. This information is stored in a compact way, that is, as an adjacency list (see Subsection 4.4.1) with n vector components indicating the neighbours of each vertex i.

The following `spdep` commands create `nb` objects:

spdep::cell2nb assumes that we have a rectangular grid with `nrow` rows and `ncol` columns. These two values must be specified as arguments to the command. By default the command uses the *rook's case* to create neighbours, but the argument `type="queen"` will use the queen's case instead. The appropriate commands for the rook's case 3×3 example given above is:

```
cell2nb(3,3)


Neighbour list object:
Number of regions: 9
Number of nonzero links: 24
Percentage nonzero weights: 29.62963
Average number of links: 2.666667
```

The displayed information states that there are 9 polygons, or cells (represented by graph vertices), in the 3×3 grid, which in theory could provide $9^2 = 81$ links (edges) between pairs of neighbours, counting edges between each cell and itself (also called *loops* in graph theory), and also counting directed edges uniting a same pair of vertices, such as edges e_{12} and e_{21}, as different edges. In other words, there are a maximum of n^2 graph edges in a *directed* graph of neighbours, and allowing for edges from a vertex to itself. Of these, only 24 are, in fact, existing pairs of neighbours (according to the default rook's case criterion), which gives a percentage of $\frac{24}{81} \times 100\% = 29.62963\%$. The average number of links (edges) per cell (vertex), that is, the mean degree, is $\frac{24}{9} = 2.666667$. The way in which the information on neighbours is stored can be seen by inspecting the output with `R`'s `str` command, as shown below. Each object in the adjacency list (the '`List of 9`' component below) is the vector of neighbours of each cell (graph vertex). The output below can be compared with Figure 4.8.

```
# the structure of a 3x3 (rook's case) grid, as created by the cell2nb command.
str(cell2nb(3,3))


List of 9
 $ : int [1:2] 2 4
 $ : int [1:3] 1 3 5
 $ : int [1:2] 2 6
 $ : int [1:3] 1 5 7
 $ : int [1:4] 2 4 6 8
 $ : int [1:3] 3 5 9
 $ : int [1:2] 4 8
 $ : int [1:3] 5 7 9
 $ : int [1:2] 6 8
 - attr(*, "class")= chr "nb"
 - attr(*, "call")= language cell2nb(nrow = 3, ncol = 3)
 - attr(*, "region.id")= chr [1:9] "1:1" "2:1" "3:1" "1:2" ...
 - attr(*, "cell")= logi TRUE
 - attr(*, "rook")= logi TRUE
 - attr(*, "sym")= logi TRUE
```

Although the Aragonez dataset is essentially associated with a rectangular grid, there are a few missing values. It is therefore not possible to merely use the command `cell2nb(nrow=26, ncol=40)` to define the neighbours for this example.

spdep::dnearneigh creates a list of neighbours based on the scalar Euclidean distances between the specified point coordinates, and so may be appropriate for geostatistical data. The function accepts as input a `SpatialPoints` object with a `coordinates` argument, or a two-column matrix of coordinates from which standard Euclidean distances can be computed. The command creates a list of neighbours, according to the criterion that the distance between the specified coordinates lies between a lower bound `d1` (usually zero, although no default value is supplied) and an upper bound `d2`.

We illustrate the use of this function with the Aragonez data set. Argument `d1` is set to 0. Given that the columns in the Aragonez data are separated by $2.25m$, whereas the center points for adjacent row cells are separated by $3.75m$, setting the argument $d2 = 3$ will only consider as neighbours points that are adjacent on the same row of the trial field. This can be checked using the `plot` method for objects of class `nb`, which produces Figure 4.13.

```
dnearneigh(AragonezPoints, d1=0, d2=3)


Neighbour list object:
Number of regions: 1019
Number of nonzero links: 1958
Percentage nonzero weights: 0.1885664
Average number of links: 1.921492
```

The number of non-zero links (1958) is almost twice the number of points (1019) since, in general each point has two row-wise adjacent neighbours. The difference results from the fact that there are border points with only one neighbour, but also missing values, which are clearly visible in Figure 4.13. The graph in Figure 4.13 is not connected. In general, each row in the trial field is a separate component of the graph, but some rows have more than one separate component. Note that the rows do *not* correspond to the physical wires in the vineyard trellis, which are associated with each column. This is an inadequate choice of neighbours for the Aragonez data set, both due to conceptual reasons (there is no plausible reason why observation in adjacent rows should not be spatially autocorrelated) and (more importantly) because it defies the visual patterns provided by the plots in, for example, Figure 4.7.

Figure 4.9 above was also created with `spdep::dnearneigh` command, but choosing an alternative maximum distance: $d2 = 4$ meters, which connects adjacent points in a way similar to the rook's case: for most points, the neighbours are the four cells immediately above, to the right, below, and to the left. The resulting graph is connected: there is a path between any two vertices (observations).

Choosing $d2 = 5$ gives the plot in Figure 4.14, which connects adjacent grid points in a way similar to the queen's case of Figure 4.11, but with an extra-long horizontal connection. Points that are diagonally adjacent are neighbours, since they are at a distance of $\sqrt{2.25^2 + 3.75^2} = 4.373214$ meters. Points on the same row, but two columns apart, are also neighbours since they are separated by a distance of $4.5m$. The latter feature allows for spatial dependence over gaps in the data, as can be seen in Figure 4.14. Thus, most grid points will have 10 neighbours: one above; two to the right; one below; two to the left; and four more in the diagonal directions. The actual number of non-zero links is slightly less, due to border points and missing values, as can be seen in the following text output.

```
par(cex=0.5, pch=16)
plot(dnearneigh(AragonezPoints, d1=0, d2=3),
        coord=coordinates(AragonezPoints), col="red")
```

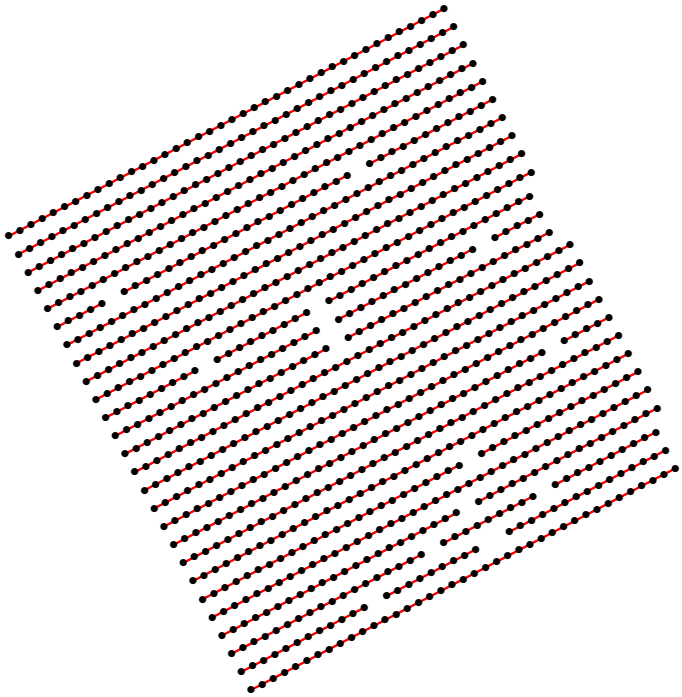


Figure 4.13: Aragonez dataset neighbours, as given by the `spdep::dnearneigh` command, with upper distance bound of 3 meters. Only corresponding points in adjacent columns (which are separated by $2.25m$) are paired up as neighbours.

```
dnearneigh(AragonezPoints, d1=0, d2=5)


Neighbour list object:
Number of regions: 1019
Number of nonzero links: 9550
Percentage nonzero weights: 0.9197187
```

```
Average number of links: 9.371933
```

```
par(cex=0.5, pch=16)
plot(dnearneigh(AragonezPoints, d1=0, d2=5),
          coord=coordinates(AragonezPoints), col="red")
```

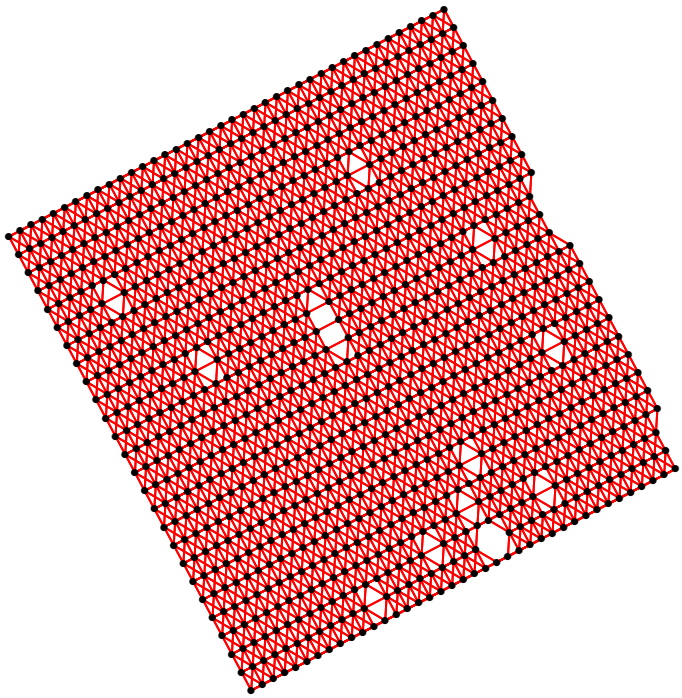


Figure 4.14: Aragonez dataset `dnearneigh` neighbours, with upper distance bound of 5 meters.

The choice of distance bounds implicitly defines at what distance spatial autocorrelation ceases to be important.

The adjacency matrices of graphs associated with the `dnearneigh` command are necessarily symmetric, since $d_{ij} = d_{ji}$, so that $a_{ij} = a_{ji}$. But a subsequent weights matrix may not be symmetric, depending on the way that weights are specified.

spdep::knearneigh whose name originates from the k-nearest neighbour classification methods. The `knearneigh` command accepts as input a set of point coordinates and a value for `k`, the number of neighbours that is to be associated with each point (by default, $k=1$). The `knearneigh` command does not directly produce `nb` objects, but rather objects of class `knn`, originally defined in the `class` package by B. Ripley and W. Venables and previously discussed in Chapter 3. However, the `spdep` package also provides a `knn2nb` function which converts `knn` objects to `nb` objects.

We illustrate the use of these functions on the Aragonez dataset, with argument `k=4`. Notice how the average number of links is exactly 4 (by design). For most points these will mean the adjacent cells, in the rook's case sense, but for borderline points the four nearest neighbours form a more complex pattern, as shown in Figure 4.15.

The adjacency matrices associated with graphs resulting from the k nearest neighbour rule are not, in general symmetric. Asymmetry will be particularly felt for observation points near the borders, or near missing values, in the dataset. It may also be the case that the sets of k neighbours are defined with subjective software-dependent rules. Consider, for example, the case of the Aragonez dataset, if $k=3$ is chosen: for most observations points, there will be two vertices (immediately above, and below, the vertex in question) which share a common third largest distance and are therefore tied for the definition of third nearest neighbour.

```
knn2nb(knearneigh(AragonezPoints, k=4))


Neighbour list object:
Number of regions: 1019
Number of nonzero links: 4076
Percentage nonzero weights: 0.3925417
Average number of links: 4
Non-symmetric neighbours list
```

Since the k nearest neighbour adjacency matrices are, in general, non-symmetric, the corresponding graphs are directed graphs (digraphs). The `plot` method for objects of class `nb`, which is provided by the `spdep` package (see `help(plot.nb)` for details) provides a logical argument called `arrows` which, when set to the logical value `TRUE`, creates a directed graph. However, even for fairly small graphs, such as the one in Figure 4.15, it is hard to read the plot produced by this option.

```
par(cex=0.5, pch=16)
plot(knn2nb(knearneigh(AragonezPoints, k=4)),
                    coordinates(AragonezPoints), col="red")
```

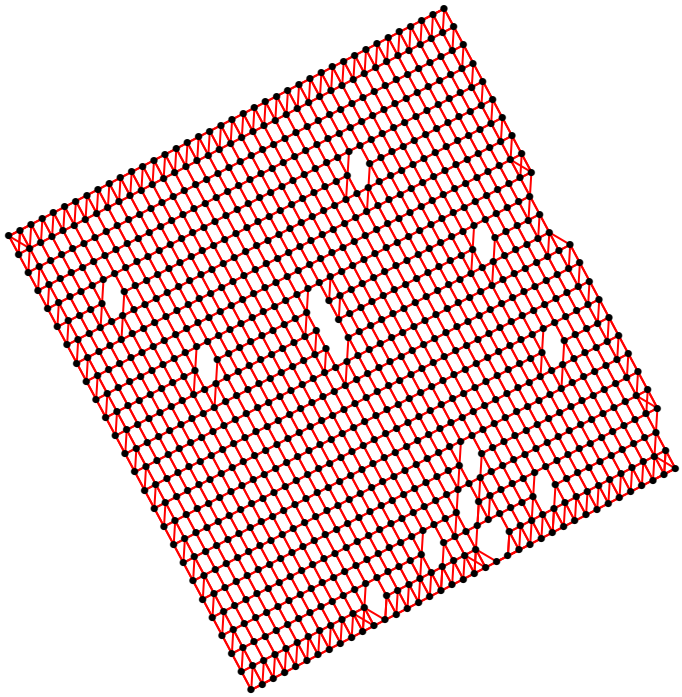


Figure 4.15: Neighbours in the Aragonez dataset, obtained with the `spdep::knearneigh` function, with $k = 4$ neighbours for each point.

spdep::poly2nb is a function that accepts an object of class `SpatialPolygonsDataFrame` as an input argument, and creates a neighbour list (of class `nb`) by pairing up regions with (by default) a queen's case rule for neighbours. In the Aragonez data set, on average, each grid point has almost 8 neighbours (7.503435), as would be expected in a full rectangular grid with the queen's case. The corresponding plot is given in Figure 4.16.

```
poly2nb(AragonezPolygons)


Neighbour list object:
Number of regions: 1019
Number of nonzero links: 7646
Percentage nonzero weights: 0.7363528
Average number of links: 7.503435
```

4.4.7 Weights matrices in R

The `spdep` R package also provides two crucial functions to convert objects of class `nb` to spatial weights matrices or objects that behave like them.

spdep::nb2mat accepts as input an object of class `nb` (produced by the commands in the previous Subsection), and creates a spatial weights matrix. The use of this function is illustrated below, with the 3×3 rectangular grid, and using a rook's case neighbour pattern (the default in the `spdep::cell2nb` function) and a normalized to row-sum weights criterion (the default in the `spdep::nb2mat` function):

```
nb2mat(cell2nb(3,3))


          [,1] [,2]      [,3] [,4]      [,5] [,6]      [,7] [,8]      [,9]
1:1 0.0000000 0.50 0.0000000 0.50 0.0000000 0.00 0.0000000 0.00 0.0000000
2:1 0.3333333 0.00 0.3333333 0.00 0.3333333 0.00 0.0000000 0.00 0.0000000
3:1 0.0000000 0.50 0.0000000 0.00 0.0000000 0.50 0.0000000 0.00 0.0000000
1:2 0.3333333 0.00 0.0000000 0.00 0.3333333 0.00 0.3333333 0.00 0.0000000
2:2 0.0000000 0.25 0.0000000 0.25 0.0000000 0.25 0.0000000 0.25 0.0000000
3:2 0.0000000 0.00 0.3333333 0.00 0.3333333 0.00 0.0000000 0.00 0.3333333
1:3 0.0000000 0.00 0.0000000 0.50 0.0000000 0.00 0.0000000 0.50 0.0000000
2:3 0.0000000 0.00 0.0000000 0.00 0.3333333 0.00 0.3333333 0.00 0.3333333
3:3 0.0000000 0.00 0.0000000 0.00 0.0000000 0.50 0.0000000 0.50 0.0000000
attr(,"call")
nb2mat(neighbours = cell2nb(3, 3))
```

The `nb2mat` function has an argument `style`, which controls the type of weights that are assigned to the neighbour pairs (as discussed above), with the following conventions:

```
par(cex=0.5, pch=16)
plot(poly2nb(AragonezPolygons), coords=coordinates(AragonezPolygons), col="red")
```

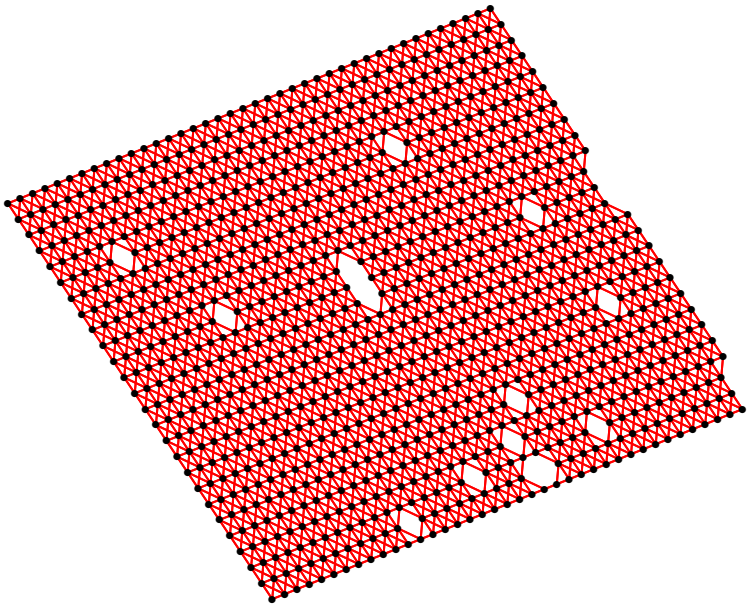


Figure 4.16: Neighbours in the Aragonez dataset, obtained from a `SpatialPolygonsDataFrame` object, using the `spdep::poly2nb` command. The `SpatialPolygonsDataFrame` `AragonezPolygons` object was previously created from the `sf` object `AragonezGrid`.

W (the default) gives a *row-normalized weights matrix*: the weights of each row add to 1.

B denotes a *binary weights matrix*, where all links have weight 1.

C is the *globally standardized by the mean number of links* (edges) weight matrix (the sum of all weights is n, the number of observations).

U is the *globally standardized by the total number of links* (edges) weight matrix (its elements

add to 1).

The globally standardized by the mean number of edges weight matrix for the 3×3 grid of Figure 4.8 is given below. The mean number of edges is $\frac{2\times12}{9} = 2.6666667$ (recall that undirected edges are counted twice), the reciprocal of which is the value of all non-zero matrix entries: 0.375.

```
nb2mat(cell2nb(3,3), style="C")


     [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
1:1 0.000 0.375 0.000 0.375 0.000 0.000 0.000 0.000 0.000
2:1 0.375 0.000 0.375 0.000 0.375 0.000 0.000 0.000 0.000
3:1 0.000 0.375 0.000 0.000 0.000 0.375 0.000 0.000 0.000
1:2 0.375 0.000 0.000 0.000 0.375 0.000 0.375 0.000 0.000
2:2 0.000 0.375 0.000 0.375 0.000 0.375 0.000 0.375 0.000
3:2 0.000 0.000 0.375 0.000 0.375 0.000 0.000 0.000 0.375
1:3 0.000 0.000 0.000 0.375 0.000 0.000 0.000 0.375 0.000
2:3 0.000 0.000 0.000 0.000 0.375 0.000 0.375 0.000 0.375
3:3 0.000 0.000 0.000 0.000 0.000 0.375 0.000 0.375 0.000
attr(,"call")
nb2mat(neighbours = cell2nb(3, 3), style = "C")
```

Spatial weights matrices are usually very large, and tend to be sparse (most points/cells are not assumed to be spatially connected). Thus, *it is advisable to avoid creating the (often extremely large)* $n \times n$ *weights matrices* for n observations of each variable.

spdep::nb2listw The authors of the `spdep` package have incorporated the functionality for sparse matrices from `R`'s `Matrix` package, to create a class `listw` of objects, which efficiently store the necessary information for (sparse) spatial weights matrices. In this class of objects, the first component is an `nb` object specifying the neighbours, a second component is a list of numeric vectors giving the non-zero spatial weights and a third component records the style of weights used. The `nb2listw` command provides the same `style` argument as `nb2mat`, and the default weighting method is again the normalized by row sum (`W`) method, as can be seen by applying the `nb2listw` command to the 3×3 cell grid:

```
nb2listw(cell2nb(3,3))


Characteristics of weights list object:
```

```
Neighbour list object:
Number of regions: 9
Number of nonzero links: 24
Percentage nonzero weights: 29.62963
Average number of links: 2.666667

Weights style: W
Weights constants summary:
  n nn S0       S1       S2
W 9 81  9 6.916667 36.80556
```

The five `weights constants summary` values that appear at the end of the output are, respectively:

n - the number n of observations (sample size);

nn - the number n^2 of elements in the $n \times n$ weight matrix;

$S0$ - the sum of all the weights in the weights matrix:

$$S_0 = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \ . \tag{4.11}$$

$S1$ - twice the sum of squares of all elements in the *symmetric part* of the weights matrix $\mathbf{W}$, which is defined as $\frac{\mathbf{W}+\mathbf{W}^t}{2}$. A symmetric matrix is equal to its symmetric part, so if $\mathbf{W}$ is symmetric, $\frac{\mathbf{W}+\mathbf{W}^t}{2} = \mathbf{W}$. In general, we have:

$$S_1 = 2\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{w_{ij}+w_{ji}}{2}\right)^2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(w_{ij}+w_{ji})^2 \ . \tag{4.12}$$

$S2$ - If $w_{i.} = \sum_{j=1}^{n} w_{ij}$ is the row sum of $\mathbf{W}$'s i-th row, and $w_{.i} = \sum_{j=1}^{n} w_{ji}$ is the column sum of $\mathbf{W}$'s i-th column, we have:

$$S_2 = \sum_{i=1}^{n}(w_{i.}+w_{.i})^2 \ . \tag{4.13}$$

Below is a `listw` object for the `nb` neighbour list obtained from the `AragonezPolygons` object, using the globally standardized by the total number of edges style:

```
nb2listw(poly2nb(AragonezPolygons), style="U")

Characteristics of weights list object:
Neighbour list object:
Number of regions: 1019
Number of nonzero links: 7646
Percentage nonzero weights: 0.7363528
Average number of links: 7.503435

Weights style: U
Weights constants summary:
     n      nn S0           S1          S2
U 1019 1038361  1 0.0002615747 0.004005657
```

We now store some of the spatial weights matrices created above, for future reference.

```
Wd3 <- nb2listw(dnearneigh(AragonezPoints, d1=0, d2=3))
Wd5 <- nb2listw(dnearneigh(AragonezPoints, d1=0, d2=5))
print(Wd5)

Characteristics of weights list object:
Neighbour list object:
Number of regions: 1019
Number of nonzero links: 9550
Percentage nonzero weights: 0.9197187
Average number of links: 9.371933

Weights style: W
Weights constants summary:
     n      nn   S0       S1       S2
W 1019 1038361 1019 221.4423 4083.586

WBd5 <- nb2listw(dnearneigh(AragonezPoints, d1=0, d2=5), style="B")
WBk4 <- nb2listw(knn2nb(knearneigh(AragonezPoints, k=4)), style="B")
WCp <- nb2listw(poly2nb(AragonezPolygons), style="C")
```

We focus next on numerical indicators that measure the degree of spatial autocorrelation.

4.5 Moran's I and Geary's c

In this Section we assume that there is a random sample $(Z_1, Z_2, ..., Z_n)$ of a fully numerical spatial process Z, as is the case with the Aragonez dataset yields. We also assume that a *spatial weights matrix* $\mathbf{W}$ has been defined (as discussed in Section 4.4). Each matrix element w_{ij} measures the intensity of the effect of observation Z_j on observation Z_i.

Probably the most frequent measure of spatial autocorrelation is Moran's I indicator, which was originally developed in the 1950's to test the null hypothesis of zero autocorrelation for the (fully numerical) random process Z. As a starting point for this indicator, we consider the following expression:

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(Z_i - \overline{Z})(Z_j - \overline{Z})}{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}} \tag{4.14}$$

Expression (4.14) resembles a weighted covariance, not between different variables measured at corresponding points, but between the values of the same variable (the sample values Z_i), measured at all possible pairs of points. The sum of the weights in the denominator is $S_0 = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}$ (eq. 4.11).

Moran's I indicator compares this 'Moran covariance' with the value that would result if the spatial weights matrix were an identity matrix ($\mathbf{W} = \mathbf{I}$), so that $w_{ij} = 1$ for $i = j$ and $w_{ij} = 0$ if $i \neq j$. This pseudo-'weight matrix' $\mathbf{I}$ assumes that value Z_i is determined only by itself, and by no other value, which is what we would expect with independent observations. In a sense, Moran's I is measuring how well the spatial weights w_{ij} applied to neighbouring values Z_j are capable of reconstituting the observed values Z_i.

Moran's I is therefore defined as the ratio:

$$I = \frac{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(Z_i-\overline{Z})(Z_j-\overline{Z})}{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}}}{\frac{\sum_{i=1}^{n}(Z_i-\overline{Z})^2}{n}} = \frac{n}{S_0}\cdot\frac{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(Z_i - \overline{Z})(Z_j - \overline{Z})}{\sum_{i=1}^{n}(Z_i - \overline{Z})^2}\,. \tag{4.15}$$

The contribution of the i-th observation to the value of Moran's I is sometimes used as a measure of *local* spatial autocorrelation. More specifically, a *local Moran's* I_i is defined so that $I = \frac{\sum_{i=1}^{n} I_i}{S_0}$, with I_i given by:

$$I_i \;=\; n \cdot \frac{(Z_i - \overline{Z}) \sum_{j=1}^{n} w_{ij}(Z_j - \overline{Z})}{\sum_{i=1}^{n} (Z_i - \overline{Z})^2} \;. \tag{4.16}$$

The behaviour of Moran's I indicator is not entirely trivial. The expected value of I in the absence of spatial autocorrelation is not zero, but $\frac{-1}{n-1}$ (see Plant [2], 2012, for the Cliff and Ord, 1981, reference). Larger values of I are associated with positive autocorrelation, and smaller values of I suggest negative autocorrelation.

Geary's c is a somewhat related indicator, which instead of using 'Moran's covariance', uses a weighted sum of the squared distances between the observed variable values at all possible pairs of observed points:

$$c \;=\; \frac{n-1}{2\,S_0} \cdot \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(Z_i - Z_j)^2}{\sum_{i=1}^{n} (Z_i - \overline{Z})^2} \;. \tag{4.17}$$

The expected value of Geary's c in the absence of spatial autocorrelation is 1. Smaller values (necessarily non-negative) indicate *positive* autocorrelation, and values $c > 1$ indicate negative autocorrelation.

It must be stressed that both indicators depend on a given spatial weights matrix, and so on a given assumption about the relevant spatial effects. A noteworthy fact is that, given the definition of both I and c, the value of each indicator remains the same if a non-symmetric weight matrix $\mathbf{W}$ is replaced by its symmetric part, that is, by the symmetric matrix $\frac{\mathbf{W}+\mathbf{W}^t}{2}$ (which replaces both w_{ij} and w_{ji} with their mean value).

Both Moran's I and Geary's c have been used to *test the null hypothesis of no spatial autocorrelation*. It has been proven that, for Normally distributed variables, both indicators have asymptotic Normal distribution, given the null hypothesis of no spatial autocorrelation. But the variance of I (or c) in this asymptotically Normal distribution can be computed in two different ways, depending on the additional assumptions that are made:

- The *resampling* option: we make the standard assumption that the sample was chosen at random, and a new sample would give different values.
- Alternatively, the *randomisation* option: we can work conditionally on the observed

values of the variables and merely assume that those values could be reassigned at random to different spatial locations.

In both cases, the test statistics are the normalized indicator, that is, in the case of Moran's I, $\frac{I-E[I]}{\sqrt{V[I]}}$. For moderate or large samples, it can be safely assumed that this statistic has a Normal distribution. The derivation of the test assumes that the weights matrix is symmetric, but replacing a non-symmetric weights matrix $\mathbf{W}$ with its symmetric part leaves the value of I intact (as was seen above) and so this is not a restrictive option.

The `R` package `spdep` provides a useful suite of functions written by Roger Bivand. The functions `spdep::moran` and `spdep::geary` compute the value of each indicator. The functions `spdep::moran.test` and `spdep::geary.test` perform tests for the absence of spatial autocorrelation (the null hypothesis). Both the `moran.test` and the `geary.test` functions will, by default, carry out the test with the randomisation-based variance. If the `randomisation` argument is set to the logical value `FALSE`, the resampling-based variance is used.

An alternative function, called `spdep::lm.morantest`, caters for Moran's I in the case of residuals from a linear regression which are not, by construction, independent. This will the case whenever detrending has been done via a regression. Asymptotic normality and resampling are assumed.

Alternatively (and in particular for smaller samples where asymptotic Normality is questionable), the function `spdep::moran.mc` performs a permutations-based test that does not assume asymptotic Normality. In these permutation tests, the variable values are re-assigned at random to the different spatial locations, a large number of times. For each re-assignmment, the value of the Moran indicator is computed and empirical quantiles are calculated for this large set of reassignment-based values of I or c. The empirical quantile of our true indicator value is registered. In the absence of spatial autocorrelation, these true empirical quantiles of I or c would not be expected to be extreme. If they are, this suggests the existence of spatial autocorrelation. The number of permutations must be specified when invoking the `moran.mc` command, through the `nsim` argument.

Since the values of Moran's I and of Geary's c are also displayed when using the test functions, we illustrate the use of these functions, with the Aragonez dataset, for the weights matrices computed in Subsection 4.4.7, and the two variants of detrended yields discussed in Subsection 5.5.

First, we consider the value of Moran's I, for `yieldct`, that is the yields detrended by simply subtracting the mean value. We begin with the `Wd5` spatial weights matrix, which gives the row-normalized weights for neighbours defined as points at a distance of up to $5m$. By

default, the test carried out by the function is a one-sided hypothesis test, testing the null hypothesis of no autocorrelation against the alternative that there is *positive* autocorrelation. This option can be changed through the argument `alternative`.

```
moran.test(AragonezPoints$yieldct, listw=Wd5)



Moran I test under randomisation

data:  AragonezPoints$yieldct
weights: Wd5

Moran I statistic standard deviate = 22.221, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     0.3218883674     -0.0009823183      0.0002111239
```

The positive, somewhat large, value of $I = 0.3218884$ is considered highly significant assuming asymptotic Normality with the randomisation option for the variance ($p < 2.2 \times 10^{-16}$, that is, less than machine precision and so indistinguishable from zero). This means a very clear rejection of the independence null hypothesis, in favour of the alternative hypothesis that positive spatial autocorrelation exists. This is entirely coherent with what was observed in the plots.

The expected value of I under independence, which is also show in the output ($E[I] = \frac{-1}{n-1} = -9.8231827 \times 10^{-4}$), is therefore considered significantly smaller than the calculated value, 0.3218884. It should be noted that this is the same value of I that would be obtained if the original variable `yield` were invoked, since the nature of Moran's I involves a subtraction of the mean. Thus, for a constant mean trend, there is considerable evidence of positive spatial autocorrelation. But, as noted previously, an undetected underlying deterministic spatial trend may be confused with spatial autocorrelation. It may be the case that a different deterministic trend (with different detrended variable values) is compatible with the absence of spatial autocorrelation. We now check the performance of the linearly detrended variable `yielddt`.

```
moran.test(AragonezPoints$yielddt, listw=Wd5)
```

```
Moran I test under randomisation

data:  AragonezPoints$yieldldt
weights: Wd5

Moran I statistic standard deviate = 11.375, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     0.1643071995     -0.0009823183      0.0002111388
```

The value of Moran's I is now noticeably smaller than before ($I = 0.1643072$), but the corresponding p-value for the null hypothesis of no spatial autocorrelation is still indistinguishable from zero, so there is still strong indication of spatial autocorrelation.

The `spdep::lm.morantest` function is better suited in this case, since the `yieldldt` values are residuals from a linear regression of the variable `yield` on the linear predictors of row and column positions. The first argument to the `lm.morantest` function is the original linear regression whose residuals are to be tested for spatial autocorrelation. In our example, the significance of the test result is practically indistinguishable from that obtained with the `moran.test` function, although it can be observed that the expected value and variance of I are now different.

```
lm.morantest(lm(yield ~ rowm + colm, data=AragonezPoints), listw=Wd5)


Global Moran I for regression residuals

data:
model: lm(formula = yield ~ rowm + colm, data = AragonezPoints)
weights: Wd5

Moran I statistic standard deviate = 11.588, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Observed Moran I      Expectation         Variance
    0.1643071995    -0.0029377790     0.0002082996
```

Finally, we illustrate the results of the permutations-based test (which does not assume asymptotic Normality of the test statistic), again for the linearly detrended yields. The value of Moran's I produced by the function `sp::moran.mc` is the same, but its associated p-value is computed through the empirical quantiles associated with the permutations.

```
moran.mc(AragonezPoints$yieldldt, listw=Wd5, nsim=10000)



Monte-Carlo simulation of Moran I

data:  AragonezPoints$yieldldt
weights: Wd5
number of simulations + 1: 10001

statistic = 0.16431, observed rank = 10001, p-value = 9.999e-05
alternative hypothesis: greater
```

The use of Geary's c gives similar results, keeping in mind that the absence of spatial autocorrelation is indicated by the value $c = 1$ and that positive autocorrelation corresponds to values $0 < c < 1$. Note, however, that the author of the `geary.test` function code has multiplied the test statistic by minus 1, in other words, the test statistic is $\frac{E[c]-c}{\sqrt{V[c]}}$. Therefore, the default value of the `alternative` argument (`"greater"`) also means *positive* autocorrelation.

```
geary.test(AragonezPoints$yieldct, listw=Wd5)



Geary C test under randomisation

data:  AragonezPoints$yieldct
weights: Wd5

Geary C statistic standard deviate = 21.853, p-value < 2.2e-16
alternative hypothesis: Expectation greater than statistic
sample estimates:
Geary C statistic       Expectation          Variance
     0.6787219164      1.0000000000      0.0002161427
```

```
geary.test(AragonezPoints$yieldldt, listw=Wd5)



Geary C test under randomisation

data:  AragonezPoints$yieldldt
weights: Wd5

Geary C statistic standard deviate = 11.208, p-value < 2.2e-16
alternative hypothesis: Expectation greater than statistic
sample estimates:
Geary C statistic       Expectation          Variance
     0.8352664464      1.0000000000      0.0002160266
```

The use of Normality-based tests does not produce major differences. This is illustrated below for the case of Moran's I. Note that the value of I does not change with the type of test used to judge its significance.

```
moran.test(AragonezPoints$yieldct, listw=Wd5, randomisation=F)



Moran I test under normality

data:  AragonezPoints$yieldct
weights: Wd5

Moran I statistic standard deviate = 22.21, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     0.3218883674     -0.0009823183      0.0002113264

moran.test(AragonezPoints$yieldldt, listw=Wd5, randomisation=F)



Moran I test under normality
```

```
data:  AragonezPoints$yieldldt
weights: Wd5

Moran I statistic standard deviate = 11.37, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     0.1643071995     -0.0009823183      0.0002113264
```

These tests are naturally affected by the specific spatial weights matrix that is used. For the Aragonez data set, the overall conclusion that spatial autocorrelation exists seems fairly robust, even for the `yieldldt` linearly detrended yields. This is illustrated below, for Moran's I and using permutation tests, with (i) the binary weights matrix using the maximum distance of $5m$ to define pairs of neighbours (`WBd5`); and (ii) the globally standardized by the mean number of edges (links) weight matrix, based on the `poly2nb` definition of neighbours (`WCp`).

```
moran.test(AragonezPoints$yieldldt, listw=WBd5)


Moran I test under randomisation

data:  AragonezPoints$yieldldt
weights: WBd5

Moran I statistic standard deviate = 11.351, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     0.1624319200     -0.0009823183      0.0002072468


moran.test(AragonezPolygons$yieldldt, listw=WCp)


Moran I test under randomisation

data:  AragonezPolygons$yieldldt
```

```
weights: WCp

Moran I statistic standard deviate = 9.8516, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     0.1576682256     -0.0009823183      0.0002593378
```

4.6 K-th order neighbours and Moran's correlogram

If a list of (first-order) neighbours for each observation has been established, the concept of *kth-order* neighbours can be defined, as was discussed in Subsection 4.4.4. This concept proves useful in determining how indicators such as Moran's I vary as successive orders of neighbours are used to compute I, thereby providing information about the way in which autocorrelation effects evolve over increasingly distant sets of observations.

Given a neighbour's list, the function `spdep::nblag` computes the neighbours of successive order, up to a value k provided by the `maxlag` argument. The use of this function is illustrated below, for the `yieldldt` variable in the Aragonez dataset, using the 4 nearest neighbours list computed with the `knearneigh` function (do not confuse the `k=4` argument in this function with the concept of k-th order neighbour, which is specified by the `maxlag` argument).

```
nb.k4 <- knn2nb(knearneigh(AragonezPoints, k=4))
nblag(nb.k4, maxlag=3)


[[1]]
Neighbour list object:
Number of regions: 1019
Number of nonzero links: 4076
Percentage nonzero weights: 0.3925417
Average number of links: 4
Non-symmetric neighbours list

[[2]]
Neighbour list object:
Number of regions: 1019
Number of nonzero links: 7742
```

```
Percentage nonzero weights: 0.7455981
Average number of links: 7.597645
Non-symmetric neighbours list

[[3]]
Neighbour list object:
Number of regions: 1019
Number of nonzero links: 11217
Percentage nonzero weights: 1.08026
Average number of links: 11.00785
Non-symmetric neighbours list

attr(,"call")
nblag(neighbours = nb.k4, maxlag = 3)
```

The output of the `nblag` function is a list of length `maxlag`, which indicates the summary characteristics of the neighbour's list for each order (lag). Thus, the first object in the output list summarizes the first-order neighbours list (the output is identical to that of `nb.k4`). The second list object summarizes the neighbours of order 2: there are in all 7728 second-order neighbours (at a distance 2 in the neighbours' graph), of the $n = 1019$ cells/points in our rectangular grid, for an average of $\frac{7728}{1019} = 7.5839058$ edges per vertex (links per cell), which means that $0.0074425 \times 100\%$ of the 1019^2 possible links are actually established in this second-order neighbour relation. Likewise, the third-order neighbours connect slightly over 1% of all possible pairs of cells/points.

The function `spdep::sp.correlogram` computes Moran's I for the neighbours of each successive order, when the `method="I"` argument value is used. Other arguments which must be specified are the original neighbours list (an object of class `nb`), the variable for which the I indicator is to be calculated (argument `var`) and the order up to which neighbours are to be computed (argument `order`). The `style` of the weight matrix may be specified. By default it is `style="W"`, that is, a row-normalized weights matrix. Here are the results for the `yieldldt` linearly detrended yields, with the `nb.k4` neighbours specified above ($k = 4$ nearest neighbours for each point), with the default weights matrix of style `W`:

```
sp.correlogram(nb.k4, var=AragonezPoints$yieldldt, method="I", order=3)


Spatial correlogram for AragonezPoints$yieldldt
method: Moran's I
```

```
            estimate expectation    variance standard deviate
1 (1019)  0.19952493 -0.00098232  0.00047807           9.1703
2 (1019)  0.12454917 -0.00098232  0.00024658           7.9942
3 (1019)  0.10268912 -0.00098232  0.00016876           7.9803
         Pr(I) two sided
1 (1019)       < 2.2e-16 ***
2 (1019)       1.305e-15 ***
3 (1019)       1.460e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Each row in the output corresponds to the output of a `moran.test` function with the same arguments, but with neighbours of order k in the k-th row. As would be expected, the value of Moran's I decreases as the order k of neighbours grows, in other words, spatial correlation tends to be stronger for smaller spatial lags.

Plotting the values of Moran's I against the order k of the neighbours produces a *Moran's correlogram*. It is easy to request a plot of Moran's correlogram, since there is a *plot method* for the output of the function `sp.correlogram` (which is of class `spcor`). Figure 4.17 shows a Moran's correlogram of order up to 10.

Although Moran's I falls sharply after lag 1, there is evidence of spatial autocorrelation for neighbours of up to about lag $k=6$. The largest lag for which the Moran's randomisation test (the default test for the `sp.correlogram` function) would reject the null hypothesis of no spatial autocorrelation (for a significance level $\alpha=0.05$) is $k=7$.

If significant spatial autocorrelation exists for lags larger than 1, it may be advisable to redefine the first-order neighbours, so as to include the relevant neighbours of higher order. This can be done using the `spdep::nblag_cumul` function. The `nblag_cumul` command accepts the output from an `nblag` command, as illustrated below.

```
nb.k4lg6 <- nblag_cumul(nblag(nb.k4, maxlag=6))
nb.k4lg6


Neighbour list object:
Number of regions: 1019
Number of nonzero links: 75326
Percentage nonzero weights: 7.254317
```

```
plot(sp.correlogram(nb.k4, var=AragonezPoints$yieldldt, method="I", order=10))
```

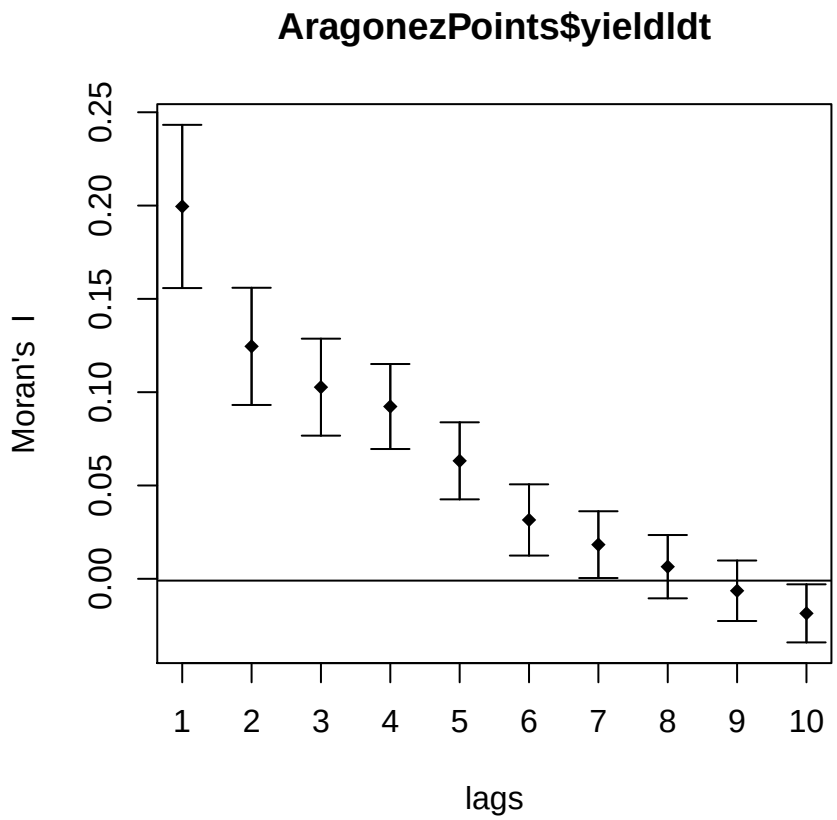


Figure 4.17: Moran's correlogram for the linearly detrended Aragonez yields, based on a $k = 4$ nearest neighbours list and a row-normalized weight matrix, with lags of up to 10. This Moran's correlogram was produced by the function `sp.correlogram`. Vertical bars indicate intervals that extend to two standard deviations from the mean, in each direction.

```
Average number of links: 73.92149
Non-symmetric neighbours list
```

The argument value `method="C"` gives similar results, using Geary's "c" indicator.

4.7 Variograms and related tools

For geostatistical data, that is, variables $Z(s)$ varying continuously over some space $\mathcal{S}$, the (semi-)variogram is an important tool in assessing spatial patterns.

4.7.1 Covariograms, variograms and semi-variograms

We begin by introducing some concepts. We assume that $Z(s)$ is a fully numerical **random spatial process** where $s \in \mathcal{S}$. Note that s is a vector. We define:

the mean function $\boldsymbol{\mu_s}$ as the function that, for each location $s \in \mathcal{S}$ gives the expected value $\mu_s = E[Z(s)]$.

the covariogram $\boldsymbol{C(s_1, s_2)}$, or **auto-covariance function**, is the function that, for any pair of locations $s_1, s_2 \in \mathcal{S}$, gives the covariance between $Z(s_1)$ and $Z(s_2)$:

$$C(s_1, s_2) = Cov[Z(s_1), Z(s_2)] = E[(Z(s_1) - \mu_{s_1})(Z(s_2) - \mu_{s_2})] \ . \tag{4.18}$$

We define the following terms, associated with a random spatial process $Z(s)$:

spatial lag $\boldsymbol{s_1 - s_2}$ is the difference between two locations s_1 and s_2 where $Z(s)$ is observed. It should be noted that the spatial lag is usually a 2-dimensional (or 3-dimensional, depending on the nature of the space $\mathcal{S}$) vector.

second-order (or weakly) stationary if μ_s does not depend on the location s (is constant over $\mathcal{S}$) and $C(s_1, s_2)$ depends only on the *spatial lag*, that is (and simplifying notation):

$$\mu_s = \mu \ , \quad \forall\, s \in \mathcal{S} \quad ; \quad \text{and} \tag{4.19}$$

$$C(s_1, s_2) = C_\ell(s_1 - s_2) \ , \quad \forall\, s_1, s_2 \in \mathcal{S}. \tag{4.20}$$

isotropic when the covariogram $C(s_1, s_2)$ depends only on the *scalar distance* between the points s_1 and s_2, and not on the precise direction in which that distance occurs: $C(s_1, s_2) = C_s(d(s_1, s_2))$. Thus, an isotropic process necessarily satisfies the covariogram condition for second-order stationarity, although the converse is not true. **Anisotropic** usually denotes a second-order stationary process which does *not* have isotropy, that is, for which the covariogram $C(s_1, s_2)$ only depends on the spatial lag, but in ways that vary according to the direction of the spatial lag vector $s_1 - s_2$.

An extremely useful concept is the **variogram**, $\boldsymbol{2\gamma(s_1, s_2)}$. It is defined as follows.

variogram is the function

$$2\gamma(s_1, s_2) \;=\; V[Z(s_1) - Z(s_2)] \;. \tag{4.21}$$

semi-variogram is the function

$$\gamma(s_1, s_2) \;=\; \frac{1}{2}\, V[Z(s_1) - Z(s_2)] \;. \tag{4.22}$$

The semi-variogram is often (as in some `R` packages) just called a variogram, which may be confusing. The reason for the constant 2 has to do with the relation between the variogram and the previously defined covariogram, in particular for second-order stationary processes $Z(s)$, for which:

$$\begin{aligned} 2\gamma(s_1, s_2) \;&=\; V[Z(s_1) - Z(s_2)] \;=\; V[Z(s_1)] + V[Z(s_2)] - 2\, Cov[Z(s_1), Z(s_2)] \\ &=\; C(s_1, s_1) + C(s_2, s_2) - 2\, C(s_1, s_2) \;. \end{aligned}$$

In the case of a second-order stationary process, $C(s_1, s_1) = C(s_2, s_2) = C_\ell(0)$, and $C(s_1, s_2) = C_\ell(d)$, where $d = s_1 - s_2$ denotes the spatial lag vector. Thus, *for second-order stationary processes* $Z(s)$, the semi-variogram is simply:

$$\gamma(d) \;=\; C_\ell(0) - C_\ell(d) \;. \tag{4.23}$$

With the further assumption of **isotropy**, *the semi-variogram becomes a function of a single real variable, the scalar distance* $d = d(s_1 - s_2)$ associated with the spatial lag:

$$\gamma_s(d) \;=\; C_s(0) - C_s(d) \;. \tag{4.24}$$

4.7.2 Properties of the semi-variogram

Here are some of the properties of the semi-variogram function, with special emphasis on the case of *isotropy*.

- By definition, the semi-variogram is **nonnegative**: $\gamma(s_1, s_2) \geq 0$, for any pair of locations s_1, s_2. This property carries over to the isotropic version: $\gamma_s(d) \geq 0$, $\forall\, d$.
- By definition, for any process $Z(s)$ the semi-variogram is a **symmetric** function: $\gamma(s_1, s_2) = \gamma(s_2, s_1)$, $\forall\, s_1, s_2$. In the case of an *isotropic* process, the semi-variogram is an even function: $\gamma_s(d) = \gamma_s(-d)$, which implies that only positive spatial lags, $d > 0$, need to be considered.

- $\boldsymbol{\gamma(s,s)=0}$ necessarily flows from the definition of a semi-variogram. For *isotropic* processes, this implies that $\boldsymbol{\gamma_s(0)=0}$.

- *In the absence of spatial autocorrelation*, $C(s_1,s_2)=0$, whenever $s_1 \neq s_2$. Hence, the semi-variogram becomes $\gamma(s_1,s_2)=\frac{1}{2}[C(s_1,s_1)+C(s_2,s_2)]$. With second order stationarity, this is just the constant variance $V[Z(s)]$. In the case of isotropy, we have $\gamma_s(d)=C_s(0)=V[Z(s)]$, $\forall\, d \neq 0$. Note that, *in the absence of spatial autocorrelation, the graph of the semi-variogram γ_s is a horizontal line at height $C_s(0)$, with a discontinuity at the origin*, as can be seen in the left plot of Figure 4.19.

- *The variogram is not, in general, continuous at the origin.* This may be thought of as a feature of the semi-variogram itself, or as a consequence of the necessary discretization that any measurement of the covariances underlying the semi-variogram necessarily imply, in practical terms. In the case of isotropic processes, $\boldsymbol{\lim_{d\to 0}\gamma_s(d)=c_0}$ is called the **nugget effect**. The nugget effect can be viewed as the part of the variance of the random process $Z(s)$ that has *not* been explained by the spatial autocorrelation process.

- Since the effect of spatial autocorrelation drops off as observation points are further apart, it is reasonable to assume that the covariance $C(s_1,s_2)$ tends to zero, as the distance between observations tends to infinity. Thus, in an isotropic process, $\boldsymbol{\lim_{d\to+\infty}\gamma_s(d)=C_s(0)=V[Z(s)]}$, the constant variance of the second-order stationary process $Z(s)$. This limiting value is called the **sill** of the semi-variogram.

- the **range** r of a spatial process $Z(s)$ is loosely defined to be the size of the region of the space $\mathcal{S}$ for which spatial correlation effects are felt. In the case of an isotropic stationary process, a rigorous definition for the range is the *largest value of d for which the semi-variogram is smaller than the sill*, $\gamma(d)<V[Z(s)]$. For semi-variograms, in which the sill is an unattained asymptotic value, the range is often defined to be the distance d for which the semi-variogram becomes some proportion, very close to 1 (say 95%), of the sill.

- the difference between the sill and the nugget is the **partial sill**, p. It can be viewed as that part of $V[Z(s)]$ that *is* explained by the spatial autocorrelation process.

The notions of sill, range and nugget for a generic semi-variogram are shown in Figure 4.18.

If, as seen above, the semi-variogram γ_s for data without spatial autocorrelation is a horizontal line at height $C_s(0)$, with a discontinuity at the origin, the typical semi-variogram, in

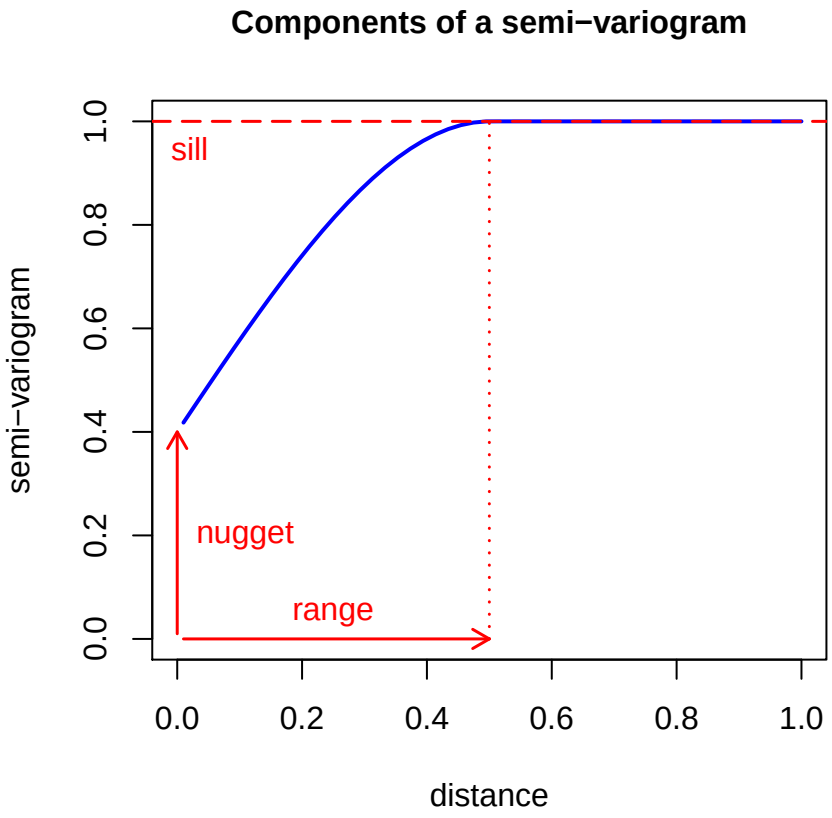


Figure 4.18: Definitions of sill, nugget and range in semi-variograms. The partial sill is the difference between the sill and the nugget.

the case of isotropic spatially correlated processes, is a non-decreasing curve, contained in the horizontal interval defined by the nugget and the sill, as shown in the plot on the right of Figure 4.19. The nature of the underlying points in the plots will be described in Subsection 4.7.3, and the way in which the curve was obtained and plotted is described in Subsection 4.7.4.

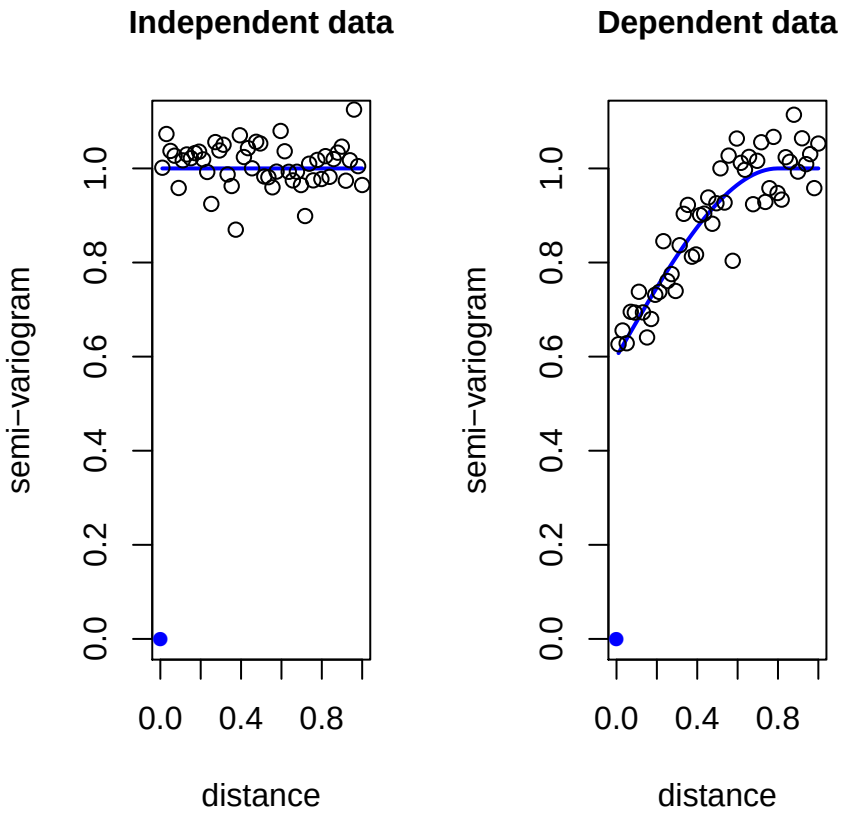


Figure 4.19: Semi-variogram patterns. On the left, a semi-variogram for independent spatial data. On the right, a semi-variogram for data with spatial correlation.

4.7.3 Empirical variograms

In practice, and assuming isotropy, the semi-variogram is estimated by the **empirical semi-variogram**, from the sample $(z(s_1), z(s_2), ..., z(s_n))$:

$$\hat{\gamma}(d) \;=\; \frac{1}{2}\,\frac{1}{|Neigh(d)|}\sum_{(i,j)\in Neigh(d)} (z(s_i) - z(s_j))^2 \;, \tag{4.25}$$

where, for any given distance $d = dist(s_1, s_2)$, $Neigh(d)$ denotes the set of pairs of locations s_1, s_2 which are the given distance d, $|Neigh(d)|$ is the cardinality (size) of this set, and the summation is over all pairs of locations s_i, s_j at that given distance. Usually, and to

ensure the existence of enough pairs $Neigh(d)$ of observations, instead of considering a single distance d, the range of distances is partitioned into small intervals (bins) $[d_{min}, d_{max}]$, and for each interval, we consider the pairs (i, j) such that $d_{min} < dist(s_i, s_j) < d_{max}$. To interpret the empirical semi-variogram, we must consider the properties of the semi-variogram which it is estimating.

Several packages in `R` provide commands to compute empirical semi-variograms. Among them, the `gstat` package, co-authored and maintained by Edzer Pebesma, and the `geoR` package, co-authored and maintained by Paulo J. Ribeiro Jr.

`gstat` package The `gstat::variogram` function computes the empirical semi-variogram, accepting as input arguments a formula to detrend the variable (similar to the `R` formulas for linear regression), and a `SpatialPointsDataFrame` or `sf` object. Alternatively, the latter argument may be replaced by the name of the data frame containing the variable and a list of coordinates for each observed point. This command is invoked here to compute the empirical semi-variogram of the Aragonez variable `yieldct` (centred yields):

```
variogram(yield ~ 1, data=AragonezPoints)
```

```
      np      dist     gamma dir.hor dir.ver   id
1   1944  3.026404 0.8617851       0       0 var1
2   6513  5.666938 0.9560390       0       0 var1
3  13187  9.613532 0.9693001       0       0 var1
4  14887 13.512151 1.0027140       0       0 var1
5  20259 17.441649 1.0266800       0       0 var1
6  20529 21.302718 1.0582606       0       0 var1
7  24687 25.061267 1.0702769       0       0 var1
8  28165 29.142116 1.1049442       0       0 var1
9  26756 33.097528 1.1325510       0       0 var1
10 28621 36.892389 1.1600034       0       0 var1
11 29117 40.763906 1.1997004       0       0 var1
12 29146 44.621526 1.2327257       0       0 var1
13 28860 48.412351 1.2727498       0       0 var1
14 29956 52.323021 1.3368361       0       0 var1
15 27872 56.240912 1.3874178       0       0 var1
```

The `dist` column indicates the distances d between observed points, and the column `gamma` gives the corresponding value of the semi-variogram value $\gamma(d)$, based on the empirical value

computed from the `np` available points. This empirical semi-variogram can be plotted by enclosing the previous command inside a `plot()` call. This is possible because an appropriate `plot` method has been provided by the `R` package `gstat` for objects of class `gstatVariogram`, which is the class of the output objects from the `variogram` command.

```
plot(variogram(yield ~ 1, data=AragonezPoints), pch=16)
```

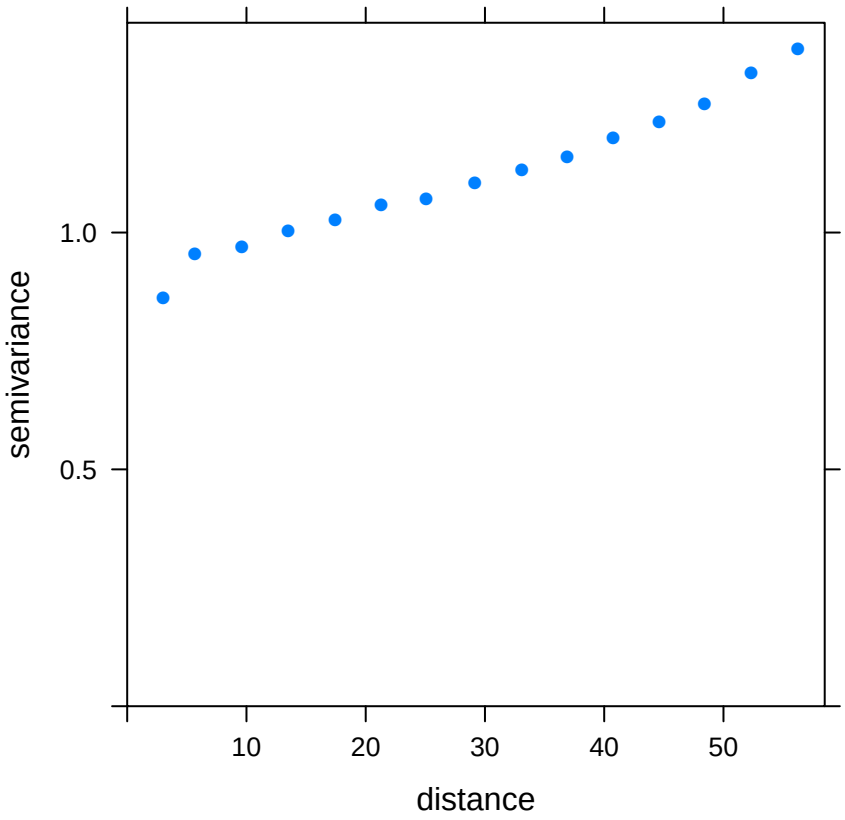


Figure 4.20: The plot of the empirical semi-variogram for the Aragonez yields, detrended by the (constant) mean, as produced by the `variogram` command in package `gstat`.

In Figure 4.20 it is not apparent that the semi-variogram has reached the sill, and therefore the range is also unclear. The nugget effect appears to be close to 0.85. To check whether the curve is approaching a horizontal asymptote, the `cutoff` argument in the command `variogram` will be set to a larger value ($75m$), as illustrated below.

```
variogram(yield ~ 1, data=AragonezPoints, cutoff=75)


      np      dist     gamma dir.hor dir.ver   id
1   4775  3.885159 0.9006839       0       0 var1
2  11662  8.016826 0.9640507       0       0 var1
3  17587 12.546920 0.9995182       0       0 var1
4  27227 17.609841 1.0286987       0       0 var1
5  27150 22.604166 1.0598021       0       0 var1
6  32791 27.481308 1.0921583       0       0 var1
7  35735 32.444811 1.1267466       0       0 var1
8  39019 37.578379 1.1697366       0       0 var1
9  33814 42.471293 1.2151499       0       0 var1
10 42088 47.444342 1.2590447       0       0 var1
11 34834 52.536738 1.3352649       0       0 var1
12 36149 57.389630 1.4167549       0       0 var1
13 35044 62.448067 1.4409542       0       0 var1
14 31021 67.475473 1.5638226       0       0 var1
15 26936 72.392091 1.6330283       0       0 var1
```

The resulting empirical semi-variogram is plotted in Figure 4.21.

Increasing the `cutoff` argument has its drawbacks: as the distance d grows, the values of γ will be estimated with fewer points, becoming prone to erratic behaviour if the number of pairs becomes very small. For the 75 cutoff value chosen above, the estimated sill appears to be larger than 1.7, but it is still unlikely that an asymptotic stabilization has been achieved. This could be the result, either of a non-stationary variance in the process, or of an inappropriately removed underlying trend (see Plant, [2]).

The authors of the `gstat::variogram` provide the possibility of removing a deterministic trend directly in this command. To filter out a linear trend along the coordinates, as was done to create variable `yieldldt` in the Aragonez data, a linear regression on column and row distances (`AragonezPoints` variables `colm` and `rowm`) is given in the command's `formula`, as illustrated below .

```
variogram(yield ~ colm + rowm, data=AragonezPoints)


      np      dist     gamma dir.hor dir.ver   id
1   1944  3.026404 0.8612585       0       0 var1
2   6513  5.666938 0.9533967       0       0 var1
3  13187  9.613532 0.9616996       0       0 var1
4  14887 13.512151 0.9861200       0       0 var1
5  20259 17.441649 0.9984865       0       0 var1
```

```
plot(variogram(yield ~ 1, data=AragonezPoints, cutoff=75), pch=16)
```

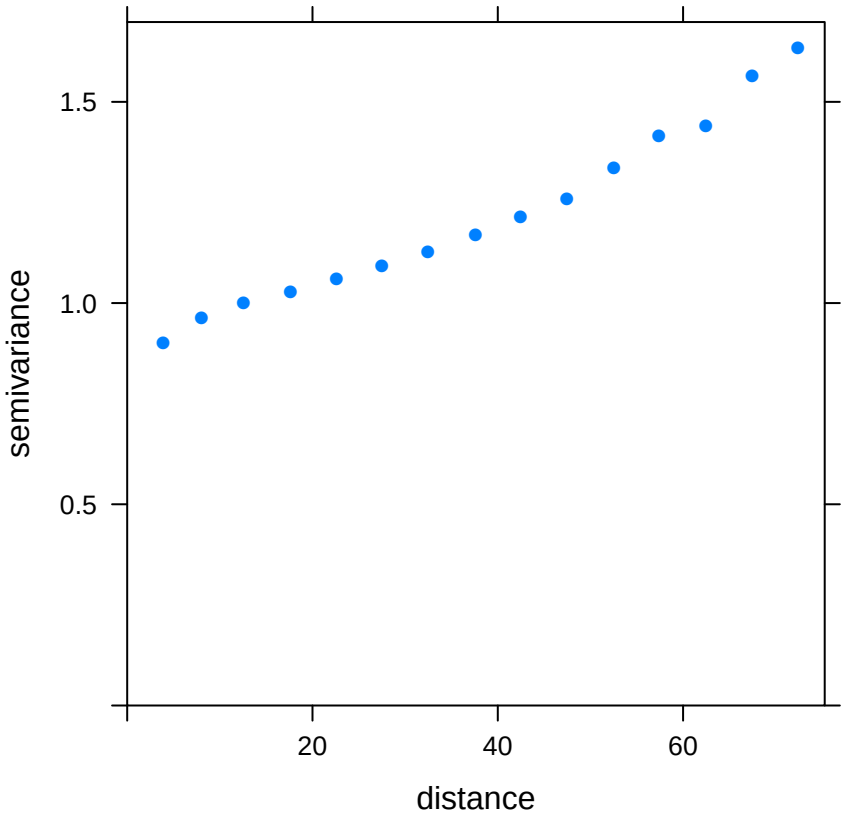


Figure 4.21: The plot of the empirical semi-variogram for the Aragonez yields, detrended by the (constant) mean, but with a cutoff value of $75m$.

```
6  20529 21.302718 1.0167690       0       0 var1
7  24687 25.061267 1.0102689       0       0 var1
8  28165 29.142116 1.0295995       0       0 var1
9  26756 33.097528 1.0337417       0       0 var1
10 28621 36.892389 1.0427308       0       0 var1
11 29117 40.763906 1.0539613       0       0 var1
12 29146 44.621526 1.0555425       0       0 var1
13 28860 48.412351 1.0741435       0       0 var1
14 29956 52.323021 1.0991958       0       0 var1
15 27872 56.240912 1.1142427       0       0 var1
```

The corresponding plot is given in Figure 4.22. The semi-variogram flattens out, suggesting

```
plot(variogram(yield ~ colm + rowm, data=AragonezPoints), pch=16)
```

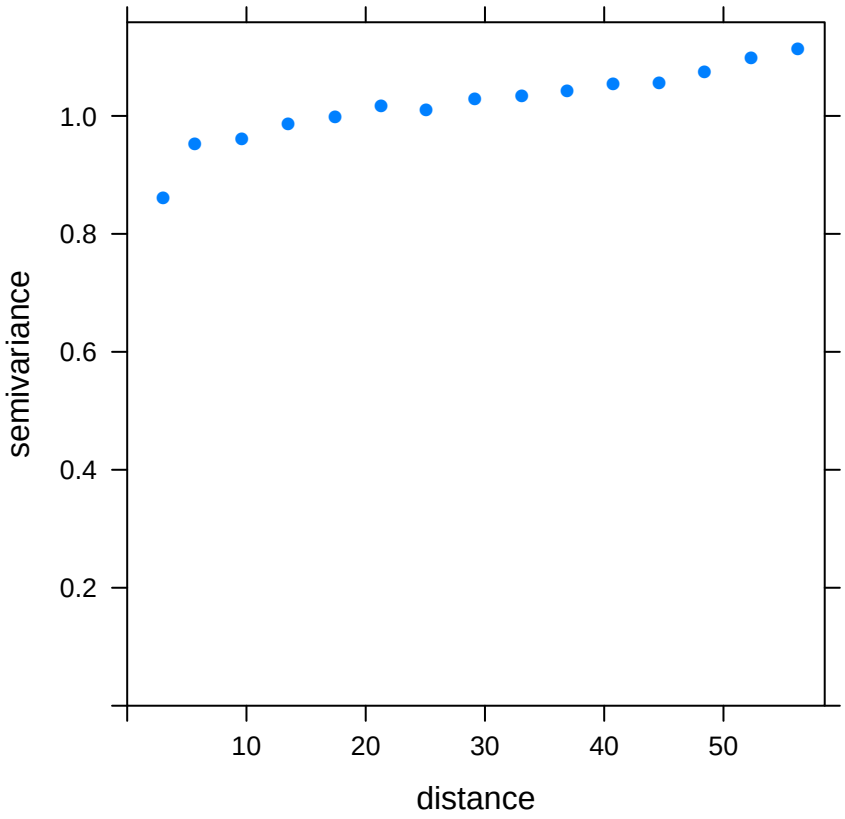


Figure 4.22: The plot of the empirical semi-variogram for the Aragonez yields, detrended by a linear regression on column and row distances.

that the linear detrending has been more successful than detrending by just a constant value. The sill appears to be slightly above 1 and the nugget value at 0.8. The range of values of d for which $\gamma(d)$ is clearly smaller than the sill appears to end at about $d = 20$, although the exact borderline is debatable.

The package `geoR` An alternative way of computing the empirical semi-variogram is through the `geoR::variog` function. One possible way of invoking this function is to provide

the matrix of coordinates via the `coords` argument, and the vector with the detrended values through the `data` command. Alternatively, we can provide the original data vector and request a specific form of detrending, using the `trend` argument. Below we show how to use this command with our dataset, using the previously linearly detrended variable `yieldldt`. As the full output is rather lengthy, we show only the components `u` and `v` which correspond to the lags d and the values $g(d)$, respectively, as well as the output component n which indicates the number of points used to compute each of the estimates above.

```
AragPointsVariog <- variog(coords=coordinates(AragonezPoints),
                           data=AragonezPoints$yieldldt)

variog: computing omnidirectional variogram

AragPointsVariog$u

 [1]   4.977219  14.931657  24.886094  34.840532  44.794970  54.749408
 [7]  64.703845  74.658283  84.612721  94.567159 104.521596 114.476034
[13] 124.430472

AragPointsVariog$v

 [1] 0.9431928 0.9913348 1.0151768 1.0393587 1.0617242 1.1096586 1.1409804
 [8] 1.1868190 1.2239065 1.2387683 1.2562398 1.2209068 1.0539211

AragPointsVariog$n

 [1] 15572 44191 60139 73986 75903 71073 66636 51950 34997 16787  5712
[12]  1553   172
```

Unlike the `gstat::variogram` command, `geoR::variog` uses, by default, all the spatial lags d (as midpoints of intervals, or *bins*) for which it finds pairs of points. The number of corresponding points, given in the output component `n`, decreases substantially as the spatial lag d grows, and the estimated values of $\gamma(d)$ drop, for large d, as can be seen in Figure 4.23. This Figure resulted from applying a `plot` command to the above `variog` function, using the plot method provided by the `geoR` package. The `variog` command also has `max.dist` argument to control the maximum distance d that is used, therefore avoiding this undesirable

effect.

```
plot(variog(coords=coordinates(AragonezPoints), data=AragonezPoints$yieldldt))

variog: computing omnidirectional variogram
```

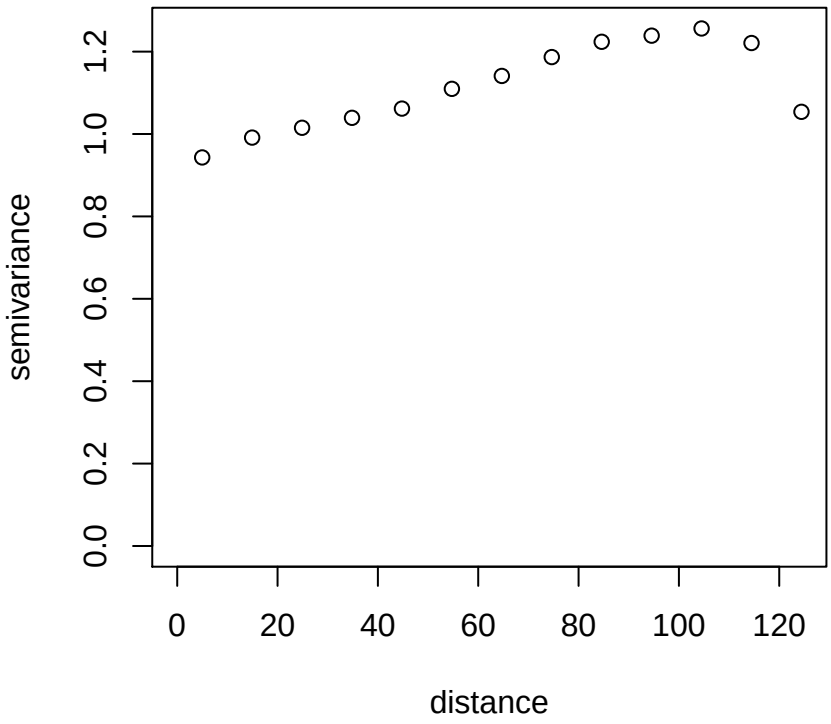


Figure 4.23: The plot of the empirical semi-variogram for the Aragonez yields, detrended by a linear regression on column and row distances, obtained using the `variog` command form package `geoR`.

4.7.4 Variogram models

Several functions have been proposed for smooth semi-variogram curves. The `gstat` package provides tools to fit many such models, plot smooth curves on the empirical semi-variogram, and obtain estimates for the sill, the nugget effect and the range. The basic function is

the `gstat::fit.variogram` command. This takes two main arguments: an empirical semi-variogram, and a corresponding model function. The model function is specified with the help of the `gstat::vgm` command, which requires as arguments initial estimates of the nugget effect c_0, the range r and the partial sill p (the difference between the sill and the nugget, that is, $p = sill - c_0$), as well as the *type* of model function. Some common types are:

exponential model: for $d > 0$, the semi-variogram is given by the function

$$\gamma(d) \;=\; c_0 + p\left[1 - \mathrm{e}^{-\frac{d}{r}}\right] \;, \tag{4.26}$$

where c_0 is the nugget, r the range and p the partial sill. As the distance d increases, this function grows towards an asymptotic sill (given by $c_0 + p$), which is not attained.

spherical model: *for* $0 < d < r$, the semi-variogram is given by the function

$$\gamma(d) \;=\; c_0 + p\left[\frac{3}{2}\frac{d}{r} - \frac{1}{2}\left(\frac{d}{r}\right)^3\right] \;, \tag{4.27}$$

with $\gamma(d) = sill = c_0 + p$ *for* $d > r$. This model assumes that for $d > r$ there ceases to be spatial dependence and thereafter the semi-variogram $\gamma(d)$ is constant (as is the case when no spatial autocorrelation exists).

gaussian model: for $d > 0$, the semi-variogram is given by the function

$$\gamma(d) \;=\; c_0 + p\left[1 - \mathrm{e}^{-\frac{d^2}{r^2}}\right] \;, \tag{4.28}$$

with constants defined as above.

Table 4.1 collects these and other frequent semi-variogram models for isotropic spatial dependence.

The commands to fit the exponential and the spherical model to the empirical semi-variogram obtained with the linearly detrended Aragonez yields are given below.

```
AragVarioLin <- variogram(yield~colm+rowm, data=AragonezPoints,
                                    locations=coordinates(AragonezPoints))
m.fit <- fit.variogram(AragVarioLin, model=vgm(psill=0.3,"Exp", range=7, nugget=0.8))
m.fit
```

Model	Formula
Exponential	$\gamma(d) = c_0 + p\ [1 - \exp(-d/r)]$
Gaussian	$\gamma(d) = c_0 + p\ \left[1 - \exp\left[-(d/r)^2\right]\right]$
Linear	$\gamma(d) = c_0 + p\ [1 - (1 - d/r)\mathbb{1}(d < r)]$
Rational quadratic	$\gamma(d) = c_0 + p\ \left[(d/r)^2/\left[1 + (d/r)^2\right]\right]$
Spherical	$\gamma(d) = c_0 + p\ \left[1 - \left[1 - 1.5(d/r) + 0.5(d/r)^3\right]\mathbb{1}(d < r)\right]$

Table 4.1: Some common isotropic semi-variogram models $\gamma(d)$ for spatial correlation structures, where the variable d stands for distance between points of observation. The parameter c_0 is the nugget effect, p the partial sill (sill minus nugget) and r is the range. The notation $\mathbb{1}(d < r)$ stands for an indicator variable for that condition: if $d < r$ is true, it takes the value 1, and multiplies the expression to its left, whereas if $d > r$, it takes the value zero and the expression which it multiplies disappears.

```
  model     psill    range
1   Nug 0.7906716 0.000000
2   Exp 0.2483298 7.297103


m2.fit <- fit.variogram(AragVarioLin, model=vgm(psill=0.3,"Sph", range=7, nugget=0.8))
m2.fit


  model     psill    range
1   Nug 0.7277332 0.000000
2   Sph 0.2759964 9.032198
```

While both models estimate a sill value slightly greater than 1 and a nugget effect close to 0.75, the estimates for the range differ substantially: the spherical model, which assumes that the semi-variogram becomes constant after a given value of d is more sensitive to the effects of small oscillations in the estimated values of $\gamma(d)$. The exponential model appears to be better suited in this case. Plotting the fitted model curves on the empirical semi-variogram plot gives the results in Figures 4.24 and 4.25.

These and other variogram models can be quickly viewed using the `gstat::show.vgms` function, with the result in Figure 4.26.

The `geoR` package also provides functionality to fit variogram models to an empirical semi-variogram model, namely the `geoR::lines.variomodel` command. The use of this command is illustrated below, after fitting an empirical variogram with the `max.dist` argument

```
plot(AragVarioLin, m.fit)
```

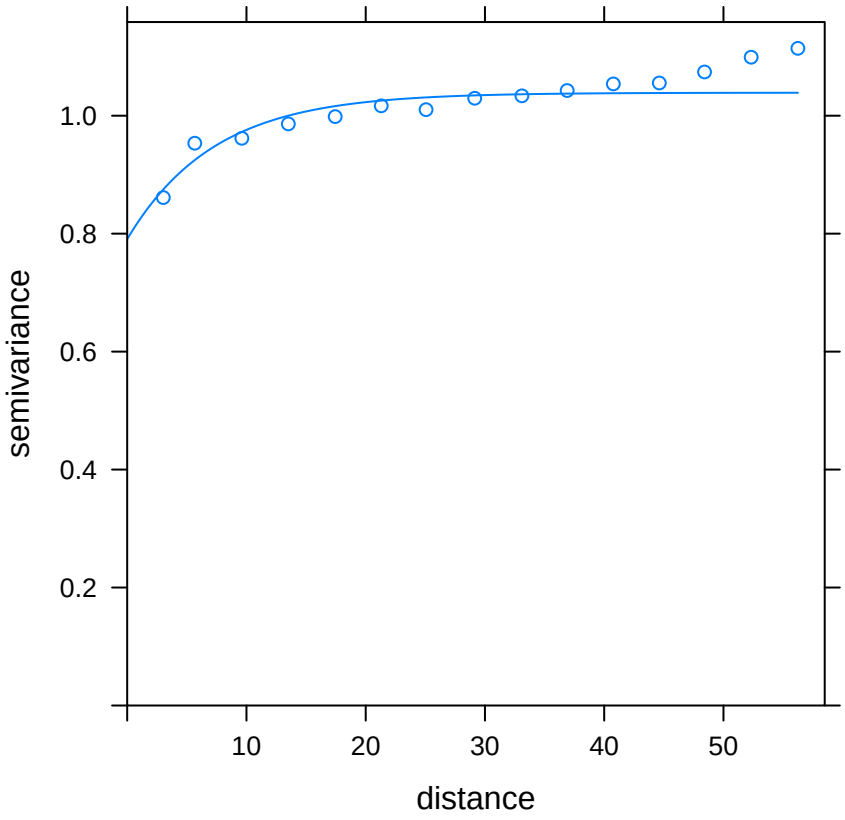


Figure 4.24: The empirical semi-variogram and the exponential semi-variogram model, fitted with the commands in package `gstat`.

set to a maximum spatial lag of 80. In a `lines.variomodel` command, it is necessary to define:

- the model type (`cov.model` argument);
- the initial estimates of the partial sill and the range (`cov.pars` argument); and
- the initial estimate of the nugget (`nugget` argument).

The results of the following commands are shown in Figure 4.27.

```
plot(AragVarioLin, m2.fit)
```

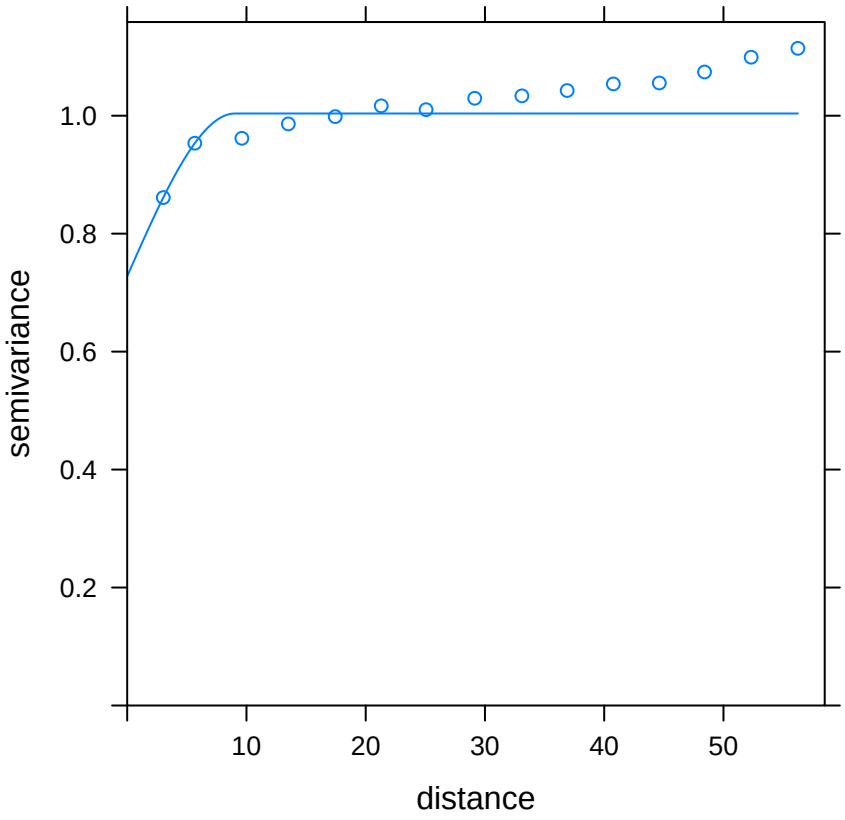


Figure 4.25: The empirical semi-variogram and the spherical semi-variogram model fitted with the commands in package `gstat`.

```
AragVariog <- variog(coords=coordinates(AragonezPoints),
                data=AragonezPoints$yieldldt, max.dist=80)


variog: computing omnidirectional variogram
```

Variogram models play an important role in spatial statistical models, as they are used to define the structure of the (co-)variance matrices for the error terms. Variogram models will be further discussed in Chapter 5.

```
show.vgms()
```

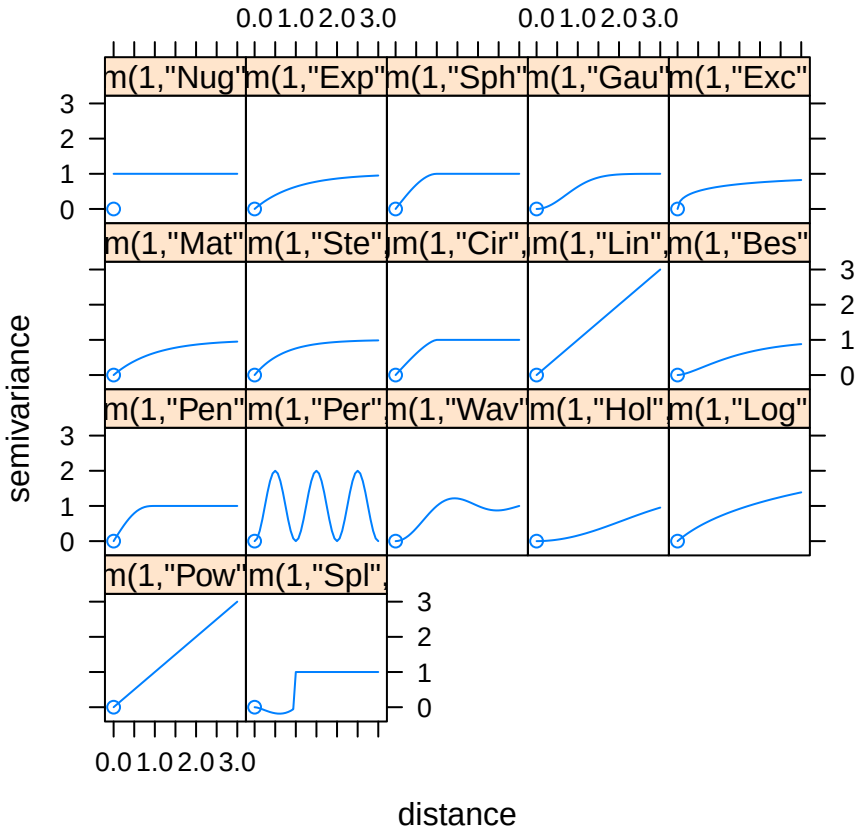


Figure 4.26: Various variogram model curves provided by the `gstat` package.

4.7.5 Anisotropy

Anisotropy is harder to identify, and to work with, than isotropy. The authors of the `gstat` package provide an argument `alpha` for the `gstat::variogram` function, which allows the user to define a vector of angles giving the main directions along which to inspect if the resulting semi-variograms are similar. Let us compare the results of the function call without the `alpha` argument (page 118) with the empirical semi-variogram γ for each of two directions specified by `alpha`: 0 degrees (vertical) and 90 degrees (horizontal). The number of points used to estimate γ in each direction (`np`), for any given distance d, is now smaller. When very few points are used, the empirical semi-variogram may become erratic. This tends to

```
plot(AragVariog, pch=16)
lines.variomodel(cov.model="exp", cov.pars=c(0.2, 20), nugget=0.9)
```

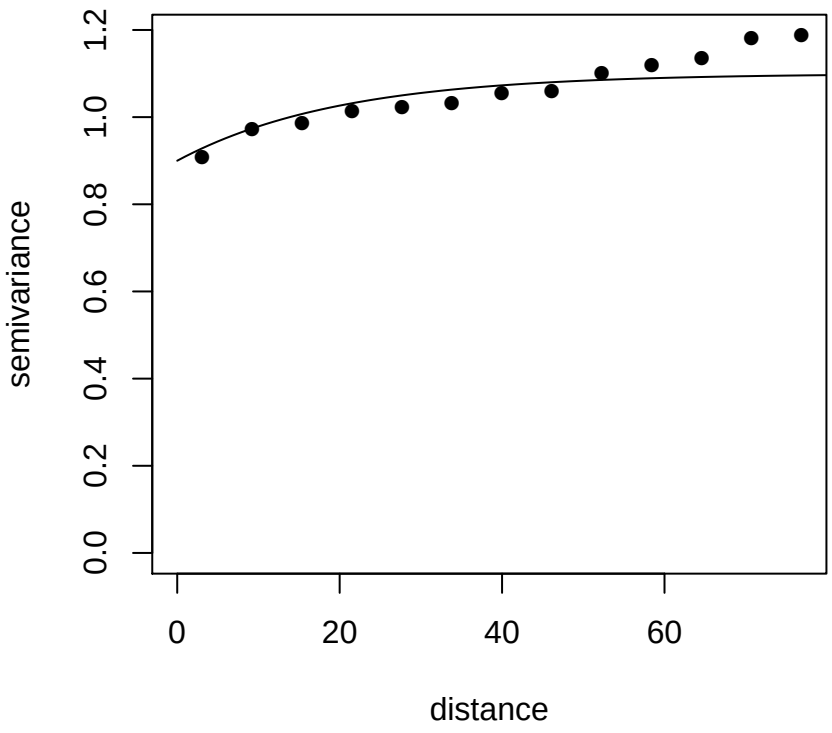


Figure 4.27: An exponential semi-variogram model, fitted on the empirical semi-variogram for the Aragonez linearly detrended yields, with the commands in package `geoR`.

occur if too many angular directions are specified. It is also possible to see that the distance bins chosen (by default) by the `variogram` command are not the same in the vertical (0 degrees) and the horizontal (90 degrees) directions, reflecting the fact that in the horizontal directions there are points separated by smaller distances than in the vertical (recall that columns are separated by $2.25m$, whereas in a vertical direction, the separation between two adjacent points is $3.75m$). Isotropy should produce similar estimates of γ in both directions, whereas if anisotropy is present, we would expect to see different behaviour in the empirical semi-variograms for each angular sector.

Enclosing the above command within a `plot` function produces the graphs in Figure 4.28 which suggest that the differences between both semi-variograms are not substantial. Isotropy appears to be a reasonable assumption.

```
variogram(yield ~ colm + rowm , data=AragonezPoints, alpha=c(0,90))
```

```
      np      dist     gamma dir.hor dir.ver   id
1    965  3.761890 0.8553530       0       0 var1
2   2782  5.937408 0.9799105       0       0 var1
3   7021  9.841740 0.9568937       0       0 var1
4   7459 13.651210 0.9929169       0       0 var1
5   9349 17.644283 1.0216893       0       0 var1
6  11031 21.423086 1.0491466       0       0 var1
7  11696 25.208794 1.0482439       0       0 var1
8  14136 29.217699 1.0542855       0       0 var1
9  13115 33.227594 1.0453946       0       0 var1
10 14459 36.916577 1.0690293       0       0 var1
11 14372 40.867855 1.0663127       0       0 var1
12 14254 44.649496 1.0609603       0       0 var1
13 14835 48.420389 1.0816942       0       0 var1
14 15112 52.398543 1.1151889       0       0 var1
15 13430 56.296716 1.1327193       0       0 var1
16   979  2.301436 0.8670795      90       0 var1
17  3731  5.465264 0.9336268      90       0 var1
18  6166  9.353679 0.9671719      90       0 var1
19  7428 13.372512 0.9792947      90       0 var1
20 10910 17.268009 0.9786036      90       0 var1
21  9498 21.162923 0.9791656      90       0 var1
22 12991 24.928447 0.9760794      90       0 var1
23 14029 29.065957 1.0047252      90       0 var1
24 13641 32.972477 1.0225382      90       0 var1
25 14162 36.867693 1.0158808      90       0 var1
26 14745 40.662587 1.0419223      90       0 var1
27 14892 44.594754 1.0503568      90       0 var1
28 14025 48.403849 1.0661568      90       0 var1
29 14844 52.246134 1.0829139      90       0 var1
30 14442 56.189018 1.0970608      90       0 var1
```

Package `geoR` also provides a function `geoR::variog4` that computes the empirical variograms for four different directions specified by the user.

```
plot(variogram(yield ~ colm + rowm , data=AragonezPoints, alpha=c(0,90)))
```

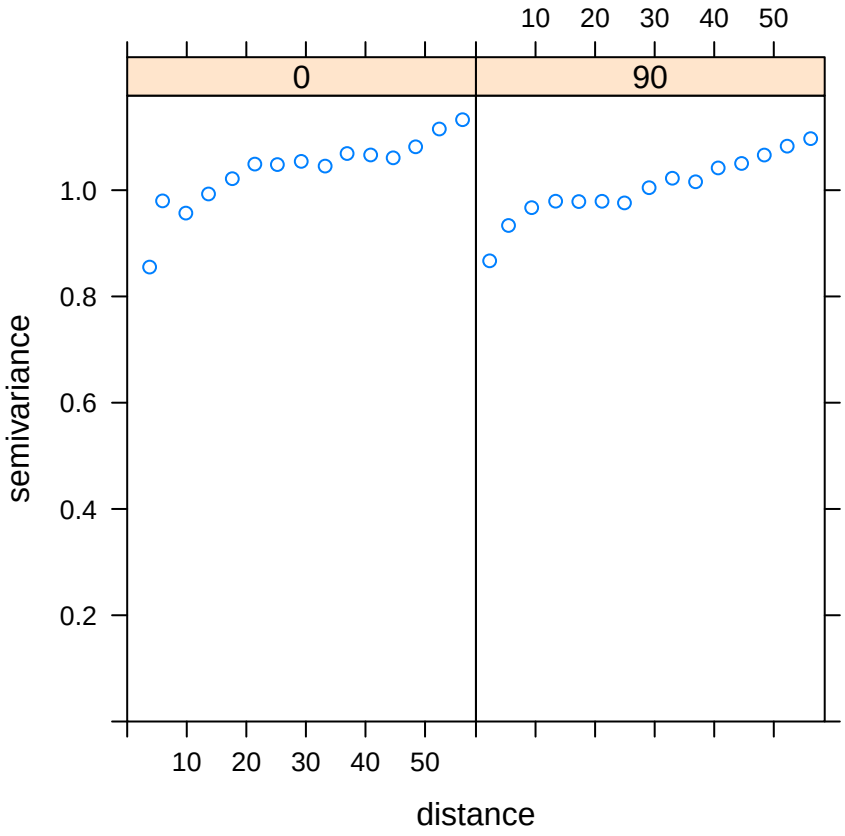


Figure 4.28: The empirical semi-variograms, computed for linearly detrended Aragonez yields which, in relation to the each observation, are in the angular sectors defined by the two main bisecting lines. With anisotropy, we would expect to see differences in the semi-variograms for points that tend to be along the vertical (0 degrees) and the horizontal (90 degrees) directions.

4.7.6 Correlograms

For second-order stationary isotropic models, the *correlogram*, or *autocorrelation function*, may be easier to interpret, although it is intimately connected to the semi-variogram. It basically considers the correlation coefficient between observations that are separated by a

spatial lag d:

$$\rho(d) \;=\; \frac{Cov[Z(s), Z(s+d)]}{\sqrt{Var[Z(s)]\,Var[Z(s+d)]}} \;=\; \frac{C_s(d)}{C_s(0)}\,. \tag{4.29}$$

The relation between the semi-variogram $\gamma(d)$ and the correlogram $\rho(d)$ therefore follows directly:

$$\begin{aligned} \gamma(d) \;&=\; C_s(0) - C_s(d) \;=\; C_s(0)\left[1 - \frac{C_s(d)}{C_s(0)}\right] \\ \Leftrightarrow \qquad \gamma_s(d) \;&=\; C_s(0)\,[1 - \rho(d)]. \end{aligned} \tag{4.30}$$

The intuitively obvious relation $\lim_{d\to+\infty} \rho(d) = 0$ is coherent with the idea that the sill is the asymptotic value of the semi-variogram as d tends to infinity. It is also natural that $\gamma_s(0) = 0$, since $\rho(0) = 1$.

The behaviour of the autocorrelation function for neighbours of increasing order k can be visualized with the `spdep::sp.correlogram function`, by choosing the option "`corr`" in the `method` argument, as shown in Figure 4.29.

4.8 Two or more spatial variables

Let us briefly consider some concepts relating to spatial correlation between *different* variables. The study of relations between different variables has a long tradition in standard statistics. Methods such as linear and non-linear regression, generalized linear models, mixed models, and others, are part of standard statistical courses. If spatial autocorrelation and cross-correlation between different variables exists, it should be taken into account.

An important preliminary consideration is whether the variables that have been observed are *collocated* (*co-located*), that is, if they are observed at the same set of locations. To simplify what follows, we will assume that different variables are indeed collocated. If the observed variables are *not* collocated, it is necessary to interpolate in order to obtain a collocated set of data, an issue that will be dealt with in Chapter 6.

```
plot(sp.correlogram(nb.k4, var=AragonezPoints$yieldldt, method="corr", order=10))
```

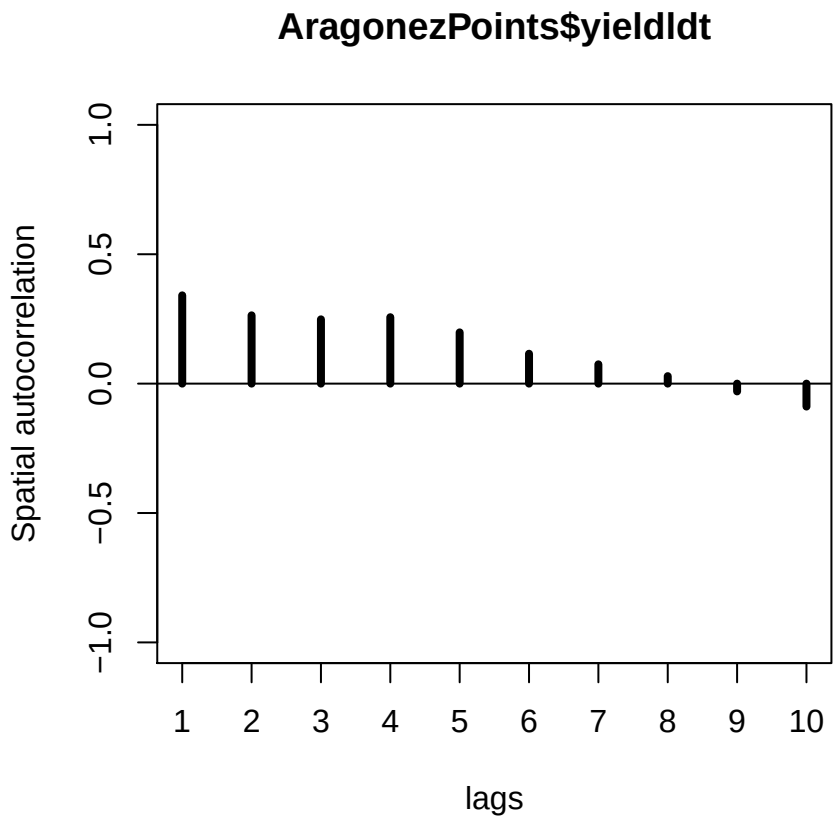


Figure 4.29: The autocorrelation function for the linearly detrended Aragonez yields, based on a $k = 4$ nearest neighbours list and a row-normalized weight matrix, with lags of up to 10. This correlogram was produced by the function `spdep::sp.correlogram`. Autocorrelation seems become negligeable at about lag 6.

4.8.1 The cross-variogram

Consider a set of different variables $\{Z_{[i]}\}_i$ that are isotropic spatial processes. The variogram function for any single variable $Z_{[i]}$ was defined in equation (4.21) as:

$$2\gamma_{ii}(d) \;=\; Var\left(Z_{[i]}(s) - Z_{[i]}(s+d)\right) \tag{4.31}$$

The most frequent definition to extend this concept to a pair of different variables, $Z_{[i]}$ and

$Z_{[j]}$, is (see, for example Bivand *et al*, [7]):

$$2\gamma_{ij}(d) \;=\; Cov\left[\left(Z_{[i]}(s) - Z_{[i]}(s+d)\right), \left(Z_{[j]}(s) - Z_{[j]}(s+d)\right)\right] \tag{4.32}$$

Cressie [1] gives an alternative definition, which is better suited for some purposes:

$$2\gamma_{ij}(d) \;=\; Var\left(Z_{[i]}(s) - Z_{[j]}(s+d)\right) \tag{4.33}$$

Note that both these extensions give the standard variogram when $i=j$.

The `gstat` package computes and plots cross-variograms when multiple variables are supplied. We will illustrate their use with a second, meteorological dataset, and inspired by a similar example in Bivand *et al* [7].

4.8.2 A meteorological dataset

A small meteorological dataset was downloaded from the website of the European Centre for Medium-Range Weather Forecasts (ECMWF)[1]. The data are not direct measurements, but rather *reanalysis* data, that is, data that has been collected from various sources and processed, in this case by the ERA-Interim data assimilation system. It is natural that reanalysis data be smoother, with 'nicer' properties than directly observed data.

For a given hour of June 18, 2016, reanalysis values were obtained, relative to a rectangular grid covering 24 longitudes from 9W to 8E and 23 latitudes from 36N to 52N. The variables in the dataset are:

Short name	Long name	Units
t2m	temperature at 2 meters	°K
stl1	soil temperature level 1 (surface)	°K
stl2	soil temperature level 2	°K
sund	sunshine duration	(s)
tp	total precipitation	(m)

The data format in the ERA-Interim website is NetCDF (see also Exercise E.2.3, in Appendix E). With the help of the `R` packages `ncdf4` and `raster`, the dataset was transformed into an `R` data frame, called `meteo`, whose first six lines are shown below:

[1] `apps.ecmwf.int/datasets/interim-full-daily`

```
load(file="datasets/meteo.RData")
head(meteo)
```

```
    lon  lat      t2m     stl1     stl2     sund           tp
1 -9.00 52.5 283.7192 284.7060 286.6936 23399.97 0.0012258549
2 -8.25 52.5 283.6690 284.8637 286.8675 22049.91 0.0009417163
3 -7.50 52.5 284.0929 285.2671 287.2538 22219.33 0.0010797265
4 -6.75 52.5 284.6273 285.6325 287.4712 22949.73 0.0012786235
5 -6.00 52.5 285.9954 285.8502 285.8498 24637.31 0.0007062872
6 -5.25 52.5 285.9974 285.9481 285.9473 25986.71 0.0007062872
```

The longitudes were converted to the range -9 to 8, so that contiguous plotting of any results could be ensured. The temperatures are in degrees Kelvin, but since the data will be detrended, it is irrelevant if the units are given in degrees Celsius. Total precipitation is recorded in meters and sunshine duration in seconds. The standard linear correlation coefficients are given below (to two decimal places). Unsurprisingly, they reveal strong positive correlations between the three temperature variables and negative correlations between rainfall and the temperature variables. Somewhat surprisingly, sunshine duration is almost uncorrelated with most variables, with only a small negative correlation with total precipitation.

```
round(cor(meteo[,3:7]),d=2)
```

```
       t2m  stl1  stl2  sund    tp
t2m   1.00  0.97  0.86 -0.01 -0.47
stl1  0.97  1.00  0.95  0.02 -0.50
stl2  0.86  0.95  1.00  0.05 -0.50
sund -0.01  0.02  0.05  1.00 -0.24
tp   -0.47 -0.50 -0.50 -0.24  1.00
```

We now build objects of class `sf`, and then of class `SpatialPointsDataFrame`, as described in Subsection 4.1. Two different objects of class `sf` are built, one using the standard EPSG 4326, in longitudes and latitudes, and another using the EPSG 3034 conformal mapping CRS for Europe, which has the advantage of providing coordinates in meters, and is therefore better suited to calculating distances between observations.

```
meteo4326.sf <- st_as_sf(meteo, coords=c("lon","lat"), crs=4326)
meteo.sf <- st_transform(meteo4326.sf, crs=3034)
meteo.sp <- as_Spatial(meteo.sf)
```

The `plot` method for objects of class `sf` provides a first visualization of the dataset.

```
plot(meteo.sf, pch=16)
```

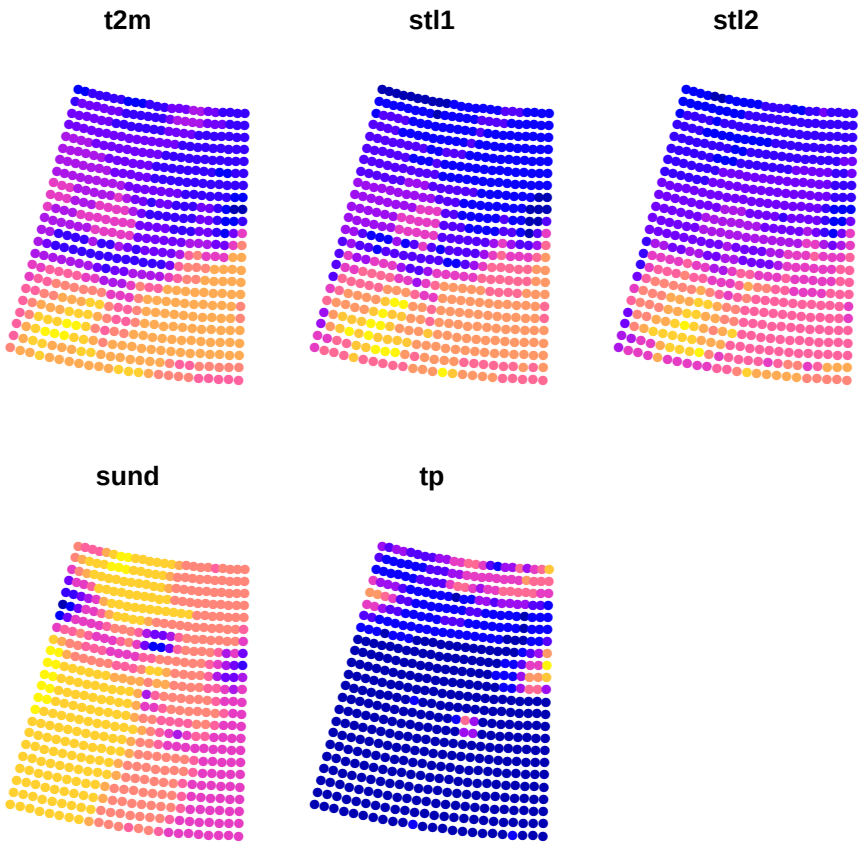


Figure 4.30: The meteorological variables in the `meteo` dataset, as plotted by the `plot` function, using the `meteo.sf` object of class `sf` (the coordinate reference system is EPSG:3034, giving distances in meters). A clear North-South gradient is visible for the temperature variables: temperature at 2 meters (`t2m`) and surface temperatures level 1 (`slt1`) and level 2 (`slt2`).

It is helpful to place the dataset on a map of Europe, and this will be done using the

`mapview` `R` package, and its function `mapView`. The commands are given in the simplest form. A tab should open in your browser, with the observed locations appropriately geo-referenced. A small dialogue window on the left of the browser window will allow you to select different types of maps. Figure 4.31 illustrates the result of the comand, with the "ESRI.WorldImagery" map option. Clicking on any of the circles will open a window with the information regarding that location, as illustrated in Figure 4.32.

```
mapView(meteo.sp, zcol="stl1")
```

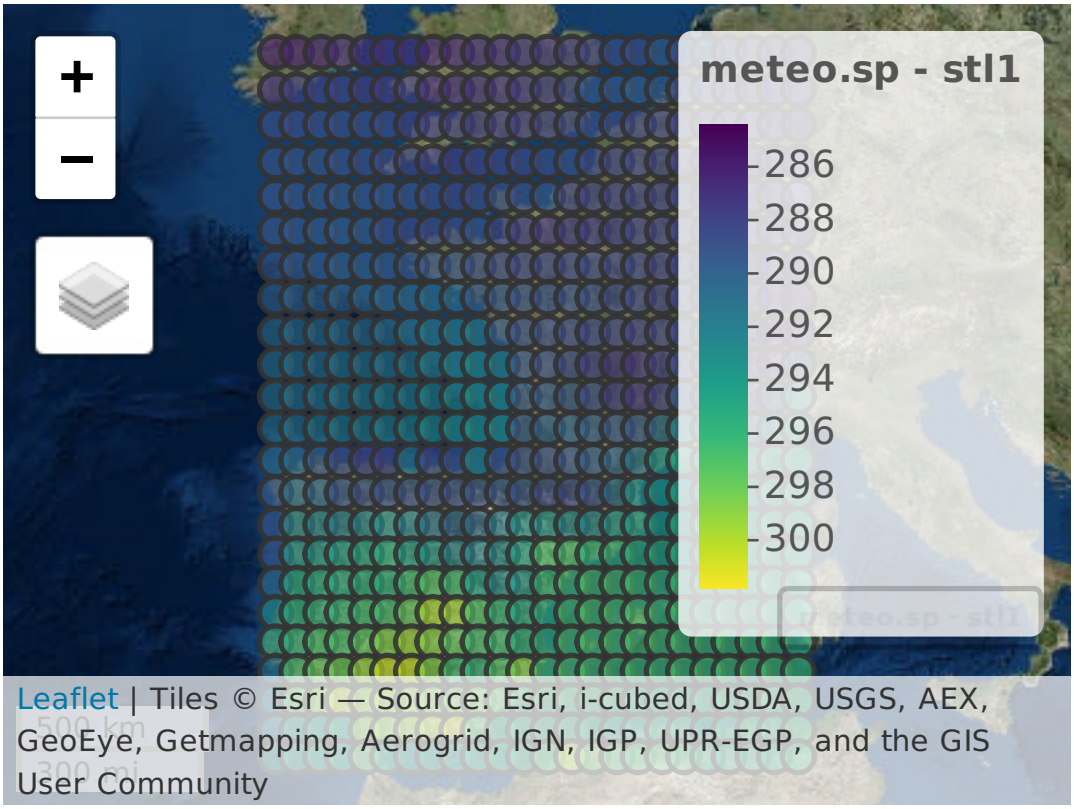


Figure 4.31: The meteorological variable `slt1` (surface temperature level 1) in the `meteo` dataset.

Several variables can be made available, using the `mapView` function, simply by providing a vector with their names to the `zcol` argument. This possibility will be illustrated below. But first, the object of class `SpatialPointsDataFrame` object, `meteo.sp`, will be converted into an object of class `SpatialPolygonsDataFrame` object. This will be done using the `dismo::voronoi` function. The `voronoi` function takes the coordinates of a set of points (in the example, as provided by the `SpatialPointsDataFrame` object `meteo.sp`) and creates

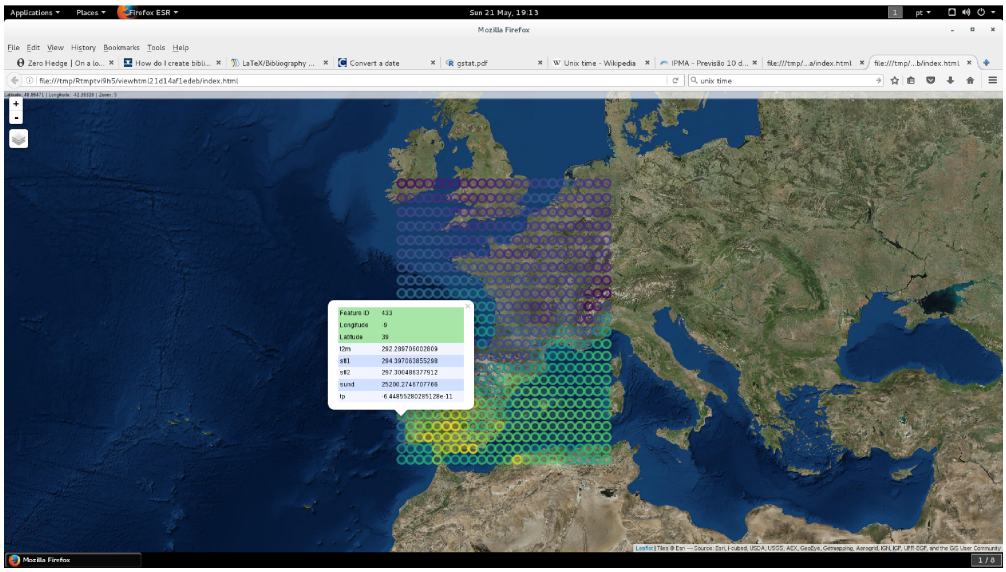


Figure 4.32: The meteorological variable `slt1` (surface temperature level 1) in the `meteo` dataset, with the information on a given location, interactively selected on the browser window.

Voronoi polygons (also known as Thiessen or nearest neighbour polygons). These are polygons that cover two-dimensional space, that is, define a *tessellation* of the space. For each given point in the set of coordinates, the associated Voronoi polygon is defined as the set of all locations in space that are closer to the given point than to any other point in the set. The `voronoi` function uses the `deldir` function in the package with the same name, that also defines tessellations and triangulations in space. The `mapView` command is invoked, with results given in Figure 4.33. As can be observed, the default sizes of the external polygons are too large (especially when the region is not aligned with the axes), but these can be controlled by the argument `ext`.

```
meteo.voronoi <- voronoi(as_Spatial(meteo4326.sf))
mapView(meteo.voronoi[,"tp"])
```

4.8.3 Cross-variograms in R

We now use the `gstat` package to produce the cross-variograms for these meteorological variables. We begin by defining an object `gobj`, of class `gstat`, which collects variables and allows for the possibility of detrending in ways that are defined within the command.

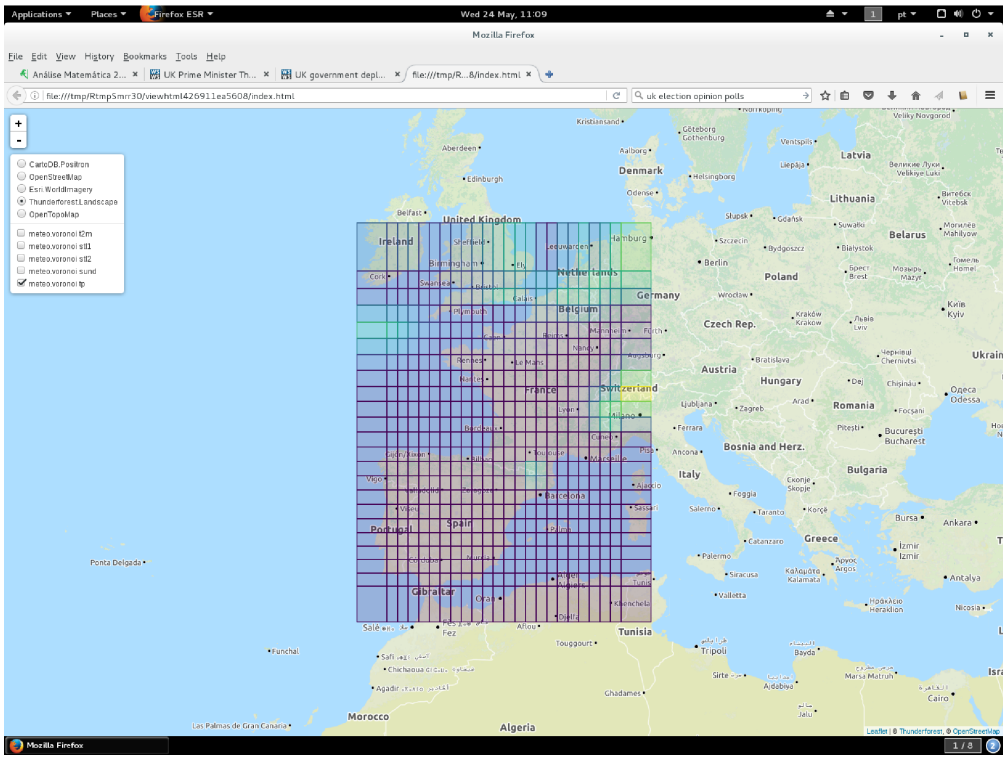


Figure 4.33: The `meteo.voronoi` polygons, created by the `voronoi` function, superimposed by the `mapView` function on their appropriate geographical coordinates. The background map is the `Thunderforest.Landscape` option on the browser window. The screenshot displays variable `tp`, total precipitation.

Objects of class `gstat` may be attached to each other, so as to produce a sequence of models for the different variables. Inspired by Bivand *et al* [7], each variable in the `meteo` dataset, will be detrended using a linear trend on the geographical coordinates:

```
gobj <- gstat(NULL, "t2m", t2m ~ coords.x1 + coords.x2, meteo.sp)
gobj <- gstat(gobj, "stl1" , stl1 ~ coords.x1 + coords.x2, meteo.sp)
gobj <- gstat(gobj, "stl2" , stl2 ~ coords.x1 + coords.x2, meteo.sp)
gobj <- gstat(gobj, "sund" , sund ~ coords.x1 + coords.x2, meteo.sp)
gobj <- gstat(gobj, "tp" , tp ~ coords.x1 + coords.x2, meteo.sp)
gobj


data:
t2m : formula = t2m``~``coords.x1 + coords.x2 ; data dim = 552 x 5
```

```
stl1 : formula = stl1`~`coords.x1 + coords.x2 ; data dim = 552 x 5
stl2 : formula = stl2`~`coords.x1 + coords.x2 ; data dim = 552 x 5
sund : formula = sund`~`coords.x1 + coords.x2 ; data dim = 552 x 5
tp : formula = tp`~`coords.x1 + coords.x2 ; data dim = 552 x 5
```

Having collected the five models, a call to `gstat`'s `variogram` function will compute both the empirical variograms and the empirical cross-variograms, as shown in Figure 4.34. It is useful to compare the resulting empirical cross-variograms with the correlation coefficients computed above. The variables whose cross-variograms have a clearer pattern are best suited for subsequent use in spatial models that use information from multiple variables.

Variogram models may be fitted to the empirical variograms and cross-variograms, using the `gstat::fit.lmc` function, as shown in Figure 4.35, where an exponential model was fitted in all cases. The numerical estimates of the ranges, nuggets and partial sills can be viewed by just writing the name of the object that results from invoking the `fit.lmc` function.

```
vario.meteo <- variogram(gobj)
plot(vario.meteo)
```

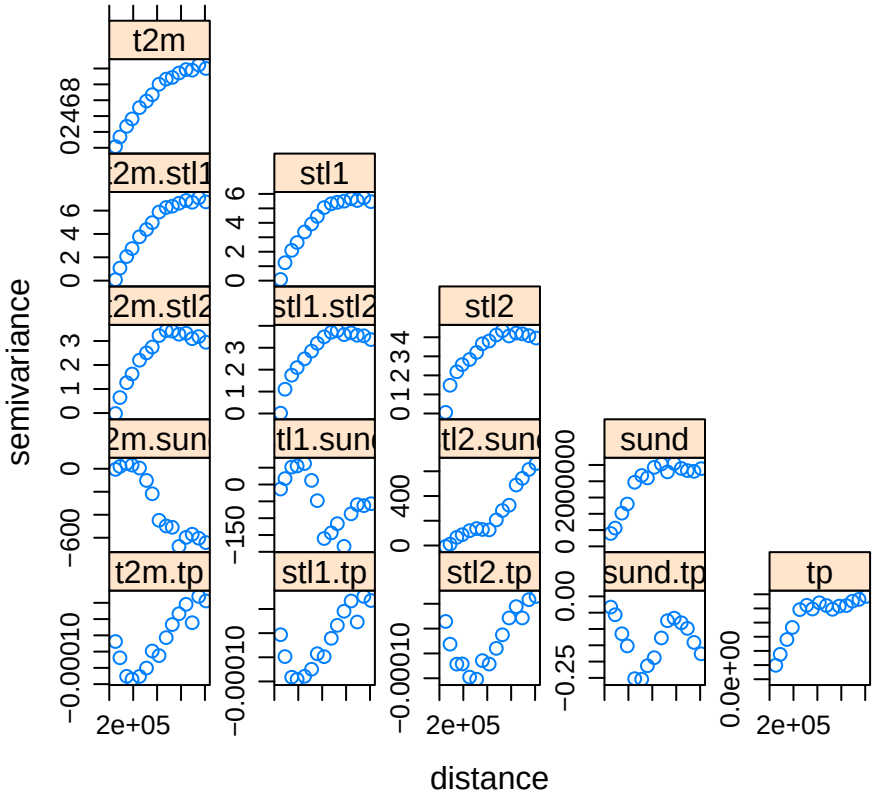


Figure 4.34: The variograms and cross-variograms for the variables in the `meteo` dataset, after detrending with a linear regression on the geographical coordinates.

```
vmeteo.fit <- fit.lmc(vario.meteo, gobj,
      vgm(psill=6, "Sph", range=750000, nugget=1))
plot(vario.meteo, vmeteo.fit)
```

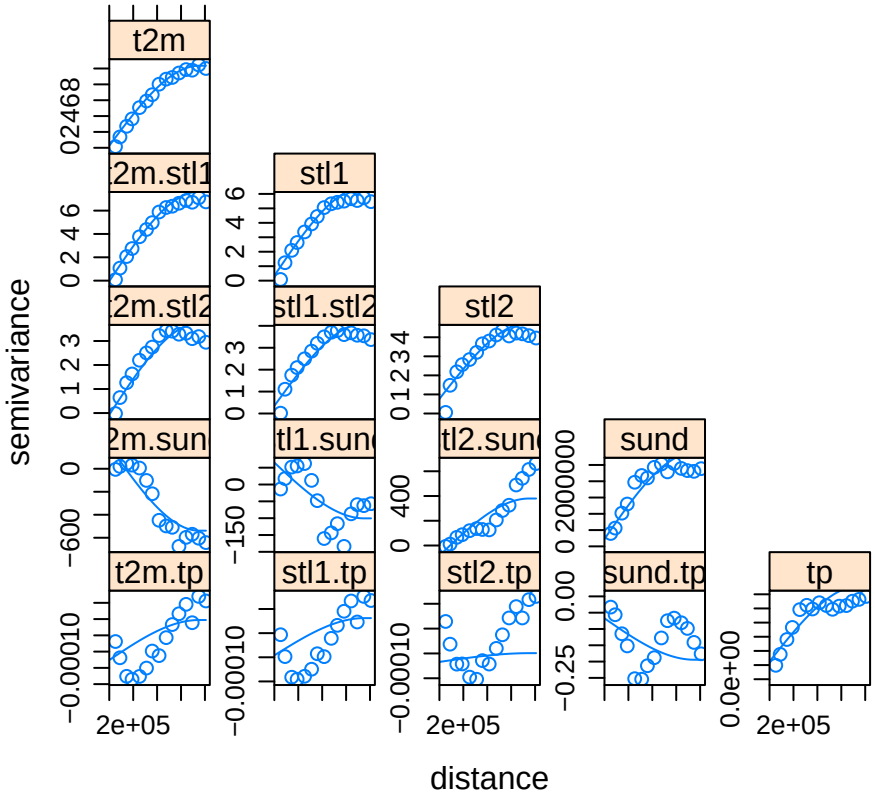


Figure 4.35: Fitted spherical variogram models for the variograms and cross-variograms for the variables in the `meteo` dataset, after detrending with a linear regression on the geographical coordinates.

Chapter 5

Regression Models for Spatially Autocorrelated Variables

Some parts of this chapter are inspired from the book *'Spatial Data Analysis In Ecology and Agriculture using R'*. R.E. Plant, CRC Press, 2012.

In these chapter we study linear models which are quite used in practice for regression. In practice, when using these models the error terms are classically assumed to be independent and identically distributed according to a Gaussian distribution. However this is often not the case when dealing with spatial dataframes, and in the previous chapter we studied how to detect a spatial autocorrelation. In this chapter, we will see that ignoring such a spatial structure of the error terms can have consequences (section 5.1), and we will study the possible linear models that can be used in presence of spatially autocorrelated variables (sections 5.3 to 5.6).

The following `R` packages will be needed in this chapter. We begin by loading them (they must have been previously installed on your platform).

```
library(RColorBrewer)
library(gstat)
library(spdep)
library(spatialreg)
```

5.1 Origins and Consequences of Spatial Autocorrelation

When we detect an apparent spatial autocorrelation (on residuals for instance), this spatial autocorrelation may or may not be the result of a spatial autocorrelation. In 1984, Miron identified three origins of apparent or real spatial autocorrelation: interaction, reaction and missspecification.
To explain these notions, we will take the example of a population of plants growing in a particular region. Suppose Y_i represents a measurement of plant productivity such as tree height or population density, and that the population is sufficiently dense relative to the spatial scale that the productivity measurement may be modeled as varying continuously with the location. We note X_{i1} the amount of light available at location i and X_{i2} the amount of available nutrients at location i. Using these two explanatory variables, the simplest model is the classical linear model:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \epsilon_i, \quad \text{with} \quad \epsilon_i \underset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2). \tag{5.1}$$

In matrix notation:

$$\begin{aligned} Y &= X\beta + \epsilon \\ \epsilon &\sim \mathcal{N}_n(0, \sigma^2 I). \end{aligned} \tag{5.2}$$

Below, we explain the three notions separately, but they can be combined in a same model.

5.1.1 Origin: interaction

Spatial autocorrelation induced by interaction occurs when the response variables at different sites interact with each other. For instance, negative autocorrelation may occur if trees in close proximity compete with each other for light and nutrients, so that relatively productive tree populations tend to inhibit the growth of other trees. Positive autocorrelation would occur if existing trees produced acorns that do not disperse very far, which in turn results in more trees in the vicinity. If Y is positively autocorrelated, then the true underlying model is:

$$\begin{aligned} Y &= X\beta + \rho WY + \epsilon \\ \epsilon &\sim \mathcal{N}_n(0, \sigma^2 I), \end{aligned} \tag{5.3}$$

with WY the spatial lag.

Illustration using simulated data

We generate a dataset `simu_modlin` satisfying model (5.2) with $\beta = (0, 0.5, 0.3)$ and a dataset `simu_interaction` satisfying model (5.3) with $\beta = (0, 0.5, 0.3)$ and $\rho = 0.6$. Each dataset contains 1000 observations and X_1 and X_2 are simulated independently using gaussian distributions.

We can see that fitting the classical linear model (5.2) on `simu_modlin` gives good results, the β vector is well estimated, and the variance of the residuals is approximately 0.0001:

```
mod <- lm(Ylin ~ X1 + X2)
print(coef(mod), digits = 2)

(Intercept)          X1          X2
   -0.00021     0.49979     0.30028

var(mod$res)

[1] 9.560601e-05
```

However, if we fit the classical linear model (5.2) on `simu_interaction`, we note that the estimation of the β vector is biased, and the variance of the residuals is 0.06:

```
mod <- lm(Yinter ~ X1 + X2)
print(coef(mod), digits = 2)

(Intercept)          X1          X2
      0.027       0.550       0.327

var(mod$res)

[1] 0.06299029
```

The difference is not very large on the estimates because Y is not very large, but you can note the effect of positive interaction among the Y on the estimates of the regression coefficients.

You can also note how the variance of the residuals is increased. Indeed, as the lag term is not included in the fitted model (5.2), some of the variability that would be assigned to this term, if it were present, is instead assigned to the regression coefficients, and the other is assigned to the error term. For example, the true marginal effect of X_1, which is measured by β_1, will be incorrectly estimated because it will include some of the effects of the lag WY. The effects of the lag which are not assigned to β_1 or β_2 will be assigned to the error term, inflating the variance of the residuals.

5.1.2 Origin: reaction

Spatial autocorrelation induced by reaction occurs when the response variables are reacting to an external factor that varies in space, and when this factor is not taken into account by the model. For instance, if nearby plants are reacting to availability of water (which varies in the 'space'). In this case, the inclusion of this external factor in the linear model may be appropriate. It may be sufficient to explain the spatial autocorrelation, and to obtain non-autocorrelated residuals. For instance, the true model should be:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} + \epsilon_i, \quad \text{with} \quad \epsilon_i \underset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2), \tag{5.4}$$

with X_{i3} the distance from the river at location i.

Illustration using simulated data

1. We generate a dataset `simu_reaction1` satisfying model (5.4) with $\beta = (0, 0.5, 0.3, 0.8)$ and X_3 correlated with X_2.

   We can see that fitting model (5.4) on `simu_reaction1` gives good results, the β vector is well estimated:

```
print(coef(lm(Yreact1 ~ X1 + X2 + X3)), digits = 2)

(Intercept)          X1          X2          X3
     0.0088      0.4837      0.3315      0.7716
```

   However, if we fit model (5.1) on `simu_reaction1`, we note that the estimation of β_2 is biased. The reason is that the effect of X_3 has been 'loaded' on X_2.

```
mod <- lm(Yreact1 ~ X1 + X2)
print(coef(mod), digits = 2)

(Intercept)          X1          X2
       0.51        0.50        1.01
```

X_3 maybe interpreted as a 'spatial' variable, but its role in the model is identical to that of another explanatory variable without any spatial connotation.

2. We generate a dataset `simu_reaction2` satisfying model (5.4) with $\beta = (0, 0.5, 0.3, 0.8)$, and X_3 non correlated with X_1 or X_2 but spatially autocorrelated.

   We can see that fitting model (5.4) on `simu_reaction2` gives good results, the β vector is well estimated, and the variance of the residuals is approximately 1.

```
mod <- lm(Yreact2 ~ X1 + X2 + X3)
print(coef(mod), digits = 2)

(Intercept)          X1          X2          X3
      0.027       0.483       0.327       0.777

var(mod$res)

[1] 1.004552
```

   If we fit model (5.1) on `simu_reaction2`, we note that the estimation of the β vector is not biased. However, the variance of the residuals is doubled: it is approximately 1.86.

```
mod <- lm(Yreact2 ~ X1 + X2)
print(coef(mod), digits = 2)

(Intercept)          X1          X2
      0.051       0.469       0.312

var(mod$res)

[1] 1.90547
```

This is because the effect of X_3 which is not taken into account in this model is entirely loaded in the error term. As X_3 was spatially autocorrelated, the result is that the residuals are spatially autocorrelated:

```
lm.morantest(mod,W)


Global Moran I for regression residuals

data:
model: lm(formula = Yreact2 ~ X1 + X2)
weights: W

Moran I statistic standard deviate = 6.7573, p-value = 7.027e-12
alternative hypothesis: greater
sample estimates:
Observed Moran I      Expectation         Variance
    0.1550852149    -0.0010125073     0.0005336302
```

5.1.3 Origin: misspecification

In this case, the measured autocorrelation is not due to interaction or reaction but to the incorrect form of the model. For instance if we assume homoscedastic errors when in fact they are heteroscedastic. The true model should be for instance:

$$\begin{aligned} Y &= X\beta + \epsilon \\ \epsilon_i &\underset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2 \times \exp(1 + 2X_{i2})). \end{aligned} \tag{5.5}$$

Here the variance of the errors increases with the amount of available nutrients X_{i2}. In this case, the measured autocorrelation can be induced by the wrong modelisation, it is then an apparent autocorrelation and not a real autocorrelation (this autocorrelation cannot be explained by spatial considerations).

Illustration using simulated data

We generate a dataset `simu_modmiss` satisfying model (5.5) with $\beta = (0, 0.5, 0.3)$. X_2 is

spatially autocorrelated and the error variance is an increasing function of X_2.

We can see that fitting the classical linear model (5.2) on `simu_modmiss` gives a biased estimate for the β vector, and indicates a spatial autocorrelation of the residuals while in reality none exists:

```
mod <- lm(Ymiss ~ X1 + X2)
print(coef(mod, digits = 2))

(Intercept)          X1          X2
   30.95456   -68.36515    84.34902

lm.morantest(mod, W)


Global Moran I for regression residuals

data:
model: lm(formula = Ymiss ~ X1 + X2)
weights: W

Moran I statistic standard deviate = 2.3661, p-value = 0.008989
alternative hypothesis: greater
sample estimates:
Observed Moran I      Expectation         Variance
     0.159988117     -0.014521469      0.005439884
```

Indeed, the error terms are uncorrelated, but because the error variance is a function of X_2 and high values of X_2 tend to be near other high values of X_2, a test for spatial autocorrelation of the residuals has a high type I error rate.

5.1.4 Consequences of the spatial autocorrelation on classical linear models

Whatever the origin of apparent two-dimensional spatial autocorrelation, the effects of this autocorrelation on standard statistical methods are similar to those for one-dimensional au-

tocorrelation discussed in Chapter 2. The presence of autocorrelation decreases the effective sample size, as there are no longer n *independent* sources of information. Thus, the standard statistical techniques which are derived under the assumption of independence will provide mistaken significance levels and p-values, as well as mistaken confidence levels for confidence intervals. You can have some details by reading the Appendix D.
In particular, using simulations we have seen that spatial effects can impact the results of a classical linear model if not taken into account. For the three possible origins of spatial autocorrelation, we have seen the following consequences:

Interaction We obtain biased estimates of the regression coefficients, and the variance of the residuals is inflated, which can result in inflated type I or type II error rates of certain tests.

reaction If the reaction variable (not included in the model) is correlated to a variable present in the model, the estimate of the coefficient associated with the variable present in the model will be biased. If the reaction variable (not included in the model) is not correlated to a variable present in the model, but is spatially autocorrelated, the variance of the residuals will be inflated, resulting in Type I or type II error rates increased for certain tests, and an indication of spatial autocorrelation when none really exists.

Misspecification If the model is misspecified, that can lead to both biased estimates of the regression coefficient and indication of spatial autocorrelation when none really exists.

5.2 Working example: Las Rosas

In section 5.1, we showed that a classical linear model is impacted when spatial effects are not taken into account. To analyze spatial data, an adapted methodology can be summarized as follows:

1. Fit the data with a classical linear model like (5.2).
2. Check the model assumptions on the residuals: normality, homoscedasticity and independence.
    - To detect non-normality, some plots are possible: histogram, Q-Q plot.
    - To detect heteroscedasticity or the exclusion of a reaction variable, we will plot the residuals against the fitted values, and against the different variables included or not in the model.

- To detect dependence, note that the spatial autocorrelation often manifests itself in autocorrelation of the residuals. Hence we will try to detect a spatial autocorrelation of the residuals: bubble plots, semi-variograms, Moran correlogram, test for spatial autocorrelation of the residuals using the Moran's I.

3. If we detect some problems on the residuals:

   - Non-normality: the model can be misspecified. You can try a transformation of your variable to be explained and/or of your explanatory variables. It can also be the consequence of a relevant explanatory variable forgotten in the model.
   - Heteroscedasticity: you can take into account this heteroscedasticity in your model, see Chapter 5.6.
   - Spatial autocorrelation: you first need to check that you have not forgotten a reaction variable, and that you are not in presence of heteroscedasticity (misspecified model). If this is not the case, you need to fit a more complicated model with an autocorrelation structure. Two models specifically designed for spatial data are presented in sections 5.3 and 5.4. You can also use an extended linear model with a spatial autocorrelation structure, see section 5.6.

In section 5.2, the Las Rosas dataset ([6]) has been presented and a `SpatialPointsDataFrame` object `Xutm` containing the yield and relevant geographical variables to explain it has been created. We can inspect its structure using the `str` command.

```
load(file.path("datasets","LasRosas.RData"))
str(Xutm)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 1704 obs. of  10 variables:
  .. ..$ YIELD : num [1:1704] 4225 4308 4301 4443 4343 ...
  .. ..$ N     : num [1:1704] 125 125 125 125 125 ...
  .. ..$ elev  : num [1:1704] 272 272 272 272 272 ...
  .. ..$ slope : num [1:1704] 0.022 0.0238 0.0256 0.027 0.0282 ...
  .. ..$ slopeX: num [1:1704] 13.4 14.5 15.7 16.6 17.4 ...
  .. ..$ accu  : num [1:1704] 72.5 70.4 68.3 65.9 61.2 ...
  .. ..$ aspect: num [1:1704] 4.46 4.5 4.53 4.55 4.58 ...
  .. ..$ hshade: num [1:1704] 0.864 0.864 0.864 0.864 0.864 ...
```

```
  .. ..$ x     : num [1:1704] 420774 420781 420787 420794 420800 ...
  .. ..$ y     : num [1:1704] 6342855 6342853 6342850 6342847 6342845 ...
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1:1704, 1:2] 420774 420781 420787 420794 420800 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:1704] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "LONGITUDE" "LATITUDE"
  ..@ bbox       : num [1:2, 1:2] 420773 6342608 421412 6342981
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "LONGITUDE" "LATITUDE"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+proj=utm +zone=20 +south +ellps=WGS84 +datum=WGS84 +units
```

The yield, as well as relevant variables to explain it, can be represented using the function `spplot`. We can for instance represent the yields measured at each location on a map, with a color and a size proportional to the measured diameters, see figure 5.1.

We will use the data matrix `Xutm@data`, where slot `@data` returns a `data.frame`, which we can explore. For instance we can build the correlation matrix for `Xutm`.

```
round(cor(Xutm@data[,1:8]),3)
```

```
        YIELD      N   elev  slope slopeX   accu aspect hshade
YIELD   1.000  0.079 -0.881 -0.627 -0.107  0.889 -0.144  0.378
N       0.079  1.000 -0.022  0.008  0.003 -0.001  0.002 -0.043
elev   -0.881 -0.022  1.000  0.584  0.123 -0.954  0.108 -0.306
slope  -0.627  0.008  0.584  1.000 -0.051 -0.525  0.033 -0.368
slopeX -0.107  0.003  0.123 -0.051  1.000  0.016  0.965  0.708
accu    0.889 -0.001 -0.954 -0.525  0.016  1.000  0.015  0.424
aspect -0.144  0.002  0.108  0.033  0.965  0.015  1.000  0.613
hshade  0.378 -0.043 -0.306 -0.368  0.708  0.424  0.613  1.000
```

Interestingly, the correlations between `YIELD` and `accu` or `elev`, or even `slope`, are much stronger than the correlation between `YIELD` and the amount of nitrogen fertilizar `N`.
In order to fit a standard linear regression to the data, we will not put two variables too

correlated in the model. Hence, because of high correlations, we decide here to take into account `accu` but not `elev`, and `aspect` but not `slopeX`.

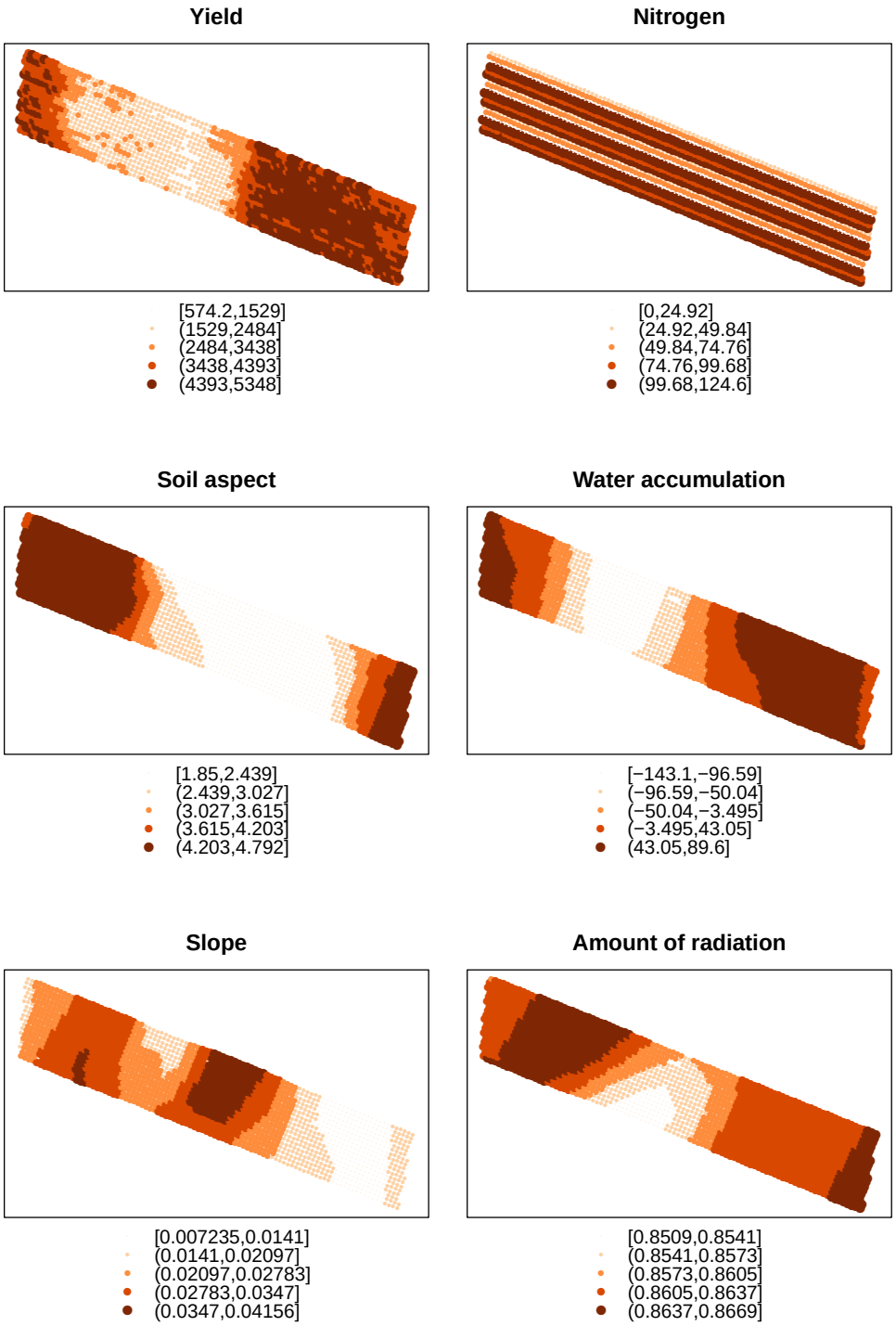


Figure 5.1: Maps of the yield, the dose of N, the aspect of the soil, the accumulation of water, the slope and the amount of radiation in the field.

This decision is conforted by the Figure 5.2, generated by the code below.

```
par(mfrow=c(3,3))
plot(YIELD ~ N, data=Xutm)
plot(YIELD ~ elev, data=Xutm)
plot(YIELD ~ slope, data=Xutm)
plot(YIELD ~ slopeX, data=Xutm)
plot(YIELD ~ accu, data=Xutm)
plot(YIELD ~ aspect, data=Xutm)
plot(YIELD ~ hshade, data=Xutm)
par(mfrow=c(1,1))
```

Looking at these scatterplots of Figure 5.2, we can see that the relationships between the yield and `elev` and between the yield and `accu` are similar, as well as the relationships between the yield and `aspect` and between the yield and `slopeX`. Moreover, we have the feeling that except for the scatterplot between the yield and `N`, the others can be separated in two scatterplots having different relationships with the yield. We suspect that the separation can be made regarding on the elevation, or the water accumulation. As we decided to keep only the water accumulation in our model, we decide to transform it into a factor with two levels: `low` and `high`, using the code below.

```
Xutm@data$accuf <- rep('low',dim(Xutm@data)[1])
for (i in 1:dim(Xutm@data)[1]){
  if(Xutm@data$accu[i] > -25){Xutm@data$accuf[i] <- 'high'}
}
Xutm@data$accuf <- as.factor(Xutm@data$accuf)
```

We plot again the yield against the explanatory variables we want to keep, but using colors to distinguish locations with low or high water accumulation. It appears clear that we indeed have different relationships between the yield and these explanatory variables, depending on the water accumulation level, see Figure 5.3

```
par(mfrow=c(2,3))
plot(YIELD ~ N, col=Xutm@data$accuf, data=Xutm)
plot(YIELD ~ accu, col=Xutm@data$accuf, data=Xutm)
plot(YIELD ~ slope, col=Xutm@data$accuf, data=Xutm)
```

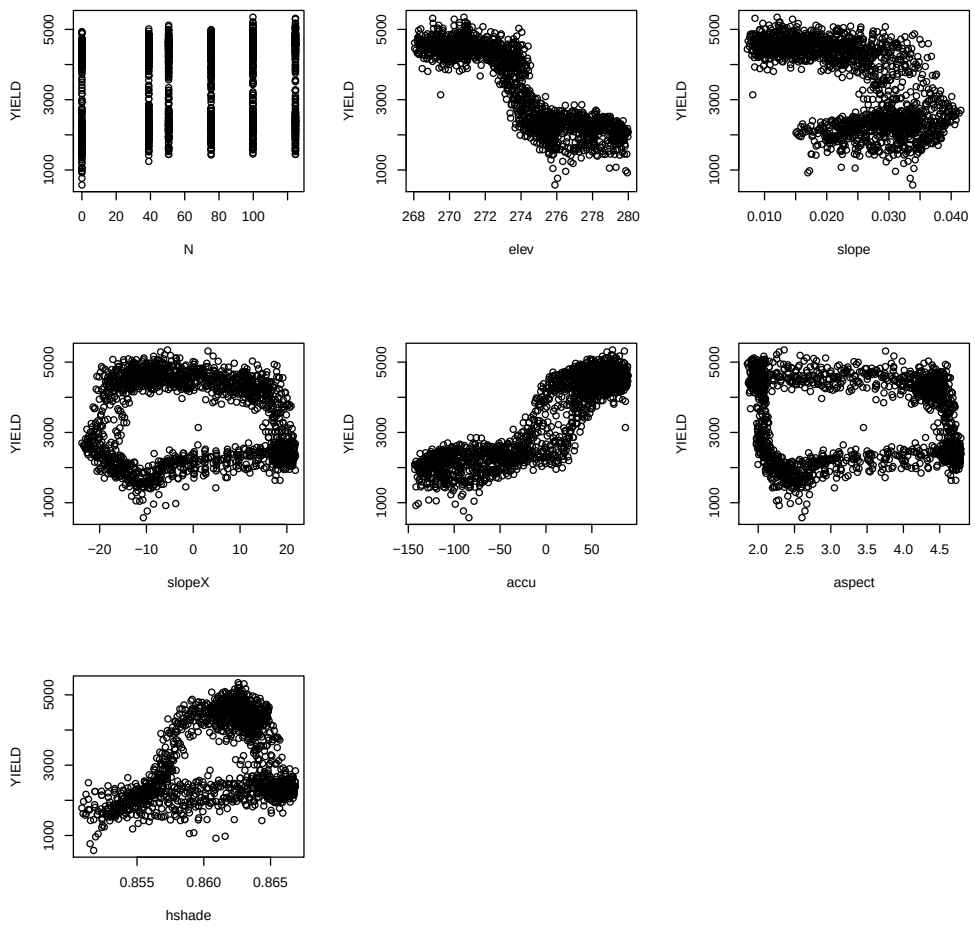


Figure 5.2: Plots of the yield against the possible explanatory variables: `N`, `elev`, `slope`, `slopeX`, `accu`, `aspect`, and `hshade`.

```
plot(YIELD ~ aspect, col=Xutm@data$accuf, data=Xutm)
plot(YIELD ~ hshade, col=Xutm@data$accuf, data=Xutm)
par(mfrow=c(1,1))
```

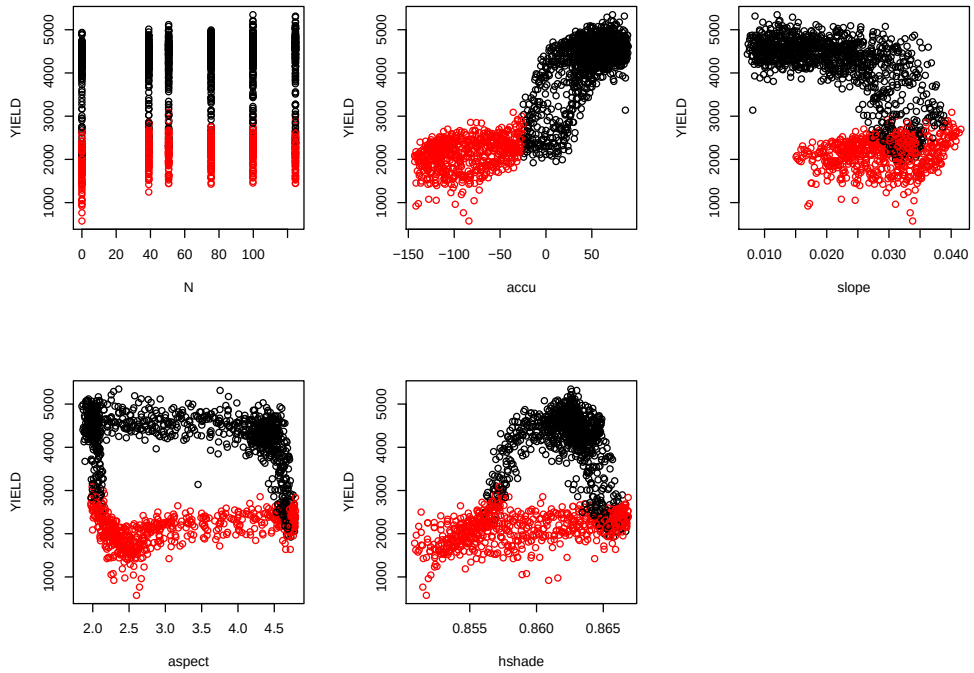


Figure 5.3: Plots of the yield against the explanatory variables: `N`, `accu`, `slope`, `aspect` and `hshade`. We suspect interactions between `accuf` and `slope`, `accuf` and `aspect`, and `accuf` and `hshade`.

We then decide to use a linear model to explain the yield with the following explanatory variables: `N`, `accuf`, `slope`, `aspect`, `hshade` and the interactions between `accuf` and `slope`, `accuf` and `aspect`, and `accuf` and `hshade`.

```
model.lm <- lm(YIELD ~ accuf + N + slope + aspect + hshade + accuf*slope
               + accuf*aspect + accuf*hshade, data=Xutm@data)
drop1(model.lm,test="F")

Single term deletions

Model:
YIELD ~ accuf + N + slope + aspect + hshade + accuf * slope +
    accuf * aspect + accuf * hshade
            Df Sum of Sq       RSS   AIC F value    Pr(>F)
```

```
<none>                    276691762 20462
N             1  18869062 295560825 20572 115.591 < 2.2e-16 ***
accuf:slope   1  96569568 373261331 20970 591.580 < 2.2e-16 ***
accuf:aspect  1   3455382 280147145 20481  21.168 4.521e-06 ***
accuf:hshade  1  16482486 293174248 20559 100.971 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(model.lm)
```

```
Call:
lm(formula = YIELD ~ accuf + N + slope + aspect + hshade + accuf *
    slope + accuf * aspect + accuf * hshade, data = Xutm@data)

Residuals:
     Min       1Q   Median       3Q      Max
-1767.17  -222.15    17.28   236.15  1517.32

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      6.570e+04  8.007e+03   8.204 4.53e-16 ***
accuflow        -1.004e+05  9.520e+03 -10.549  < 2e-16 ***
N                2.601e+00  2.419e-01  10.751  < 2e-16 ***
slope           -5.925e+04  1.626e+03 -36.436  < 2e-16 ***
aspect          -1.600e+02  1.643e+01  -9.739  < 2e-16 ***
hshade          -6.949e+04  9.314e+03  -7.461 1.37e-13 ***
accuflow:slope   7.863e+04  3.233e+03  24.322  < 2e-16 ***
accuflow:aspect  1.471e+02  3.197e+01   4.601 4.52e-06 ***
accuflow:hshade  1.116e+05  1.110e+04  10.048  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 404 on 1695 degrees of freedom
Multiple R-squared:  0.8796,Adjusted R-squared:  0.879
F-statistic:  1548 on 8 and 1695 DF,  p-value: < 2.2e-16
```

All the interactions are statistically significant, and this regression explains 88% of the yield variability.

The equation of the model is the following, with $Yield_{ij}$ the value of yield at the jth location having level i of `accu`.

$$\begin{aligned} Yield_{ij} &= \beta_0 + \alpha_i + \beta_1 N_{ij} + \beta_2 slope_{ij} + \beta_3 aspect_{ij} + \beta_4 hshade_{ij} \\ &\quad + \gamma_{2i} slope_{ij} + \gamma_{3i} aspect_{ij} + \gamma_{4i} hshade_{ij} + \epsilon_{ij}, \qquad (5.6) \\ \epsilon_{ij} &\underset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2). \qquad (5.7) \end{aligned}$$

Using this modelisation, the error terms are assumed to be independent, to follow the Gaussian distribution and to be homoscedastic. In particular, no spatial correlation of the error term is assumed. Before going further in the analysis (confidence intervals, tests, interpretation,...), we then need to validate these assumptions. Several tests and figures should be done.

First, we look at the basic diagnosis plots given by R, see Figure 5.4.

```
par(mfrow=c(2,2))
plot(model.lm)
```

Concerning the homoscedasticity, the variance does not seem to increase or decrease with the fitted values. Concerning the normality assumption, we observe some deviations for the tails (both extremities of the Q-Q plot). But importantly, we detect a pattern in the residuals, they do not appear to be independent, a trend seems to have been forgotten. As we suspect a trend to have been forgotten in the residuals, it is necessary to also plot the residuals against every possible explanatory variable, see Figure 5.5. Here some patterns appear again, the residuals seem to be correlated.

As we are in presence of spatial data, a final step is to check if the residuals are spatially independent. We can first represent the residuals on a map. We are looking for signs of spatial autocorrelation among the residuals, see Figure 5.6. Each residual is represented by a bubble whose size and color are proportional to its value. Here it is not difficult to tell from this figure that a spatial autocorrelation exists.

```
Xutm$resmodel.lm <- model.lm$res
spplot(Xutm, "resmodel.lm", col.regions=brewer.pal(9,"Oranges"),cex=.2*(1:5),
            key.space="bottom", main="Residuals of model.lm")
```

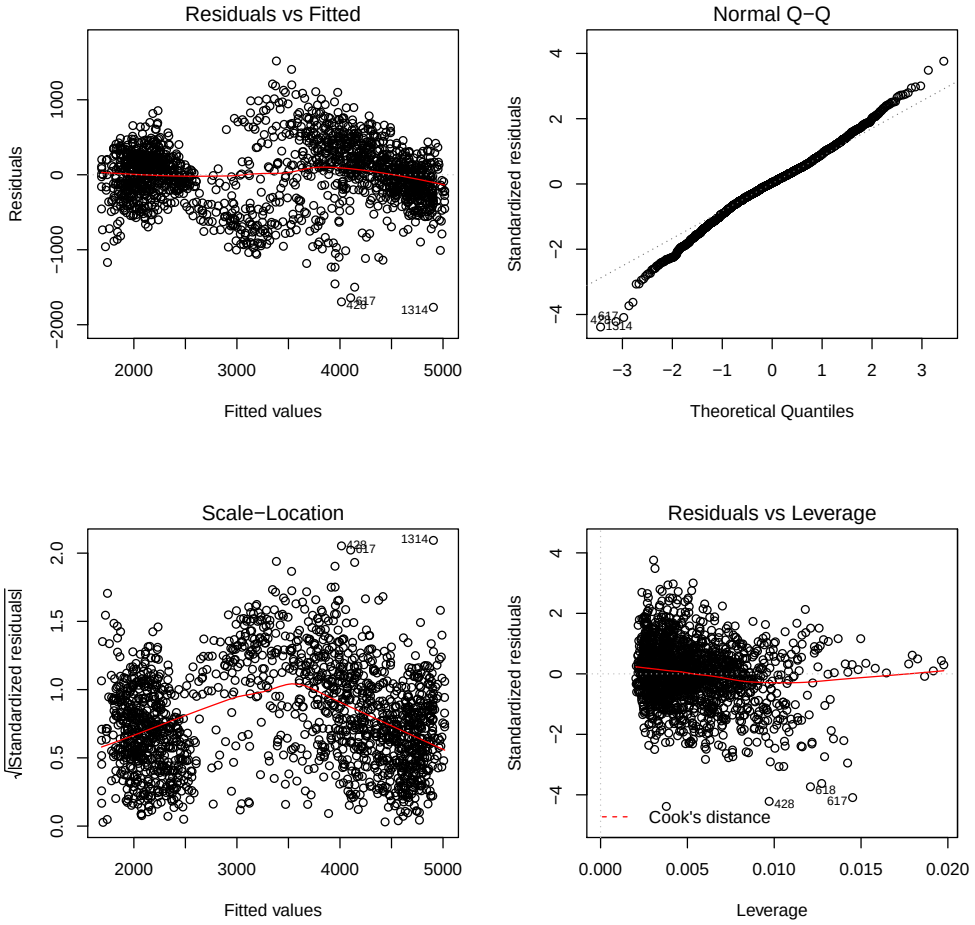


Figure 5.4: Diagnostic plots for `model.lm`.

Next, we look at the semi-variogram for the residuals of `model.lm`, see Figure 5.7. Here we can see an increase, hence we can suspect that the residuals are not independent but it is not easy to have an idea if the residuals are significantly independent or not. Some tests will be necessary.

```
library(gstat)
vgm <- variogram(resmodel.lm~1, Xutm, cutoff = 350)
plot(vgm)
```

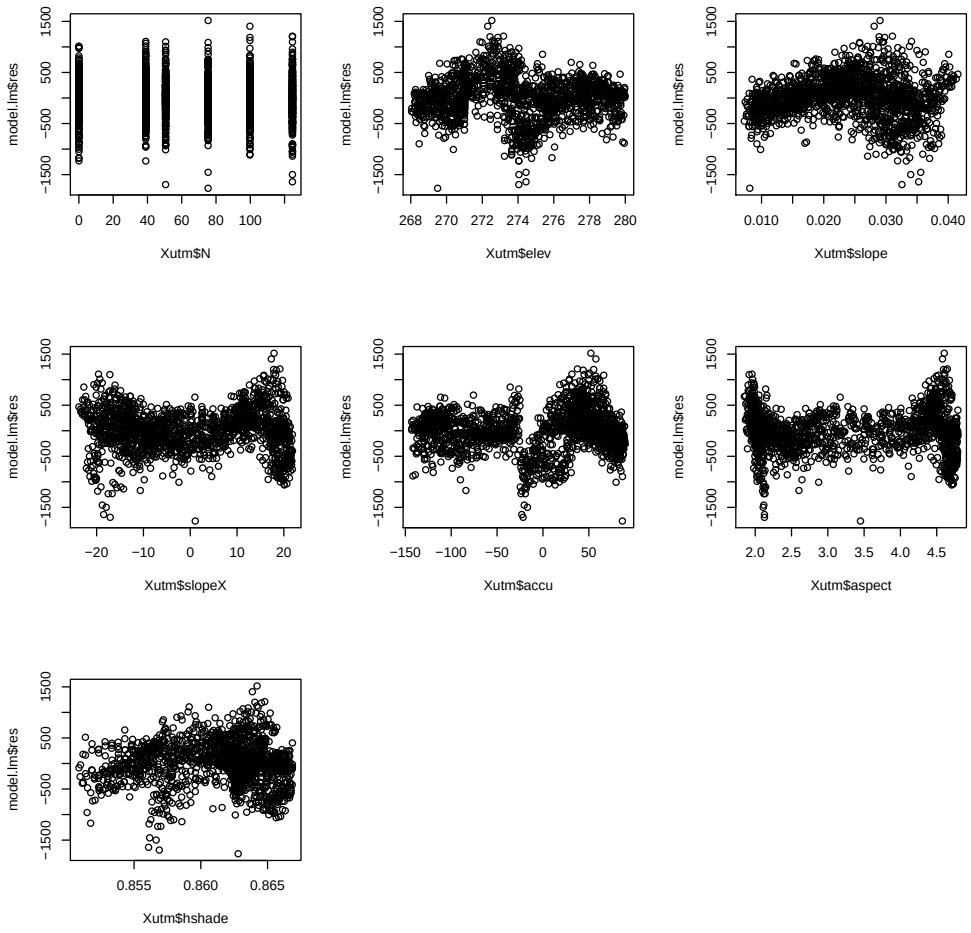


Figure 5.5: Residuals of `model.lm` against every possible explanatory variable.

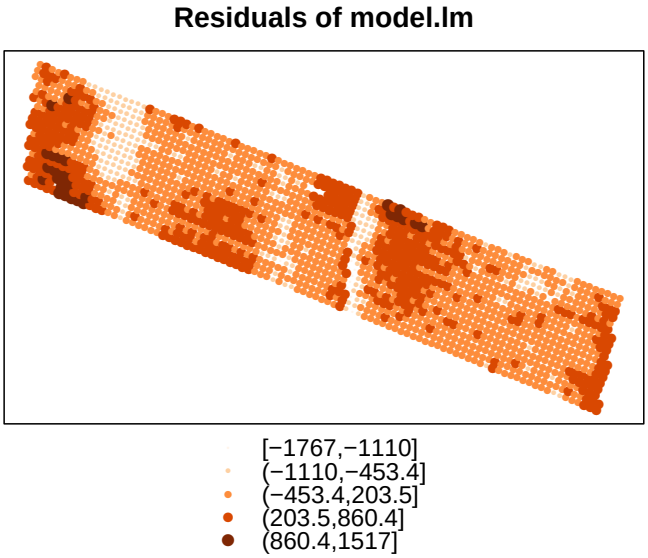


Figure 5.6: Bubble map for residuals of `model.lm`.

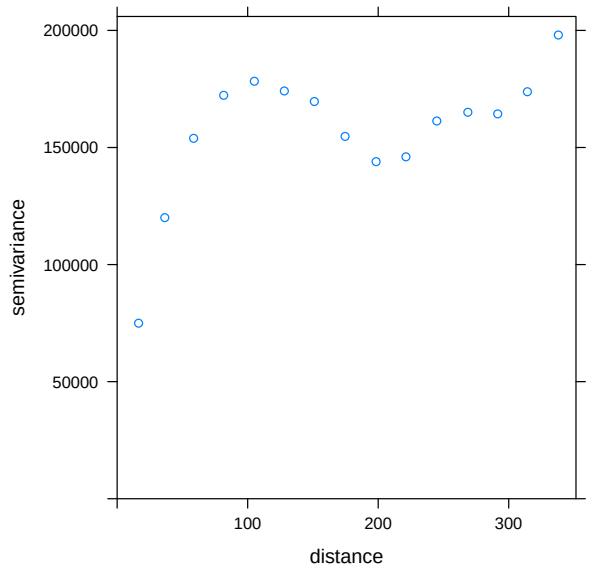


Figure 5.7: Semi-variogram for the residuals of `model.lm`.

REMARK 5.2.1 *Note that in R what is called 'variogram' is in reality the semi-variogram! It is $\gamma(.)$ which is plotted when using the functions* `Variogram` *or* `variogram`*, see sections 4.7.1 and 4.7.2.*

Similarly to the semi-variogram, we can represent the Moran correlogram (see section 4.6 for the definition of the Moran Correlogram). It gives a measurement of the change in the correlation structure as distance between cells is increased. The value of the spatial lag at which I is no longer significantly positive can be used as an indication of the range of autocorrelation of the data.

In the following code we create a list of neighbors using the k-nearest neighbors method, then we compute and plot the Moran correlogram, for a maximum lag of 10, and for a row-standardised weights matrix (see Figure 5.8). There is evidence of spatial autocorrelation. The largest lag for which the Moran's randomisation test (the default test for the `sp.correlogram` function) would reject the null hypothesis of no spatial autocorrelation (for a significance level $\alpha = 0.05$) is $k = 9$. Thus, it would be advisable to define the original neighbours list in a less restrictive way than was done for `nlist`.

```
library(spdep)
nlist <- knn2nb(knearneigh(Xutm,k=4))
I.d <- sp.correlogram(nlist,Xutm$resmodel.lm,order=10,method="I", style="W")
plot(I.d)
```

Finally, we present the test whether or not the residuals of `model.lm` are spatially autocorrelated. We first create a list of neighbors and an associated spatial weights matrix W (row-standardised). To create the list of neighbors, we can use the k-nearest neighbors method (using `knn2nb`). Then we can test for spatial autocorrelation using `lm.morantest`. Here, the alternative tested is that the moran statistic is greater than the expected value (hence we suspect a positive autocorrelation, and not a negative autocorrelation). The test is based on the resampling assumption.

```
library(spdep)
nlist <- knn2nb(knearneigh(Xutm,k=8))
W <- nb2listw(nlist,style="W")
```

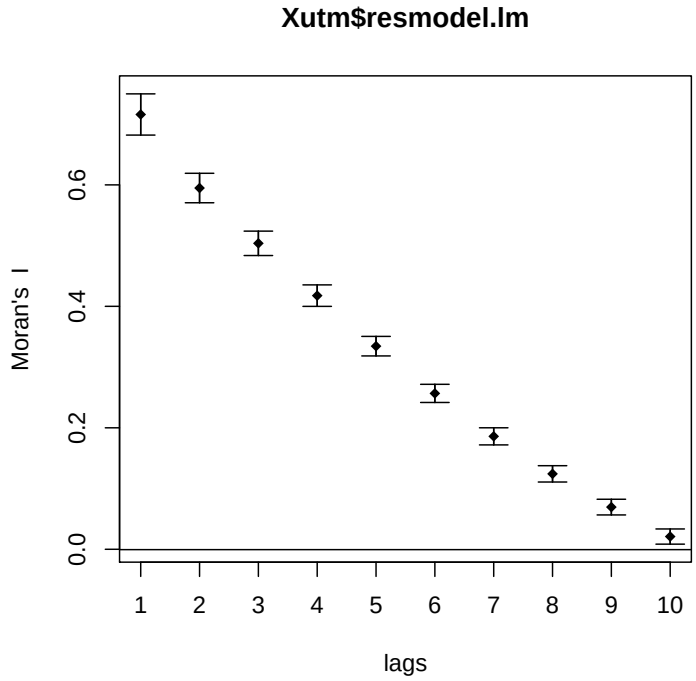


Figure 5.8: Moran correlogram for residuals of `model.lm`.

```
lm.morantest(model.lm,W)


Global Moran I for regression residuals

data:
model: lm(formula = YIELD ~ accuf + N + slope + aspect + hshade +
accuf * slope + accuf * aspect + accuf * hshade, data = Xutm@data)
weights: W

Moran I statistic standard deviate = 56.299, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Observed Moran I      Expectation         Variance
    0.6629755568    -0.0046606672     0.0001406322
```

The Moran's I is 0.66, while the expected value is -0.005. The p-value is $<$ 2.2e-16, hence we reject the null hypothesis and accept the alternative, that is the spatial autocorrelation of the residuals of `model.lm`.

We will then have to take into account this spatial autocorrelation of the residuals in our modelisation.

In the following we will present models designed specially for spatially autocorrelated data.

5.3 Spatial Lag Model

5.3.1 Without explanatory variable

A spatial lag model with zero mean value and no explanatory variable has the form:

$$\begin{aligned} Y &= \rho WY + \epsilon \\ \epsilon &\sim \mathcal{N}_n(0, \sigma^2 I), \end{aligned} \tag{5.8}$$

where WY represents the spatial lag.

Interpretation The value of Y at one location is directly associated with the values of the process Y at nearby locations. For instance high productiovity of a plant at one location is associated with high productivity at nearby locations (but there is no notion of causality).

5.3.2 With explanatory variables

If we want to include explanatory variables, the model becomes:

$$\begin{aligned} Y &= \rho WY + X\beta + \epsilon \\ \epsilon &\sim \mathcal{N}_n(0, \sigma^2 I). \end{aligned} \tag{5.9}$$

Interpretation This model can be interpreted using different points of view.

1. We are interested in the model for its own sake: specification of the spatial weights matrix W and estimation of ρ are then indicators of the nature and strength of spatial interaction.

2. We have $Y = (I - \rho W)^{-1}(X\beta + \epsilon)$, and $\mathbb{E}(Y) = (I - \rho W)^{-1}X\beta$. In this formulation, we are interested by the non-linear effect of the spatial autocorrelation on the expected value of Y. The influence of the spatial structure is modelled through the error term and through the explanatory variables (influence of the neighborhood through the explanatory variables).

The prediction $\hat{Y} = (I - \hat{\rho}W)^{-1}X\hat{\beta}$ is mainly driven by the neighborhood. If we use the formula $\hat{Y} = X\hat{\beta}$ (like for the classical linear model), we can see that we have a bias $-(\rho W)^{-1}X\beta$.

5.3.3 About the variance-covariance matrix of Y

Using this model the variance-covariance matrix of Y is the following:

$$\begin{aligned} var(Y) &= E\big[(Y - E[Y])(Y - E[Y])'\big] \\ &= E\big[(I - \rho W)^{-1}\epsilon\epsilon'((I - \rho W)^{-1})'\big] \\ &= (I - \rho W)^{-1}E[\epsilon\epsilon'](I - \rho W')^{-1} \\ &= (I - \rho W)^{-1}var[\epsilon](I - \rho W')^{-1} \\ &= \sigma^2(I - \rho W)^{-1}(I - \rho W')^{-1}. \end{aligned} \tag{5.10}$$

This variance-covariance matrix is impacted by the magnitude of the variance of the error term σ^2, and by the spatial structure through the term $(I - \rho W)^{-1}(I - \rho W')^{-1}$.
Note that this variance-covariance matrix is enforced by the model, we do not have to specify it. The spatial autocorrelation structure of Y is then enforced by the model.

5.3.4 Fitting the model

The parameters of the model are β, σ^2 and ρ. They will be estimated using the maximum likelihood approach. However, the expressions of $\hat{\beta}$, $\hat{\sigma^2}$ and $\hat{\rho}$ that maximise the likelihood are not easy to obtain (it would be much easier if ρ was known). The approach is therefore to use a numerical scheme analogous to the Newton-Raphson method:

- A value of $\hat{\rho}$ is fixed.
- The maximum likelihood estimates $\hat{\beta}$ and $\hat{\sigma^2}$ are calculated with $\hat{\rho}$ fixed.
- The two preceding steps are iterated: another value of $\hat{\rho}$ increasing the likelihood is fixed, $\hat{\beta}$ and $\hat{\sigma^2}$ are calculated to maximise the likelihood, then fix $\hat{\rho}$ again,...

```
library(spdep)
library(spatialreg)
nlist <- knn2nb(knearneigh(Xutm,k=8))
W <- nb2listw(nlist,style="W")
summary(Xutm@data[,1:8])
```

```
     YIELD              N               elev           slope
 Min.   : 574.2   Min.   :  0.00   Min.   :268.0   Min.   :0.007235
 1st Qu.:2290.1   1st Qu.: 39.00   1st Qu.:271.0   1st Qu.:0.016802
 Median :3826.5   Median : 50.60   Median :273.6   Median :0.024580
 Mean   :3412.5   Mean   : 64.93   Mean   :273.7   Mean   :0.024072
 3rd Qu.:4511.8   3rd Qu.: 99.80   3rd Qu.:276.2   3rd Qu.:0.031336
 Max.   :5347.9   Max.   :124.60   Max.   :280.0   Max.   :0.041564
     slopeX              accu              aspect          hshade
 Min.   :-23.7891   Min.   :-143.137   Min.   :1.850   Min.   :0.8509
 1st Qu.:-11.8105   1st Qu.: -67.273   1st Qu.:2.108   1st Qu.:0.8589
 Median : -1.7972   Median :  20.225   Median :2.964   Median :0.8625
 Mean   :  0.2142   Mean   :  -4.171   Mean   :3.240   Mean   :0.8613
 3rd Qu.: 13.4638   3rd Qu.:  56.062   3rd Qu.:4.428   3rd Qu.:0.8637
 Max.   : 21.8369   Max.   :  89.600   Max.   :4.792   Max.   :0.8669
```

```
Xutm$YIELD_scaled <- (Xutm$YIELD-mean(Xutm$YIELD))/sd(Xutm$YIELD)
# Xutm$N_scaled <- (Xutm$N-mean(Xutm$N))/sd(Xutm$N)
Xutm$slope_scaled <- (Xutm$slope-mean(Xutm$slope))/sd(Xutm$slope)
# Xutm$aspect_scaled <- (Xutm$aspect-mean(Xutm$aspect))/sd(Xutm$aspect)
Xutm$hshade_scaled <- (Xutm$hshade-mean(Xutm$hshade))/sd(Xutm$hshade)
f <- as.formula("YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled
                + accuf*slope_scaled + accuf*aspect + accuf*hshade_scaled")
mod.lag <- lagsarlm(f,data=Xutm,listw=W)
```

The interpretation of the spatial lag model is that there is some interaction between corn plants.

REMARK 5.3.1 *If the* `YIELD`, `slope` *and* `hshade` *variables were not scaled, the R software would not succeed in computing the spatial error model, an error would be indicated* (`inversion of asymptotic covariance matrix failed`).

```
summary(mod.lag)


Call:lagsarlm(formula = f, data = Xutm, listw = W)

Residuals:
       Min         1Q     Median         3Q        Max
-1.3179579 -0.1180341  0.0029359  0.1081311  0.6812413

Type: lag
Coefficients: (asymptotic standard errors)
                          Estimate  Std. Error z value  Pr(>|z|)
(Intercept)             0.02004197  0.02669322  0.7508 0.4527573
accuflow               -0.20148029  0.05537151 -3.6387 0.0002740
N                       0.00165631  0.00011166 14.8340 < 2.2e-16
slope_scaled           -0.05422472  0.00865827 -6.2628 3.782e-10
aspect                 -0.01605523  0.00779494 -2.0597 0.0394273
hshade_scaled          -0.03632647  0.01541618 -2.3564 0.0184537
accuflow:slope_scaled   0.06974847  0.01518287  4.5939 4.351e-06
accuflow:aspect        -0.00077238  0.01448327 -0.0533 0.9574698
accuflow:hshade_scaled  0.06926005  0.01876857  3.6902 0.0002241

Rho: 0.8753, LR test value: 1919.8, p-value: < 2.22e-16
Asymptotic standard error: 0.01351
    z-value: 64.788, p-value: < 2.22e-16
Wald statistic: 4197.5, p-value: < 2.22e-16

Log likelihood: 346.279 for lag model
ML residual variance (sigma squared): 0.033335, (sigma: 0.18258)
Number of observations: 1704
Number of parameters estimated: 11
AIC: -670.56, (AIC for lm: 1247.2)
LM test for residual autocorrelation
test value: 32.674, p-value: 1.0897e-08
```

5.4 Spatial Error Model

5.4.1 Formulation

$$\begin{aligned} Y &= X\beta + \eta \\ \eta &= \lambda W\eta + \epsilon \\ \epsilon &\sim \mathcal{N}_n(0, \sigma^2 I). \end{aligned} \tag{5.11}$$

Interpretation

1. Using this modelisation, we use a classical linear model, but with a correlated structure for the error term. This autocorrelation is generally considered to be a nuisance: when studying this model the primary interest is often the relationship between the explanatory variables X and the response variable Y. The spatial autocorrelation is just taken into account through the error term.
2. For the spatial error model, the influence of the spatial structure is modelled only on the error term: $Y = X\beta + (I - \lambda W)^{-1}\epsilon$.

The prediction $\hat{Y} = X\hat{\beta}$ is driven by the values of the explanatory variables at the location for which we want the prediction. Be careful, to have an unbiased estimation of β, you must use the spatial error model and not the classical linear model if your data are driven by this spatial error model.

REMARK 5.4.1 *This model can be written as a classical linear model:*

$$\begin{aligned} Y - \lambda WY &= X\beta - \lambda WY + \eta \\ &= X\beta - \lambda W(X\beta + \eta) + \eta \\ &= X\beta - \lambda WX\beta + \epsilon \\ &= (X - \lambda WX)\beta + \epsilon \\ \tilde{Y} &= \tilde{X}\beta + \epsilon \end{aligned}$$

5.4.2 About the variance-covariance matrix of Y

For this spatial error model we have $Y = X\beta + (I - \lambda W)^{-1}\epsilon$ and $E[Y] = X\beta$. hence $Y - E[Y] = (I - \lambda W)^{-1}\epsilon$. We can then use exactly the same calculations as for the spatial

lag model, and we obtain the same formula for the variance covariance matrix (5.10), except that ρ is replaced by λ:

$$var(Y) = \sigma^2(I - \lambda W)^{-1}(I - \lambda W')^{-1}. \tag{5.12}$$

The same remarks can be made, especially that the spatial autocorrelation structure of Y is enforced by the model.

5.4.3 Fitting the model

The approach is the same as for the spatial lag model, with ρ replaced by λ.

```
mod.err <- errorsarlm(f,data=Xutm,listw=W)
```

```
summary(mod.err)
```

```
Call:errorsarlm(formula = f, data = Xutm, listw = W)

Residuals:
        Min          1Q      Median          3Q         Max
-1.22572240 -0.10186036  0.00092688  0.10447445  0.71212632

Type: error
Coefficients: (asymptotic standard errors)
                          Estimate  Std. Error z value  Pr(>|z|)
(Intercept)             1.13645551  0.28442272  3.9957 6.452e-05
accuflow               -1.34933604  0.43064505 -3.1333 0.0017286
N                       0.00196741  0.00011066 17.7797 < 2.2e-16
slope_scaled           -0.42523669  0.06866555 -6.1929 5.908e-10
aspect                 -0.33772997  0.08921225 -3.7857 0.0001533
hshade_scaled           0.36372642  0.12239612  2.9717 0.0029614
accuflow:slope_scaled   0.58286320  0.09749498  5.9784 2.254e-09
accuflow:aspect         0.15722109  0.13008315  1.2086 0.2268089
accuflow:hshade_scaled -0.09240629  0.14049648 -0.6577 0.5107229
```

```
Lambda: 0.93964, LR test value: 1934.2, p-value: < 2.22e-16
Asymptotic standard error: 0.009443
    z-value: 99.506, p-value: < 2.22e-16
Wald statistic: 9901.5, p-value: < 2.22e-16

Log likelihood: 353.489 for error model
ML residual variance (sigma squared): 0.031593, (sigma: 0.17774)
Number of observations: 1704
Number of parameters estimated: 11
AIC: -684.98, (AIC for lm: 1247.2)
```

The interpretation of the spatial error model is that the yield at one location is supposed mainly driven by the values of the explanatory variables at this location, and the error terms are spatially autocorrelated.

5.5 Choosing between Spatial Lag, Error and SAC models

Once spatial autocorrelation has been detected in residuals of a classical linear model, we have to take into account this autocorrelation and to choose between the spatial lag model and/or the spatial error model. These two models can be combined in a SAC/SARAR model of the form:

$$\begin{aligned} Y &= \rho W_1 Y + X\beta + \eta \\ \eta &= \lambda W_2 \eta + \epsilon \\ \epsilon &\sim \mathcal{N}_n(0, \sigma^2 I), \end{aligned} \tag{5.13}$$

where W_1 can be equal to W_2 and X cannot simply be a vector of ones (for identifiability). For more details on these models, you can read [8].

The problem with choosing between a spatial lag, a spatial error or a SAC model can be expressed as two hypothesis tests:

$$\text{1st test: } \begin{cases} H_0: & \rho = 0, \\ H_1: & \rho \neq 0 \end{cases} \quad \text{and} \quad \text{2nd test: } \begin{cases} H_0: & \lambda = 0, \\ H_1: & \lambda \neq 0 \end{cases}$$

- If H_0 is non-rejected for both tests, that means that we have to keep a classical linear model, there is no spatial autocorrelation of the residuals.

- If H_0 is non-rejected for the second test and H_1 accepted for the first test we choose a spatial lag model.
- If H_0 is non-rejected for the first test and H_1 accepted for the second test, we choose a spatial error model.
- If H_1 is accepted for one of these tests, we can try to fit a SAC model, and to compare it with a spatial lag or a spatial error model (spatial lag and spatial error model are nested into a SAC model).

These tests are carried out using the maximum likelihood approach. An alternative is the use the AIC criteria to choose between these models.

```
f <- as.formula("YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled
                + accuf*slope_scaled + accuf*aspect + accuf*hshade_scaled")
```

```
model.lm_scaled <- lm(f, data=Xutm)
mod.lag <- lagsarlm(f,data=Xutm,listw=W)
mod.err <- errorsarlm(f,data=Xutm,listw=W)
LR.sarlm(model.lm_scaled,mod.lag)


Likelihood ratio for spatial linear models

data:
Likelihood ratio = -1919.8, df = 1, p-value < 2.2e-16
sample estimates:
Log likelihood of model.lm_scaled         Log likelihood of mod.lag
                        -613.6159                           346.2790

LR.sarlm(model.lm_scaled,mod.err)


Likelihood ratio for spatial linear models

data:
```

```
Likelihood ratio = -1934.2, df = 1, p-value < 2.2e-16
sample estimates:
Log likelihood of model.lm_scaled         Log likelihood of mod.err
                        -613.6159                          353.4890

AIC(mod.lag,mod.err)

        df       AIC
mod.lag 11 -670.5580
mod.err 11 -684.9779
```

The p-values are very small for both spatial lag and spatial error models. They are both preferable to the classical linear model. Using the AIC criteria, we prefer the spatial error model. As the H_1 hypothesis is accepted for both tests, we can fit a SAC model and compare it to the spatial lag, or to the spatial error model.

```
mod.sac <- sacsarlm(f,data=Xutm,listw=W)
LR.sarlm(mod.sac,mod.lag)


Likelihood ratio for spatial linear models

data:
Likelihood ratio = 104.3, df = 1, p-value < 2.2e-16
sample estimates:
Log likelihood of mod.sac Log likelihood of mod.lag
                 398.4283                  346.2790

LR.sarlm(mod.sac,mod.err)


Likelihood ratio for spatial linear models

data:
Likelihood ratio = 89.879, df = 1, p-value < 2.2e-16
```

```
sample estimates:
Log likelihood of mod.sac Log likelihood of mod.err
                 398.4283                  353.4890

AIC(mod.lag,mod.err,mod.sac)

        df       AIC
mod.lag 11 -670.5580
mod.err 11 -684.9779
mod.sac 12 -772.8566
```

It appears that the spatial SAC model is quite better than the spatial lag or the spatial error model: we both have interaction between corn plants, and the error terms are spatially correlated.

```
summary(mod.sac)
```

```
Type: sac
Coefficients: (asymptotic standard errors)
                          Estimate  Std. Error z value  Pr(>|z|)
(Intercept)             1.75765011  0.77749616  2.2607  0.023781
accuflow               -1.16490866  0.61228927 -1.9025  0.057100
N                       0.00151935  0.00011262 13.4915 < 2.2e-16
slope_scaled           -0.17491748  0.13328369 -1.3124  0.189395
aspect                 -0.52703547  0.20278078 -2.5990  0.009348
hshade_scaled           0.38036320  0.21454549  1.7729  0.076249
accuflow:slope_scaled   0.18923146  0.14064994  1.3454  0.178494
accuflow:aspect         0.26730067  0.17766099  1.5046  0.132439
accuflow:hshade_scaled -0.26141594  0.18786214 -1.3915  0.164065
```

We can improve this SAC model, by performing model selection. We can try to remove explanatory variables or interactions between them and to include variables which are not present in `mod.sac`. Here we do not have variables not included in the model. But we can try to remove some interaction. In particular, when looking at the summary of `mod.sac`, it appears that the interactions `accu*slope`, `accu*hshade` and `accu*aspect` are not significant. We can then try to simplify the model by first removing `accu*slope`.

```
f2 <- as.formula("YIELD_scaled ~ accuf + N + slope_scaled + aspect
                 + hshade_scaled + accuf*aspect + accuf*hshade_scaled")
mod.sac2 <- sacsarlm(f2,data=Xutm,listw=W)
```

```
LR.sarlm(mod.sac, mod.sac2)
```

```
Likelihood ratio for spatial linear models

data:
Likelihood ratio = 1.6409, df = 1, p-value = 0.2002
sample estimates:
 Log likelihood of mod.sac Log likelihood of mod.sac2
                  398.4283                   397.6078
```

```
summary(mod.sac2)
```

```
Type: sac
Coefficients: (asymptotic standard errors)
                          Estimate  Std. Error z value Pr(>|z|)
(Intercept)             1.66913033  0.81695206  2.0431  0.04104
accuflow               -0.99822066  0.60807547 -1.6416  0.10067
N                       0.00151476  0.00011243 13.4726  < 2e-16
slope_scaled           -0.10539026  0.12812374 -0.8226  0.41075
aspect                 -0.50983334  0.20672089 -2.4663  0.01365
hshade_scaled           0.36139428  0.21795862  1.6581  0.09730
accuflow:aspect         0.28738455  0.17766510  1.6176  0.10576
accuflow:hshade_scaled -0.30201642  0.18562952 -1.6270  0.10374
```

Using a Likelihood-Ratio test the model `mod.sac2` is better than `mod.sac`. Then, by looking at the summary we see that we can continue to simplify the model, by removing the interaction `accu*aspect`.

```
f3 <- as.formula("YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled +
mod.sac3 <- sacsarlm(f3,data=Xutm,listw=W)
```

```
LR.sarlm(mod.sac2, mod.sac3)
```

```
Likelihood ratio for spatial linear models

data:
Likelihood ratio = 2.572, df = 1, p-value = 0.1088
sample estimates:
Log likelihood of mod.sac2 Log likelihood of mod.sac3
                  397.6078                   396.3219
```

```
summary(mod.sac3)
```

```
Type: sac
Coefficients: (asymptotic standard errors)
                         Estimate Std. Error z value Pr(>|z|)
(Intercept)             1.1470900  0.7703550  1.4890  0.13648
accuflow               -0.0170678  0.0496118 -0.3440  0.73083
N                       0.0015136  0.0001124 13.4661  < 2e-16
slope_scaled           -0.0824190  0.1285267 -0.6413  0.52135
aspect                 -0.3345291  0.1766718 -1.8935  0.05829
hshade_scaled           0.1320709  0.1711735  0.7716  0.44037
accuflow:hshade_scaled -0.0085828  0.0390317 -0.2199  0.82595
```

The interaction `accu*aspect` is at the limit of the significant level (p-value of 0.10), but we can remove it. By looking at the summary of `mod.sac3`, we can try to remove the last interaction `accu*hshade`.

```
f4 <- as.formula("YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled")
mod.sac4 <- sacsarlm(f4,data=Xutm,listw=W)
```

```
LR.sarlm(mod.sac3, mod.sac4)
```

```
Likelihood ratio for spatial linear models

data:
Likelihood ratio = 0.04834, df = 1, p-value = 0.826
sample estimates:
Log likelihood of mod.sac3 Log likelihood of mod.sac4
                  396.3219                   396.2977
```

```
summary(mod.sac4)
```

```
Type: sac
Coefficients: (asymptotic standard errors)
                 Estimate  Std. Error z value Pr(>|z|)
(Intercept)    1.14654210  0.76990060  1.4892  0.13643
accuflow      -0.01694282  0.04960867 -0.3415  0.73271
N              0.00151400  0.00011239 13.4715  < 2e-16
slope_scaled  -0.08138525  0.12841259 -0.6338  0.52622
aspect        -0.33388599  0.17662274 -1.8904  0.05871
hshade_scaled  0.12672191  0.16932172  0.7484  0.45421
```

The interaction interaction `accu*hshade` can indeed be removed. Then, by looking at the summary of `mod.sac4`, we see that we can remove `accuf`.

```
f5 <- as.formula("YIELD_scaled ~ N + slope_scaled + aspect + hshade_scaled")
mod.sac5 <- sacsarlm(f5,data=Xutm,listw=W)
```

```
LR.sarlm(mod.sac4, mod.sac5)
```

```
Likelihood ratio for spatial linear models

data:
Likelihood ratio = 0.11589, df = 1, p-value = 0.7335
sample estimates:
Log likelihood of mod.sac4 Log likelihood of mod.sac5
                  396.2977                   396.2397
```

```
summary(mod.sac5)
```

```
Type: sac
Coefficients: (asymptotic standard errors)
                 Estimate  Std. Error z value Pr(>|z|)
(Intercept)    1.14286973  0.77325648  1.4780  0.13941
N              0.00151338  0.00011237 13.4679  < 2e-16
slope_scaled  -0.08042127  0.12851282 -0.6258  0.53146
aspect        -0.33420918  0.17686740 -1.8896  0.05881
hshade_scaled  0.12711273  0.16947486  0.7500  0.45323
```

The variable `accuf` can indeed be removed. Then by looking at the summary of `mod.sac5`, we see that we can remove `slope_scaled`.

```
f6 <- as.formula("YIELD_scaled ~ N + aspect + hshade_scaled")
mod.sac6 <- sacsarlm(f6,data=Xutm,listw=W)
```

```
LR.sarlm(mod.sac5, mod.sac6)
```

```
Likelihood ratio for spatial linear models
```

```
data:
Likelihood ratio = 0.37328, df = 1, p-value = 0.5412
sample estimates:
Log likelihood of mod.sac5 Log likelihood of mod.sac6
                  396.2397                   396.0531
```

```
summary(mod.sac6)
```

```
Type: sac
Coefficients: (asymptotic standard errors)
                 Estimate  Std. Error z value Pr(>|z|)
(Intercept)    1.26179451  0.77180194  1.6349  0.10208
N              0.00151327  0.00011229 13.4770  < 2e-16
aspect        -0.36893656  0.16958743 -2.1755  0.02959
hshade_scaled  0.14790879  0.16488224  0.8971  0.36969
```

The variable `slope_scaled` can indeed be removed. Then by looking at the summary of `mod.sac6`, we see that we can remove `hshade_scaled`.

```
f7 <- as.formula("YIELD_scaled ~ N + aspect")
mod.sac7 <- sacsarlm(f7,data=Xutm,listw=W)
```

```
LR.sarlm(mod.sac6, mod.sac7)


Likelihood ratio for spatial linear models

data:
Likelihood ratio = 0.73526, df = 1, p-value = 0.3912
sample estimates:
Log likelihood of mod.sac6 Log likelihood of mod.sac7
                  396.0531                   395.6855
```

```
summary(mod.sac7)
```

```
Type: sac
Coefficients: (asymptotic standard errors)
               Estimate  Std. Error z value Pr(>|z|)
(Intercept)  1.23727293  0.80598355  1.5351  0.12476
N            0.00150883  0.00011217 13.4515  < 2e-16
aspect      -0.35675187  0.17054142 -2.0919  0.03645
```

The variable `hshade_scaled` can indeed be removed. By looking at the summary of `mod.sac7`, we can see that an other simplification of this model is not advisable (the p-values are all smaller than the limit of the threshod commonly used of 5%). Hence we choose the model `mod.sac7`. The same conclusion is obtained using the AIC.

```
AIC(mod.sac,mod.sac2,mod.sac3,mod.sac4,mod.sac5,mod.sac6,mod.sac7)
```

```
         df       AIC
mod.sac  12 -772.8566
mod.sac2 11 -773.2157
mod.sac3 10 -772.6437
mod.sac4  9 -774.5954
mod.sac5  8 -776.4795
mod.sac6  7 -778.1062
mod.sac7  6 -779.3710
```

This model is quite simple compared to the classical linear model obtained before! This is because we have taken into account the spatial autocorrelation. From an agronomic point of view, we are surprised that the variable corresponding to water accumulation is not kept in the model. Maybe, the effect of water accumulation is mimiced by the spatial lag (that is the mean yield of the neighborhood in our case with a row-standardised spatial weight matrix). Indeed, you can see on Figure 5.1 that the yield and the water accumulation give very similar maps.

In the following, we check that the assumptions are verified on the residuals of `mod.sac7`. First, the normality assumption seems to be verified. We have slight deviations form the qqline on the tails, but it is acceptable, see Figure 5.9.

```
par(mfrow=c(1,2))
hist(mod.sac7$residuals)
qqnorm(mod.sac7$residuals)
qqline(mod.sac7$residuals)
```

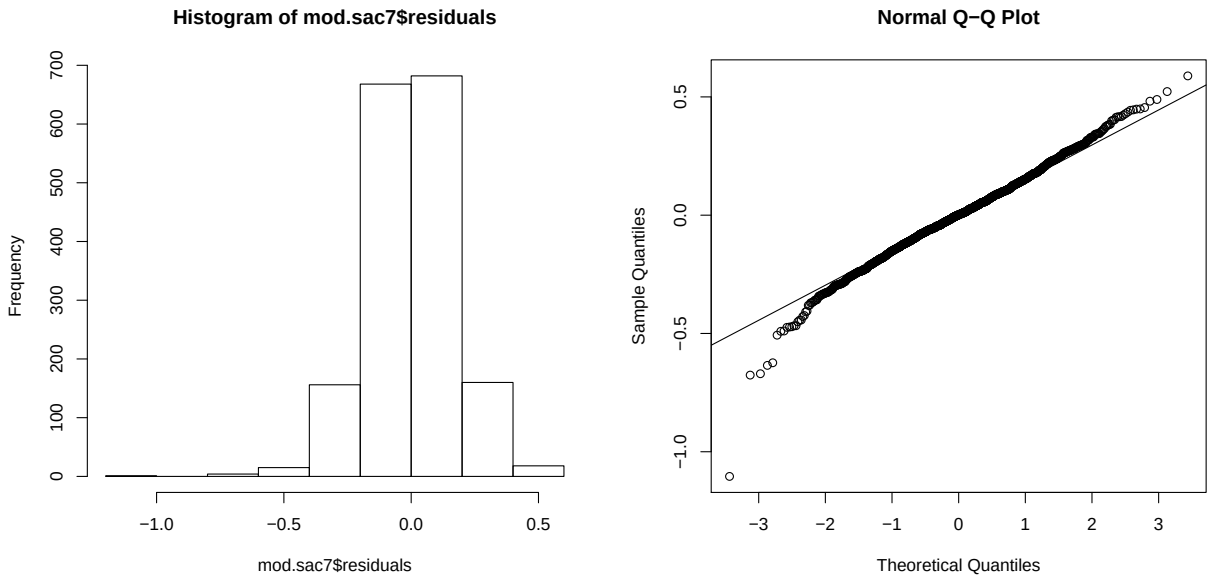


Figure 5.9: Histogram of residuals of `mod.sac7` and associated QQ plot.

Concerning the homoscedasticity, the variance do not seem to increase with fitted values or with another variable, see Figure 5.10.

```
Xutm$res.sac7 <- mod.sac7$residuals
par(mfrow=c(3,3))
plot(res.sac7 ~ fitted(mod.sac7), data=Xutm)
plot(res.sac7 ~ N, data=Xutm)
plot(res.sac7 ~ elev, data=Xutm)
plot(res.sac7 ~ slope, data=Xutm)
plot(res.sac7 ~ slopeX, data=Xutm)
plot(res.sac7 ~ accu, data=Xutm)
plot(res.sac7 ~ aspect, data=Xutm)
plot(res.sac7 ~ hshade, data=Xutm)
```

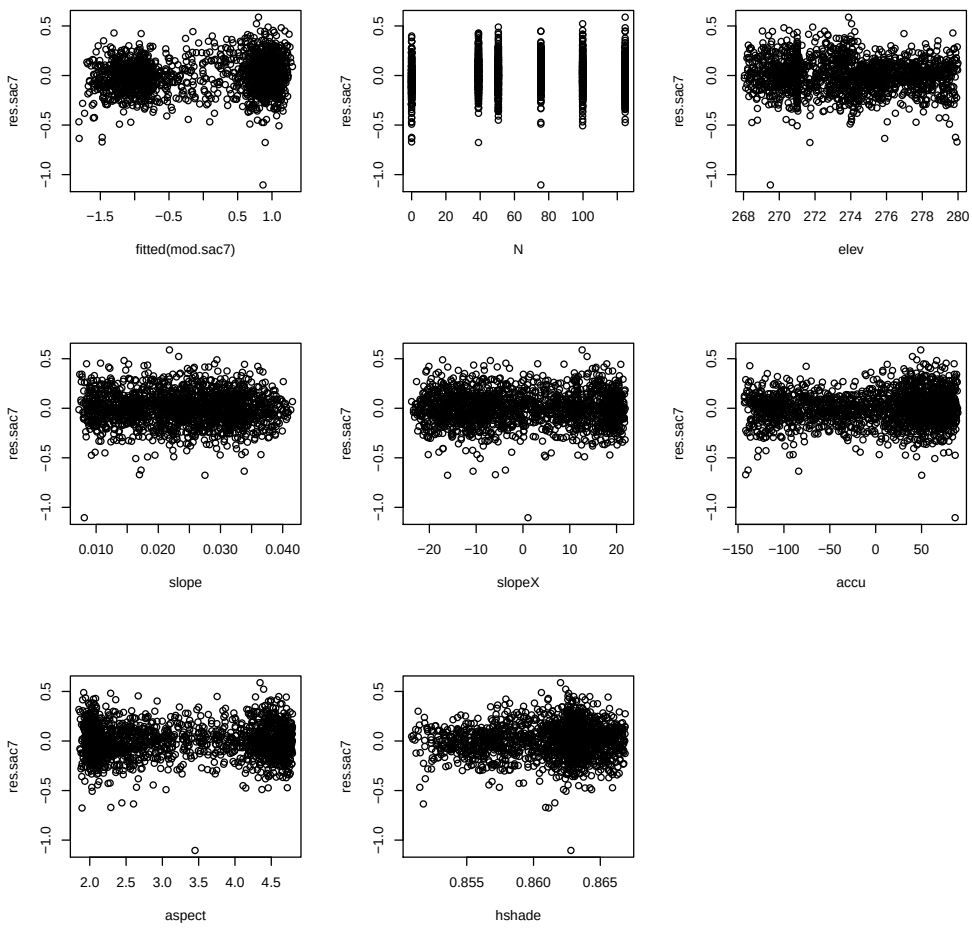


Figure 5.10: Residuals of `mod.sac7` against every possible explanatory variable.

We then have a look at the spatial organization of the residuals, see Figure 5.11. This figure is quite better than the one obtained on the classical linear model, see Figure 5.6.

```
Xutm$res.sac7 <- mod.sac7$residuals
spplot(Xutm, "res.sac7", col.regions=brewer.pal(9,"Oranges"),
            cex=.2*(1:5), aspect=1/2, main="Residuals of mod.sac7")
```

To be sure that the residuals of `mod.sac7` are not spatially autocorrelated we can have a look at the semi-variogram, see Figure 5.12. Here again it is quite good, we do not detect any increase at the beginning of the semi-variogram. It was not the case for the classical

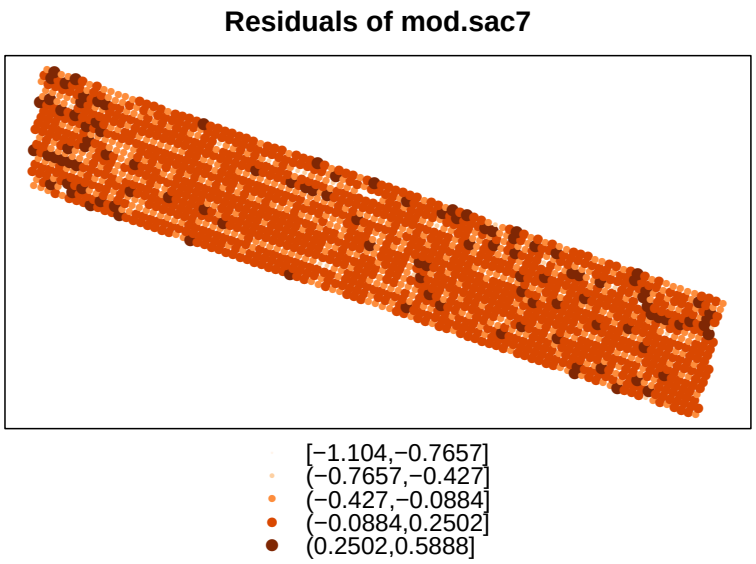


Figure 5.11: Bubble map for residuals of `mod.sac7`.

linear model, see Figure 5.7.

```
vgm <- variogram(res.sac7~1, cutoff=150, Xutm)
plot(vgm)
```

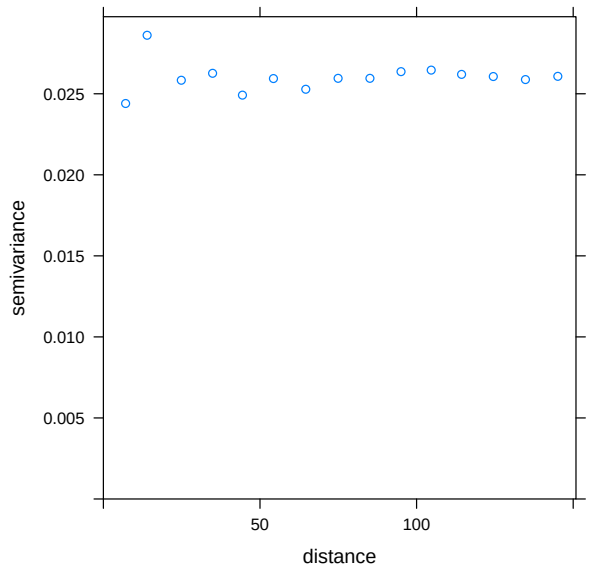


Figure 5.12: Semi-variogram for the residuals of `mod.sac7`.

We can also represent the Moran's correlogram, see Figure 5.13.

```
nlist <- knn2nb(knearneigh(Xutm,k=8))
I.d2 <- sp.correlogram(nlist,Xutm$res.sac7,order=10,method="I", style="W")
plot(I.d2)
```

Some confidence intervals for the values of this Moran correlogram do not include zero. However, we can note that:

1. This Moran correlogram is sensitive to outliers. And on Figure 5.9 we can note that there are some outliers.

2. We have lots of observations (more than 1700). Hence the tests performed are quite powerful, and the confidence intervals are quite accurate (maybe too much, sometimes a difference can be statistically significant but biologically negligeable).

3. If we look at the Moran correlogram associated to residuals from the classical linear model `model.lm` (Figure 5.8), we can see that the Moran's statistics are quite smaller (between -0.05 and 0.05 versus 0.8), hence the improvement is important.

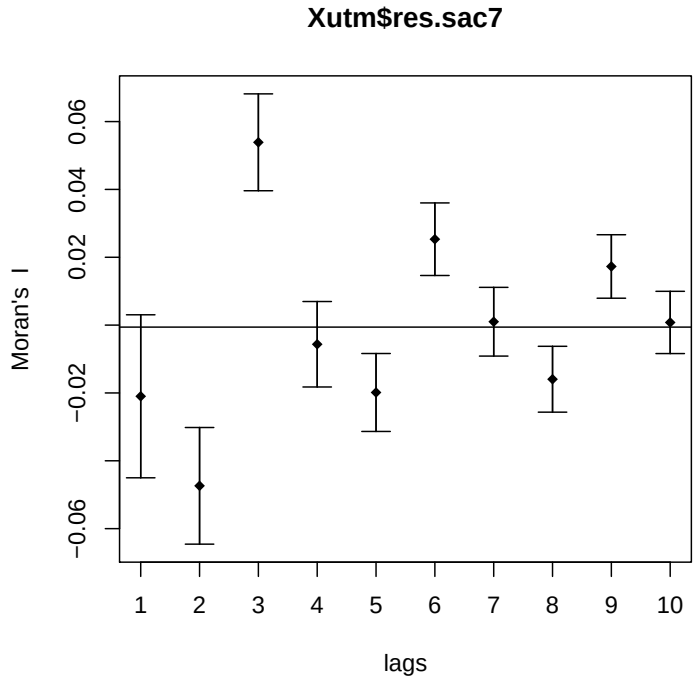


Figure 5.13: Moran correlogram for residuals of `mod.sac7`.

Eventually, we present the test whether or not the residuals of `mod.sac7` are spatially autocorrelated. Here we can see that they can not be considered as positively autocorrelated. Interestingly, they can be considered as negatively autocorrelated (we are at the limit of the threshold of 5%). It is coherent with the first values observed on the Moran's correlogram, which are negative. However, as explained before, the Moran's I around -0.02 or -0.05 for the first lags are quite small compared to the values observed for the classical linear model, so the improvement of using a spatial SAC model is important.

```
moran.mc(Xutm@data$res.sac7,W,nsim=1000,alternative="greater")


Monte-Carlo simulation of Moran I

data:  Xutm@data$res.sac7
weights: W
number of simulations + 1: 1001
```

```
statistic = -0.020985, observed rank = 53, p-value = 0.9471
alternative hypothesis: greater

moran.mc(Xutm@data$res.sac7,W,nsim=1000,alternative="less")


Monte-Carlo simulation of Moran I

data:  Xutm@data$res.sac7
weights: W
number of simulations + 1: 1001

statistic = -0.020985, observed rank = 35, p-value = 0.03497
alternative hypothesis: less
```

We can obtain fitted values of yield using this spatial SAC model (predictions of yield for the data on which the model was fitted), see Figure 5.14.

```
head(fitted(mod.sac7))


        1         2         3         4         5         6
0.8811450 0.8716522 0.8747162 0.9147915 0.9467749 0.9179487
```

```
Xutm@data$fitted <- (fitted(mod.sac7)*sd(Xutm$YIELD))+mean(Xutm$YIELD)
spplot(Xutm, "fitted", col.regions=brewer.pal(9,"Oranges"),
             cex=.2*(1:5), aspect=1/2, main="Fitted values of yield using the spatial
```

A question of interest is about predictions for new data. For instance, imagine that we want to predict the yield in the field if the nitrogen content is increased uniformely by one unit (using fertilizer). For such predictions, we need to be very careful, as with any statistical model. Indeed, our model is valid only for the ranges of values observed in our dataset for all variables. Moreover, yield depends on water and nitrogen with interaction between them (too much nitrogen without enough water can impact negatively the plant). This kind of

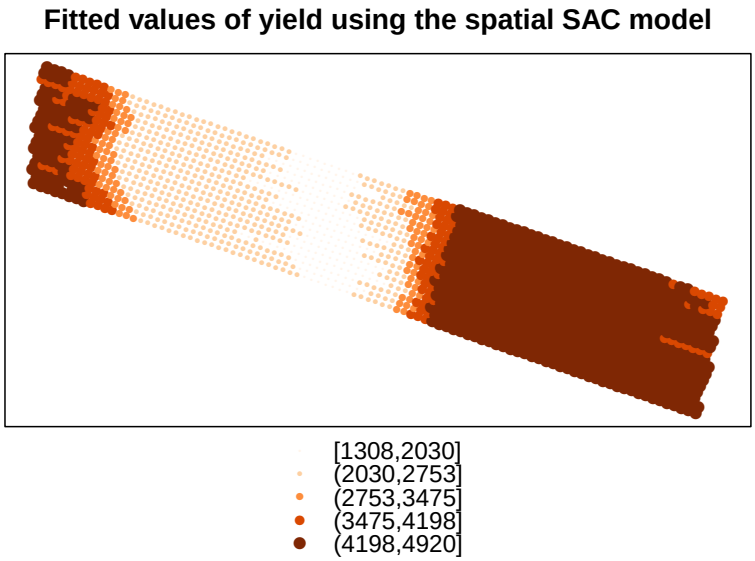


Figure 5.14: Bubble map for the fitted yield for the spatial SAC model.

biological effects are not taken into account by our model, as in the dataset such conditions were not observed. Additionally, the final model was selected using AIC and likelihood ratio tests on an explanatory basis, and not on a predictive basis. That means that it is maybe not the best model for prediction. To select such a model, cross-validation should be used. If you still want to make some predictions on the observed field but for new values of the predictors, you should remember that a spatial process can be decomposed into the sum of a deterministic trend, a spatially autocorrelated random process and an uncorrelated random process (see section). Looking at the spatial lag model, the predictions are given by the following formula:

$$\hat{Y} = (I - \hat{\rho}W)^{-1} X\hat{\beta}$$

In this formula the spatial autocorrelated part is taken into account. Concerning the spatial error model, the predictions are given by the following formula:

$$\hat{Y} = X\hat{\beta}$$

Indeed, the spatial autocorrelated part (the signal) is given by $(I - \lambda W)^{-1}\epsilon$, and is predicted by 0 as ϵ is predicted as 0. Hence, we have the same formula than in a classical linear model, the difference is that β is not estimated in the same way. The consequence is that the autocorrelated spatial part of the process is estimated by 0, we "loose" the spatial autocorrelation, only the deterministic trend is predicted. Concerning the spatial SAC model we

observe the same thing, only the trend can be safely estimated, the spatially autocorrelated part of the process is not estimated.
As a consequence, our advice is to use the `predict.sarlm` function only for spatial lag models, as they are the only model able to predict a non-null autocorrelated spatial part of the process. In the following, only predictions for a spatial lag model are presented. We will predict the yield in the field if the nitrogen content is increased uniformely by one unit (using fertilizer). As explained before, these predictions should be interpreted carefully.

```
Xutm2 <- Xutm
Xutm2@data$N <- Xutm@data$N + 1
newpred <- as.data.frame(predict.sarlm(mod.lag, newdata=Xutm2@data,
                                       listw = W, pred.type="TS"))
head(newpred)
```

```
        fit      trend    signal
1 0.4821111 0.14613577 0.3359753
2 0.4546835 0.13409381 0.3205896
3 0.4286981 0.12223805 0.3064601
4 0.3613864 0.11290496 0.2484815
5 0.3086342 0.10375643 0.2048778
6 0.2627418 0.09649667 0.1662452
```

In the R output, the `trend` is the deterministic trend, the `signal` is the spatially autocorrelated part, and the `noise` is the residual. The `fit` is the sum of the `trend` and the `signal`, the noise is estimated by 0. This fit corresponds to the previsions we want to make. These predictions are represented in Figure 5.15.

```
Xutm@data$pred.lag <- (newpred$fit*sd(Xutm$YIELD))+mean(Xutm$YIELD)
spplot(Xutm, "pred.lag", col.regions=brewer.pal(9,"Oranges"),
             cex=.2*(1:5), aspect=1/2, main="Predicted values of yield using mod.lag"
```

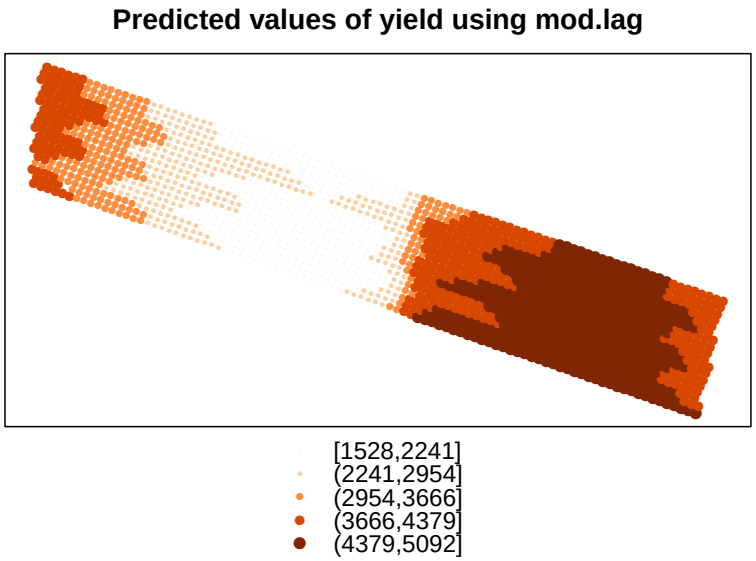


Figure 5.15: Bubble map for the predicted yield for the model `mod.lag` when the nitrogen content is increased by one unit uniformly.

5.6 Extended Linear Models

5.6.1 Classical Linear Model versus Extended Linear Model

Let Y be the quantitative variable to explain. The explanatory variables can be quantitative or qualitative, and are given in a matrix X. The classical linear model can be written as follows:

$$Y = X\beta + \epsilon \qquad \text{with} \quad \epsilon \sim \mathcal{N}_n(0, \sigma^2 I). \tag{5.14}$$

Possible extensions of this linear model concern the variance-covariance matrix of the residuals. Indeed, in classical linear models like (5.14), the residuals (therefore the observations) are supposed independent and homoscedastic. Concerning the independence, you can note that all the correlation terms of the variance-covariance matrix are null (the values outside the diagonal). Concerning the homoscedasticity, you can note that all the variance terms of the variance-covariance matrix are the same (the values on the diagonal).

In extended linear models (also called Generalized Least Squares models), the form of the variance-covariance matrix can be different, to take into account heteroscedasticity and/or

non-independance of the residuals. These models are as follows:

$$Y = X\beta + \epsilon \qquad \text{with} \quad \epsilon \sim \mathcal{N}_n(0, \Lambda). \tag{5.15}$$

1. If Λ is diagonal, but with varying coefficients on the diagonal, heteroscedasticity will be taken into account.
2. If Λ has non-null coefficients outside the diagonal, correlation between the residuals will be taken into account, that is the dependence structure of the residuals will be taken into account. This dependence can be temporal, spatial or more general.

5.6.2 Modelling Spatial Correlation

The extended linear model for spatial data (5.15) is more general than the regression models for spatially autocorrelated data presented in sections 5.3 and 5.4. Indeed, the models for spatially autocorrelated data are special cases of extended linear models. To compare the two approaches, you can note that in extended linear models the variance-covariance matrix Λ can take any form (it just needs to be symmetric and positive-definite). In the regression models designed for spatially autocorrelated data, the form of the variance-covariance matrix is enforced by the model.

To model spatial dependency using an extended linear model, we need to choose the form of the variance-covariance matrix Λ, which is equivalent to choose a model for the semi-variogram. There are two possibilities for choosing the form of the semi-variogram:

- This choice can be made by looking at the form of the semi-variogram. Figure 5.16 shows different semi-variogram patterns with different ranges. The formulae associated with these patterns are given in Table 4.1.
- Choosing the model by looking at plots can be difficult and subjective. Another option is to choose a model using classical model selection methods: AIC, BIC, or tests between nested models.

Once a model has been chosen for the semi-variogram, the parameters of the extended linear model (regression coefficients and coefficients of the variance-covariance matrix) are estimated using the maximum likelihood estimators.

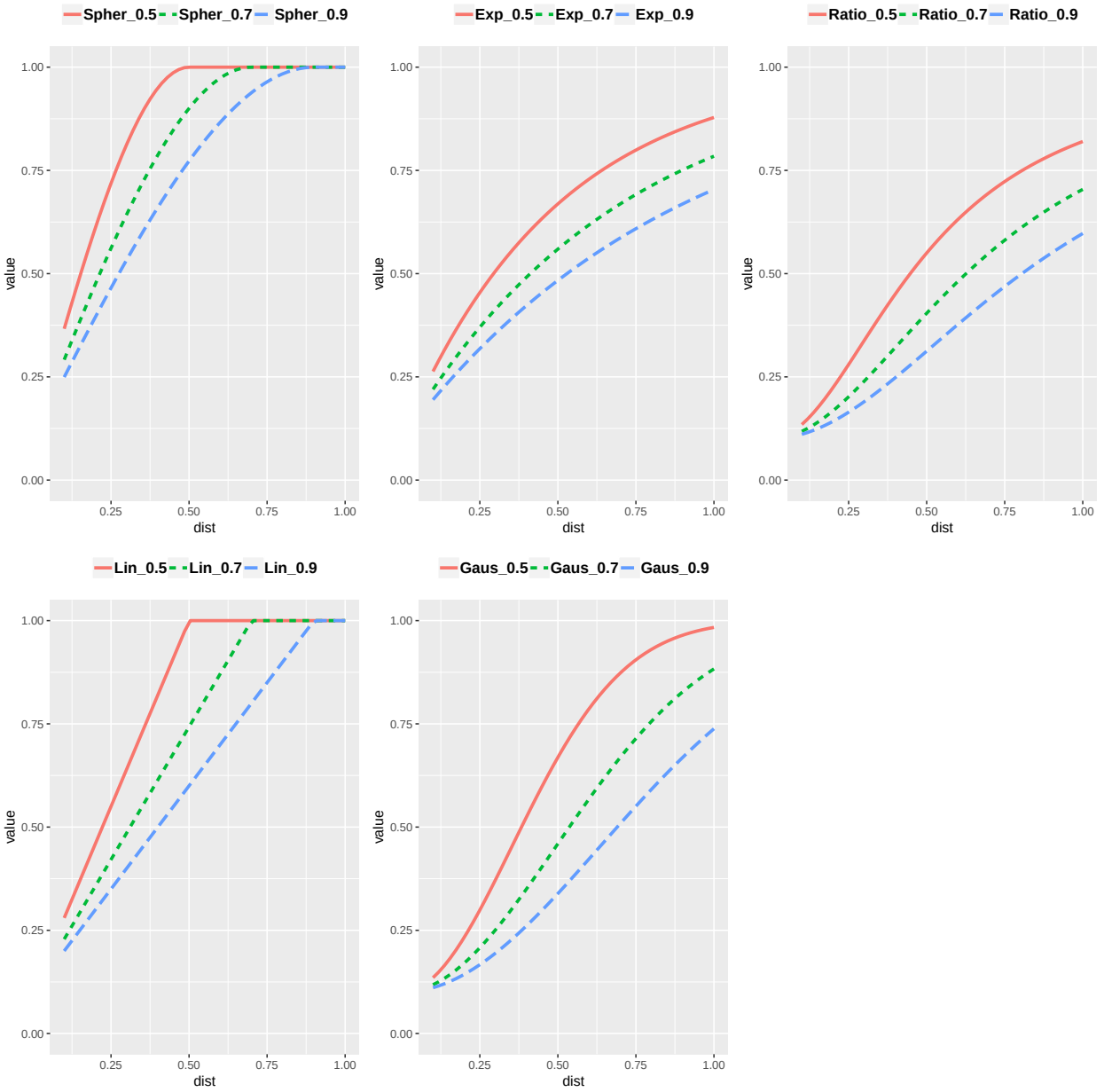


Figure 5.16: Different semi-variogram patterns: Spherical, Exponential, Rational quadratic, Linear and Gaussian. Each pattern has a nugget of 0.1. The value of the range is 0.5, 0.7 or 0.9.

Below, the models presented in Table 4.1 are implemented in R.
For this example, we do not know which form of semi-variogram is classicaly used to model

dependence of yield in a field, and we do not feel confident to choose a model from the form of the semi-variogram (left part of Figure 5.17). The choice is then made using the AIC criteria.

```
library(nlme)
model2.lm <- gls(YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled
                 + accuf*slope_scaled + accuf*aspect + accuf*hshade_scaled, data=Xutm
modSpher <- gls(YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled
                + accuf*slope_scaled + accuf*aspect + accuf*hshade_scaled,
                data=Xutm, correlation=corSpher(form=~x+y,nugget=T))
modLin <- gls(YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled
              + accuf*slope_scaled + accuf*aspect + accuf*hshade_scaled,
              data=Xutm, correlation=corLin(form=~x+y,nugget=T))
modRatio <- gls(YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled
                + accuf*slope_scaled + accuf*aspect + accuf*hshade_scaled,
                data=Xutm, correlation=corRatio(form=~x+y,nugget=T))
modGaus <- gls(YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled
               + accuf*slope_scaled + accuf*aspect + accuf*hshade_scaled,
               data=Xutm, correlation=corGaus(form=~x+y,nugget=T))
modExp <- gls(YIELD_scaled ~ accuf + N + slope_scaled + aspect + hshade_scaled
              + accuf*slope_scaled + accuf*aspect + accuf*hshade_scaled,
              data=Xutm, correlation=corExp(form=~x+y,nugget=T))
```

```
AIC(modSpher,modLin,modRatio,modGaus,modExp)

          df        AIC
modSpher 12 -837.72045
modLin   12  -14.27035
modRatio 12 -785.46606
modGaus  12 -752.27858
modExp   12 -836.82643
```

We need to check that the chosen model `modSpher` solves the problem of spatial dependency, and that the assumptions are verified on the residuals of `modSpher`. Figure 5.17 shows the semi-variogram of the raw residuals of `modSpher` and the semi-variogram of the studentized

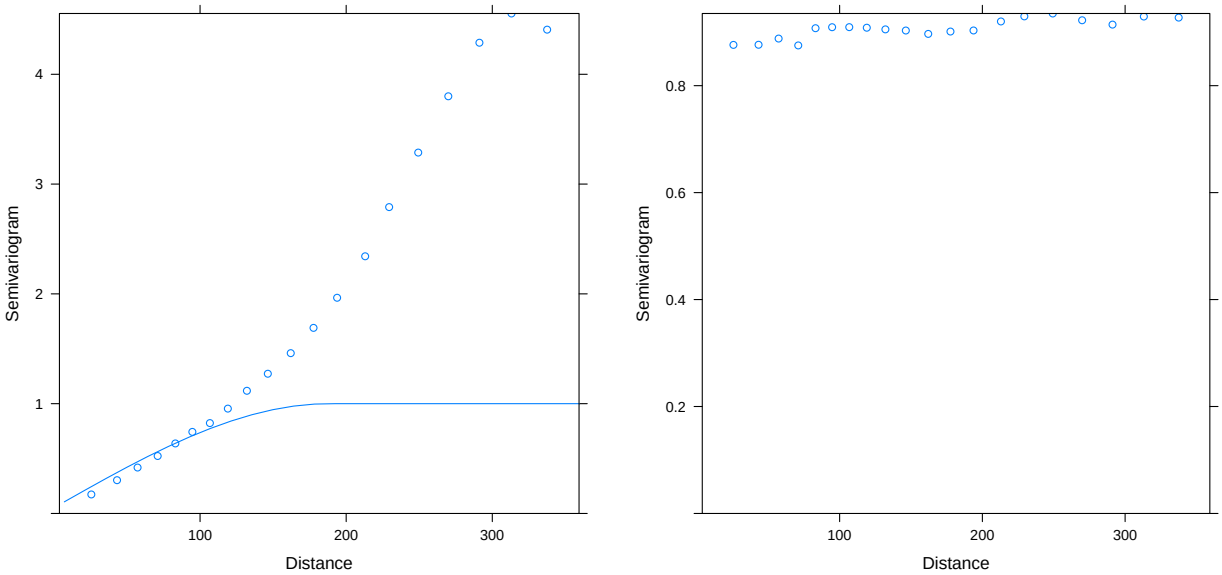


Figure 5.17: (left) Semi-variogram for the raw residuals of `modSpher`. (right) Semi-variogram for the studentized residuals of `modSpher`.

residuals of `modSpher`. The Figure 5.17 validates our choice for `modSpher`. Indeed, it is expected that the raw residuals show a spatial dependency, but we have taken into account this dependency and we have modelled it. The chosen modelisation seems correct as the Studentized residuals did not show any spatial dependence.

```
VarioSpher_raw <- Variogram(modSpher, form =~ LONGITUDE + LATITUDE,
                        robust = TRUE, maxDist = 350, resType = "pearson")
plot(VarioSpher_raw,smooth=FALSE)
```

```
VarioSpher_normalized <- Variogram(modSpher, form =~ LONGITUDE + LATITUDE,
                          robust = TRUE, maxDist = 350, resType = "normalized")
plot(VarioSpher_normalized,smooth=FALSE)
```

Then, we represent the Studentized residuals of `modSpher` on a map, see Figure 5.18. Comparing this map with Figure 5.6, it seems that a good part of the spatial autocorrelation among the residuals has been taken into account.

```
Xutm@data$resSpherNorm <- as.numeric(resid(modSpher,type="normalized"))
spplot(Xutm, "resSpherNorm", col.regions=brewer.pal(9,"Oranges"),
            cex=.2*(1:5), aspect=1/2, main="Normalized residuals of modSpher")
```

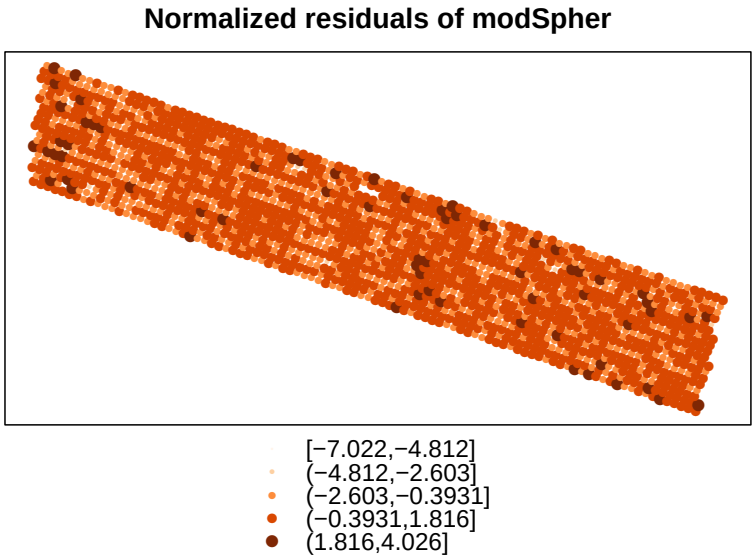


Figure 5.18: Bubble map for residuals of `modSpher`.

Looking at a Moran's test on the residuals, it seems that the spatial autocorrelation among the residuals has been taken into account.

```
moran.mc(Xutm@data$resSpherNorm,W,nsim=1000,alternative="greater")



Monte-Carlo simulation of Moran I

data:  Xutm@data$resSpherNorm
weights: W
number of simulations + 1: 1001

statistic = -0.023487, observed rank = 26, p-value = 0.974
alternative hypothesis: greater
```

We also check the normality of the residuals of `modSpher`.

```
par(mfrow=c(1,2))
hist(Xutm@data$resSpherNorm,main="Studentized residuals of modSpher")
qqnorm(Xutm@data$resSpherNorm)
qqline(Xutm@data$resSpherNorm)
```

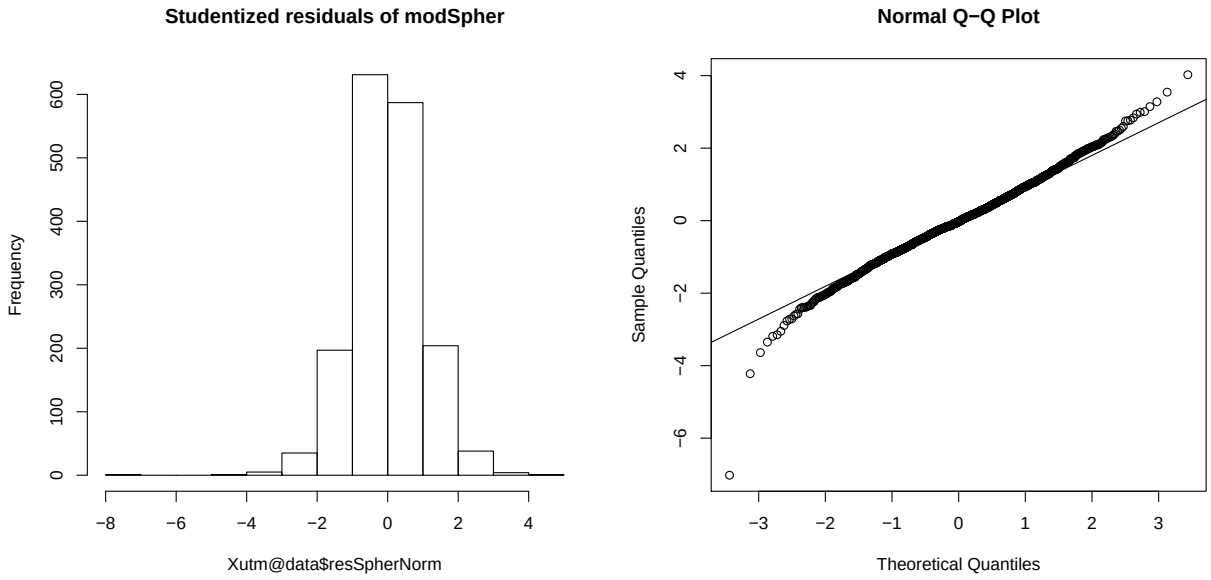


Figure 5.19: Histogram of residuals of `mod.Spher` and associated QQ plot.

Predictions are possible from extended linear model using the function `predict` or `predict.gls` from the package `nlme`. We can make predictions for the data on which the model was fitted, or for new data. Concerning new data, we can still try to make previsions for the yield if the nitrogen content is increased uniformely by one unit.

```
pred_extended <- predict(modSpher)
head(pred_extended)
```

```
         1          2          3          4          5          6
0.26553978 0.20447253 0.14433762 0.09495807 0.06079698 0.05992729
```

```
Xutm2 <- Xutm
Xutm2@data$N <- Xutm@data$N + 1
newpred_extended <- predict(modSpher, newdata=Xutm2@data)
head(newpred_extended)

[1] 0.26756172 0.20649447 0.14635956 0.09698002 0.06281892 0.06194923
```

REMARK 5.6.1 *Using the package* **nlme**, *modelling both a heteroscedasticity and a spatial correlation is possible, by using both arguments* **weights** *and* **correlation** *in a same model.*

REMARK 5.6.2 *In practice, it is easier to use a regression model designed for spatially autocorrelated data, as you do not have to specify a form for the variance-covariance matrix. However, if one of these two models does not give a satisfactory result, you can try an extended linear model. You will then have to choose the form of the variance-covariance matrix yourself, using the form of the semi-variogram, or criteria like AIC. Morevover, the regression models for spatially autocorrelated data are often more intuitive than extended linear models.*

Chapter 6

Spatial Estimation and Interpolation

The list of specific R packages used in this part to carry out spatial estimation and interpolation are:

a graph : `ggplot2`, `gridExtra`

b Spatial data management : `sp`, `spdep`, `raster`

c Spatial data analysis : `gstat` `nlme`

d Spatial data map representation : `mapview` `lattice`

The package name appears before the function (package::function()) for pedagogical reasons and in order to clarify the origin of the unusual `R` functions used in the coding.

6.1 Interpolation Map with IDW (Inverse Distance Weight)

The variable of interest $Z(s)$ was measured on six different sites $\{s_A, s_B, s_C, s_D, s_E, s_F\}$.

Now we need to interpolate its value at a new site on figure 6.1 ?

6.1.1 Principle of the IDW interpolation

The IDW or Inverse Distance Weighted interpolation gives an estimation of $Z(s_{new})$ built as a weighted linear combination of the neighborhood values. The weight is:

- high for sites nearby s_{new}

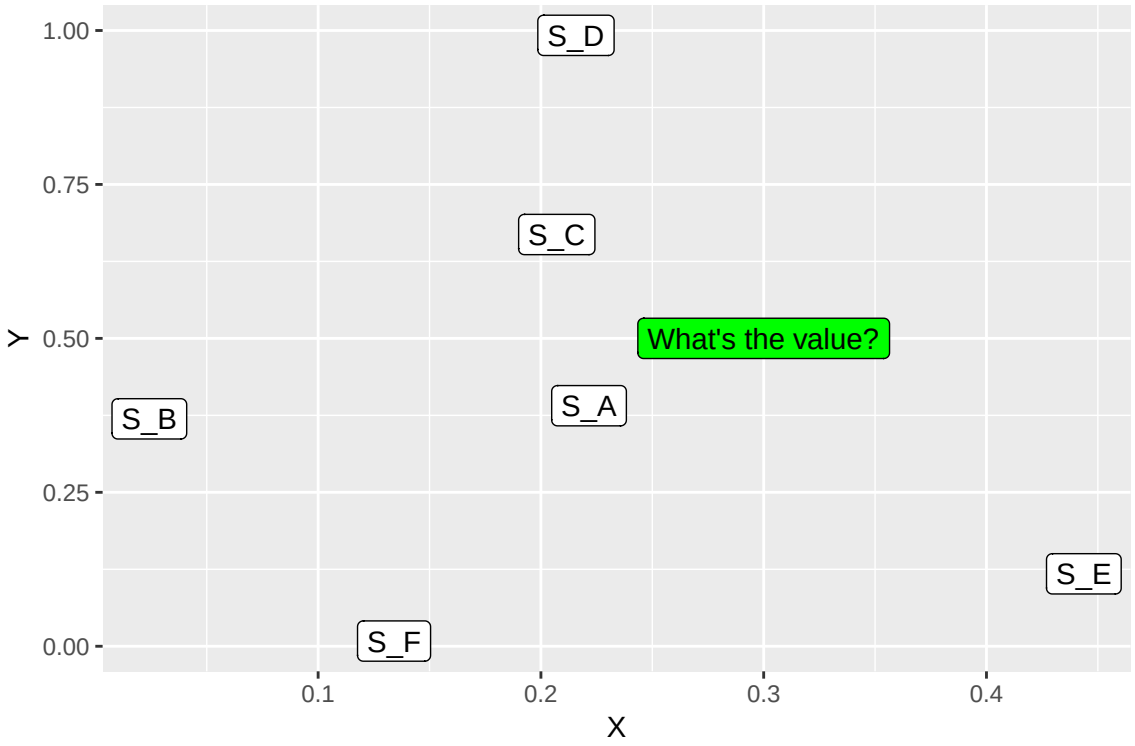


Figure 6.1: Illustration

- low for neighbours far away from s_{new}

6.1.2 Definition of "Neighborhood"

Several definitions exist (for more details see section 4.4.2), among them:

- $Neighborhood(s_{new})$ = k nearest neighbours/sites (positioning)
- $Neighborhood(s_{new})$ = neighbours/sites inside a circle centered on s_{new} and with a given radius R.

N_{new} will denote the number of sites in the $Neighborhood(S_{new})$ in the following.

6.1.3 Equation of the IDW

Each site s_i of the neighborhood has a weight inversely proportional to the distance $dist(s_i\, s_{new})$ between (s_i) and the site to be estimated (s_{new}):

$$\hat{Z}(s_{new}) = \frac{\sum_{i=1}^{N_{new}} \frac{Z(s_i)}{dist(s_i,s_{new})}}{\sum_{i=1}^{N_{new}} \frac{1}{dist(s_i,s_{new})}}$$

A broader definition uses a power function of the distance between s_i and s_{new}:

$$\hat{Z}(s_{new}) = \frac{\sum_{i=1}^{N_{new}} \frac{Z(s_i)}{dist(s_i,s_{new})^P}}{\sum_{i=1}^{N_{new}} \frac{1}{dist(s_i,s_{new})^P}}$$

6.1.4 Algorithm

1. Define the neighborhood
2. Create a grid of interpolation
3. Calculate the IDW interpolation at each node of the grid.
4. Plot the interpolation surface on a graph
5. Analyse the sensitivity of the interpolation to the definition/size of the neighborhood

6.1.5 Properties, Limits of the IDW Approach

- The IDW interpolation is an **exact estimation**. It gives the observed values as estimated for the sampled/observed sites.
- The **interpolation surface is continuous**/smooth
- The interpolation does not depend on the site configuration but on the distances between sites.

6.1.6 Example: SIC97

Data description

Dataset from the Spatial Interpolation Comparison exercise 1997 (`SIC97`) *Reference: Journal of Geographic Information and Decision Analysis, vol.2, no.2, pp. 1-11 (1997)*

This dataset is made of 467 daily rainfall measurements made in Switzerland on 8th May 1986 (`sic_full`). From them, 100 observed data (`sic_obs`) were used to estimate the rainfall at the remaining 367 locations. The aim of this SIC was to compare different spatial interpolation tools.

The data is provided within the package `gstat` and figure 6.2 gives a spatial representation of the rainfall sites.

```
# load data from package gstat
library(gstat)
data(sic97)

# transform SpatialPointsDataFrame into Data Frame
mydata <- data.frame(sic_full$rainfall,coordinates(sic_full))
myobs <- data.frame(sic_obs$rainfall,coordinates(sic_obs))

# Use classic ggplot2 commands on the Data Frame
ggplot(data=mydata,aes(x=X,y=Y))+
  geom_point(shape=21,colour = "black", fill = "white",size=2) +
  geom_point(data=myobs,shape=21,colour = "black",
             fill = "green",size=2)
```

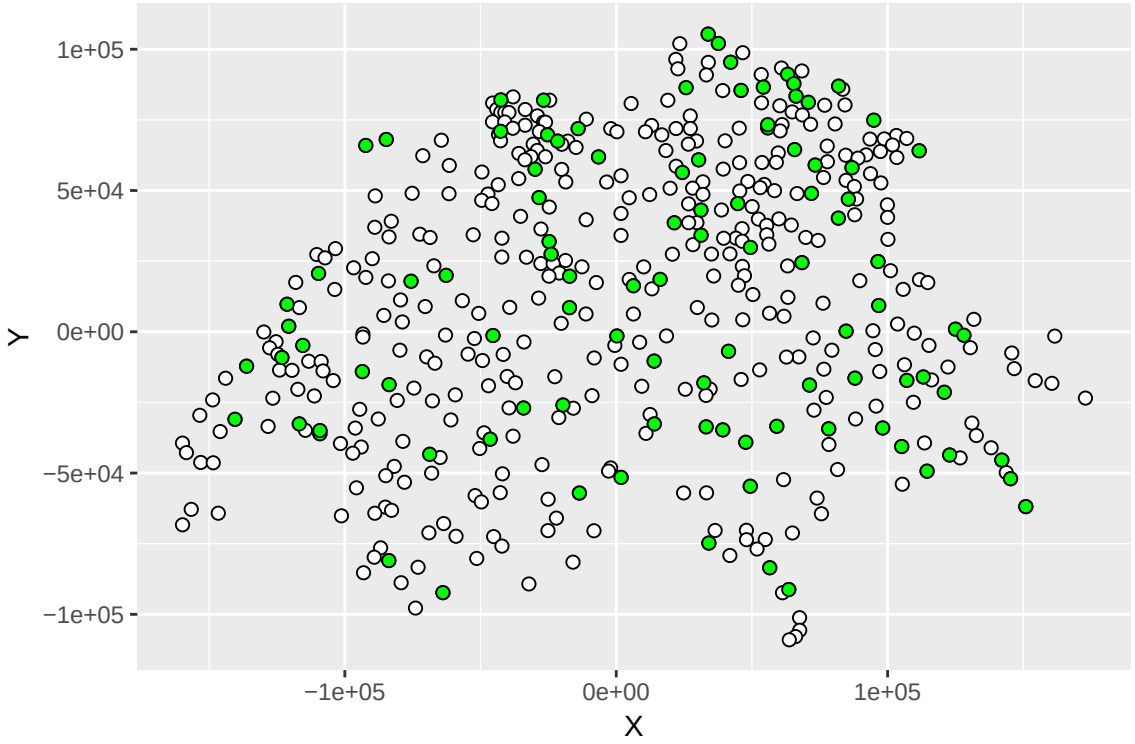


Figure 6.2: Representation of the `SIC97` full rainfall dataset with the restricted observed sites filled in green

```
# sp package allow to plot directly from SpatialPointsDataFrame:
#plot(sic_obs)
```

We will use the dataset `sic_obs` to illustrate how to create an interpolation map using *R*.

Grid

First, we need a grid. Some R functions implemented in the package `sp` can help us to create a rectangular grid as follows:

```
Xcell <- 100
Ycell <- 100
Xwidth <- (max(mydata$X)-min(mydata$X))/Xcell
Ywidth <- (max(mydata$Y)-min(mydata$Y))/Ycell

# From package sp
mydata.grid <- sp::GridTopology(c(min(mydata$X),min(mydata$Y)),
                            c(Xwidth,Ywidth), c(Xcell,Ycell))

mydata.grid <- sp::SpatialGrid(mydata.grid)
```

Map

Now we estimate the rainfall at each node of the grid with a neighborhood defined by a circle with a radius of `maxdist` = 100000

```
# IDW interpolation
obs_idw <- gstat::idw(rainfall~1,sic_obs,mydata.grid, maxdist=100000)

[inverse distance weighted interpolation]

summary(obs_idw)

Object of class SpatialGridDataFrame
Coordinates:
           min      max
```

```
[1,] -161475.5 171227.5
[2,] -110079.8 104289.2
Is projected: NA
proj4string : [NA]
Grid attributes:
  cellcentre.offset cellsize cells.dim
1           -159812  3327.03       100
2           -109008  2143.69       100
Data attributes:
   var1.pred         var1.var
 Min.   : 17.27   Min.   : NA
 1st Qu.:134.12   1st Qu.: NA
 Median :169.56   Median : NA
 Mean   :181.14   Mean   :NaN
 3rd Qu.:225.47   3rd Qu.: NA
 Max.   :583.40   Max.   : NA
                  NA's   :10000

full_idw <- gstat::idw(rainfall~1,sic_full,mydata.grid, maxdist=100000)

[inverse distance weighted interpolation]
```

IDW interpolator is an exact interpolator which gives the observed value at each observed site. The map looks pixelated (not smooth) because IDW interpolator does not take into account the correlation structure of the rainfall between sites. Increasing the number of observations like in figure 6.3 improves the estimation but highlights the pixelated appearance of the map.

```
g1 <- sp::spplot(obs_idw["var1.pred"],main="Obs")
g2 <- sp::spplot(full_idw["var1.pred"],main="Full")
gridExtra::grid.arrange(g1,g2,nrow=2)
```

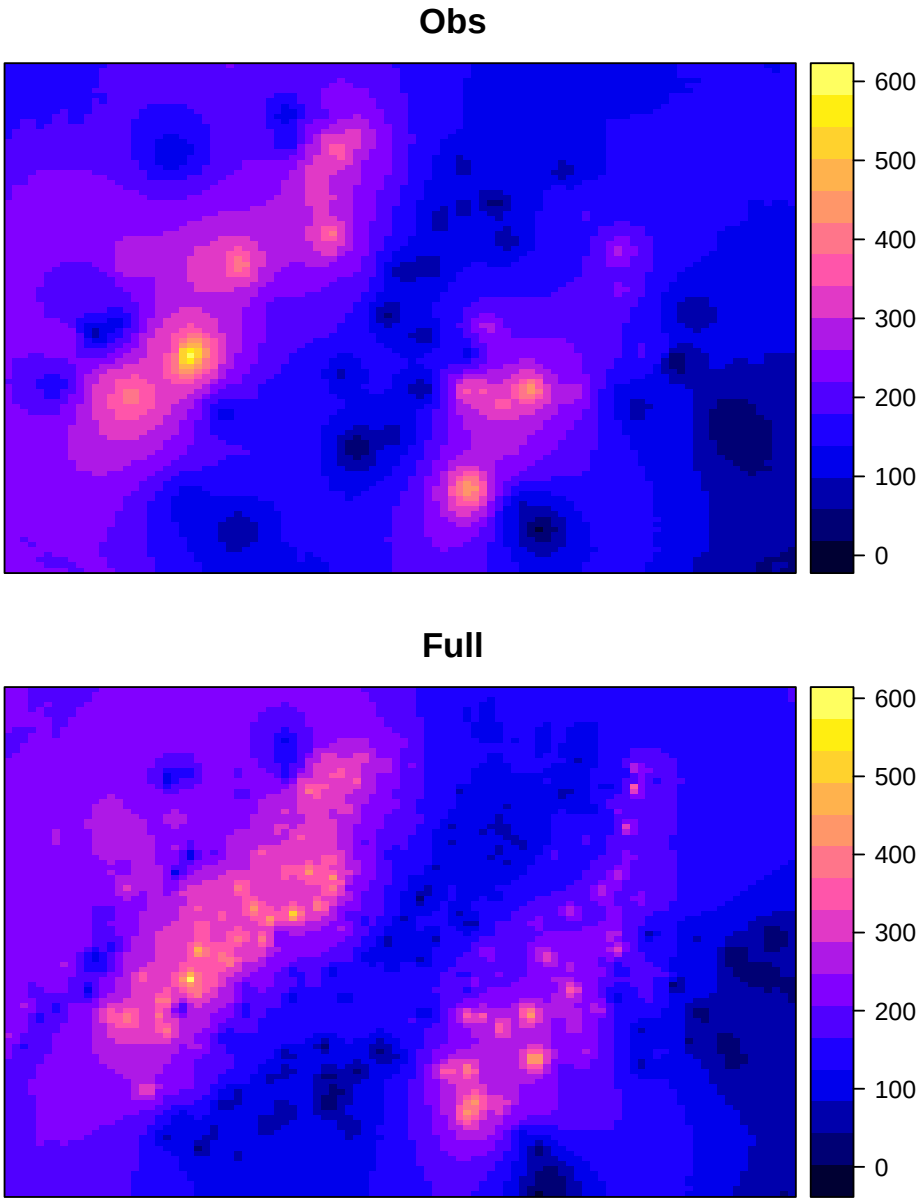


Figure 6.3: IDW interpolation map

6.2 Kriging

Now, let us consider an interpolator which takes into account spatial correlation between two sites. Kriging was first developped and used for natural resource evaluation (see books like Webster [9], Goaverts [**?**] and in French Arnaud and Emery [10]).

6.2.1 The Principle of Kriging

- To characterize the spatial structure of the random process studied by a variogram.
- To construct a linear combination that best predicts/estimates the value at a given point by taking into account the correlations between points in the neighborhood
- To quantify uncertainty related to prediction (variance of the prediction/estimation)

6.2.2 The andom Process

A random process, RP Z_s,is defined as a set of usually dependent random variables $Z(s)$, one for each site s in the study area Γ.

$$RP\ Z_s = \{Z(s), \forall s \in \Gamma\}$$

To any set of N sites a vector of N random variates corresponds; with a probability (or multivariate cdf) of:

$$F(Z(s_{(1)}), \ldots, Z(s_{(N)})) = Prob(Z(s_{(1)}) \leq z_1, \ldots, Z(s_{(N)}) \leq z_N)$$

The set of all such N-multivariate cdf for any positive integer N constitutes the spatial law of RP Z_s.

In practice, the analysis in this part will be limited to no more than two sites at a time and will mainly require the notion of variogram (section 4.7):

- Covariance: $C(s_i, s_j) = E[Z(s_i)Z(s_j)] - E[Z(s_i)]E[Z(s_j)]$
- Variogram: $2\gamma(s_i, s_j) = Var[Z(s_i) - Z(s_j)]$

6.2.3 Characterizing the Spatial Structure

This section is a brief reminder of section 4.7.

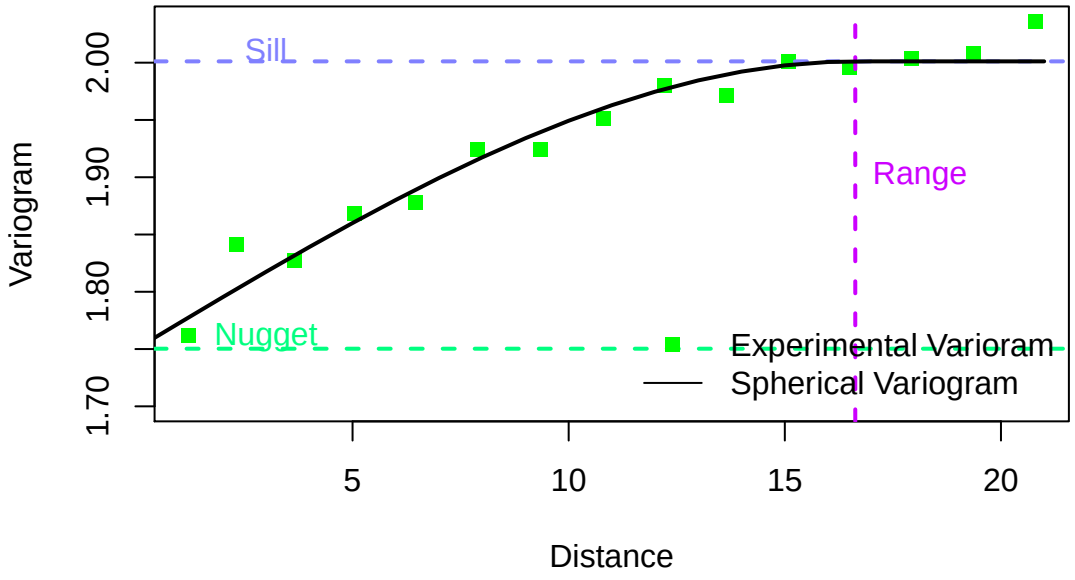


Figure 6.4: Variogram model parameters

The variogram is a method which measures the average 'pattern dissimilarity' between two samples according to their distance. The experimental variogram is computed from the data and adjusted with a mathematical model defined by three parameters (see figure 6.4).

- The nugget: the variance between two points at distances smaller than the shortest sampling interval. This variance is due to a measurement error or to spatial discontinuity.
- The range: distance above which sites are non correlated.
- The sill: The variance between 2 non correlated sites.

The variogram shape is also determined by i) the slope or speed at which destructuring occurs according to distance, and ii) the direction in which the variogram was computed/constructed (depends on the spatial anisotropy).

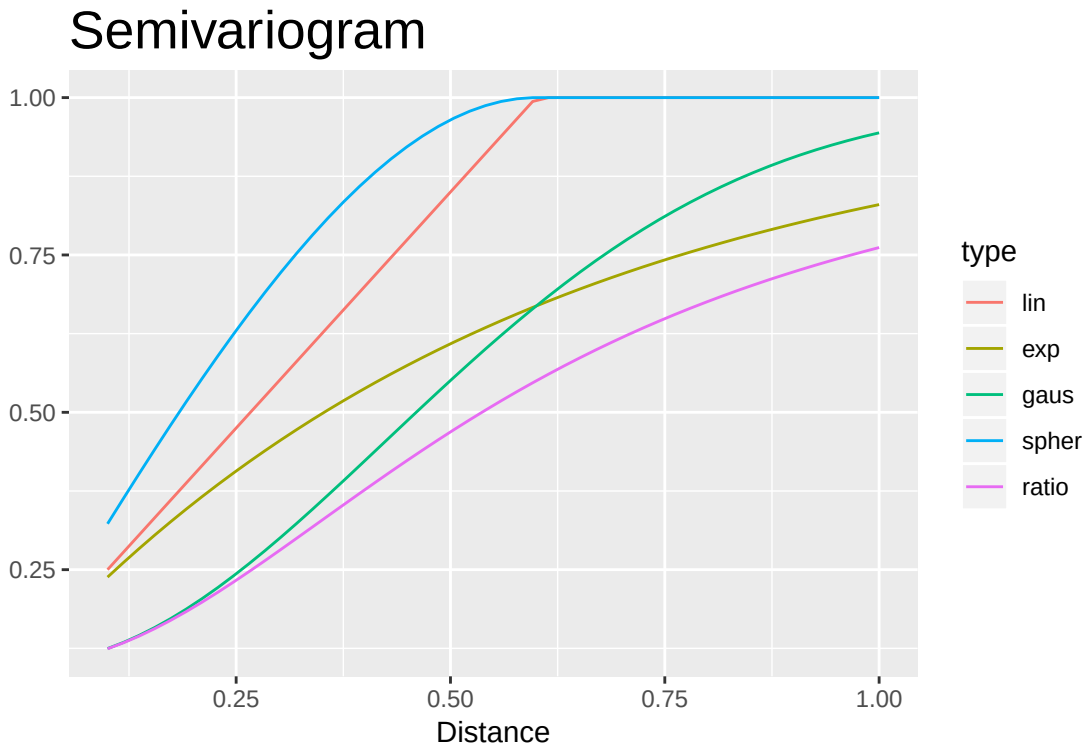


Figure 6.5: Shape of the different mathematical models used for semivariograms

6.2.4 The kriging estimator

In the context of kriging, we want to create an estimator defined as a linear combination $\sum_{i=1}^{N_{new}} \lambda_i Z(s_i)$ (where s_i is an observed site from the $Neighborhood(S_{new})$) that best estimates the value at the new site.

This estimator is a random variable and by definition its variance must be non negative. We need the variance of any finite linear combination of random variables $Z(s_i)$, $s_i \in \Gamma$, to be non negative. **To ensure this positive definite condition, The RP Z_s is assumed to be stationary**.

The mathematical model of a variogram must fulfil the positive definite condition. In fact, few permissible mathematical models exist. Among them, the most usual are the exponential and the Gaussian models for smooth increases of the variance before the range (see figure 6.5). Conversely, the linear and spherical models are preferred for rapid incrementing of the variance before the range.

To go further, the mathematical definition of the positive definite condition is: the variance of any finite linear combination of random variables $Z(s_i)$, $s_i \in \Gamma$, must be non negative. This variance can be expressed as a linear combination of the covariance values:

$$Var[\sum_{i=1}^{n} \lambda_i Z(s_i)] = \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j C(s_i, s_j) \geq 0$$

This variance can be rewritten in terms of a semivariogram model, as $\gamma(h) = C(0) - C(h)$:

$$Var[\sum_{i=1}^{n} \lambda_i Z(s_i)] = \sum_{i=1}^{n} \lambda_i \sum_{j=1}^{n} \lambda_j - \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j \gamma(s_i, s_j) \geq 0$$

Therefore, the variance of the kriging estimator is known and defined by the variogram (model) and the weights of the linear combination (λ_i). Furthermore, the value of the weights are conditioned by the variogram because of the positive definite condition.

6.2.5 Stationarity Assumptions and kriging models

The RP Z_s is assumed stationary to use variogram or kriging statistical tools.

A sufficient condition for the existence of the variogram is the intrinsic stationarity. Often, the RP is assumed to be stationary of order 2 (which implies the intrinsic stationarity).

- Second order stationary:
  - Expected value exists and is constant ($E[Z(s_i)] = m, \forall s_i \in \Gamma$).
  - The covariance function $C(h)$ exists and depends only on the distance h.
- Intrinsic stationarity: Increments $Z(s_i) - Z(s_j)$ are second order stationary.

As introduced in section 4, the RP Z_s is usually decomposed into a residual component (random function R_s) and a trend component (deterministic function $T(s)$). Sometimes the residual component is again divided into two parts: a spatially autocorrelated process ($\eta(s_i)$) and a nugget (an uncorrelated random process ($\varepsilon(s_i)$)). Conclusion :

$$Z(s_i) = R(s_i) + T(s_i)$$

where the RP R_s is assumed second order stationary with a mean value of zero.

Three kriging variants can be distinguished according to the function $T(s)$ used for the trend.

1. Simple Kriging (constant known mean value): $T(s_i) = m$, $\forall s_i \in \Gamma$
2. Ordinary Kriging (locally constant unknown mean):

$$\forall s_j \in Neighborhood(s_{new}),\ T(s_j) = m(s_{new})$$

3. Universal kriging (non constant unknown mean)

6.2.6 Ordinary Kriging

Estimator (definition and properties)

The value at a new site $\hat{Z}(s_{new})$ is estimated from the samples in its neighborhood $Z(s_1)$, $Z(s_2)$, ..., $Z(s_{N_{new}})$ by a linear combination:

$$\hat{Z}(s_{new}) = \lambda_1(s_{new})\, Z(s_1) + \lambda_2(s_{new})\, Z(s_2) + \ldots + \lambda_{N_{new}}(s_{new})\, Z(s_{N_{new}})$$

where $\lambda_i(s_{new})$, $1 \leq i \leq N_{new}$, are determined by solving the system of equations corresponding to these two assumptions:

1. The expected value of the estimator is not biased.

2. The variance of the estimator is minimal.

The first assumption implies the following constraint: $\sum_{i=1}^{N_{new}} \lambda_i(s_{new}) = 1$

Each predictor $\hat{Z}(s_{new})$ is a random variable with a variance, often called the kriging variance.This variance depends only on the model variogram and on the spatial pattern of the sites (those observed and those to be predicted). Therefore this variance does not depend on the observed values $Z(s_1), Z(s_2), \ldots, Z(s_{N_{new}})$. The kriging variance is optimal for a Gaussian RP Z_s.

To go further using **the ordinary Kriging system:**

1. Remember that $\forall s_j \in Neighborhood(s_{new})$, $T(s_j) = m(s_{new})$, so:

$$\begin{aligned}\hat{Z}(s_{new}) &= \sum_{i=1}^{N_{new}} \lambda_i(s_{new}) \left(Z(s_i) - m(s_{new})\right) + m(s_{new}) \\ &= \sum_{i=1}^{N_{new}} \lambda_i(s_{new}) \, Z(s_i) + [1 - \sum_{i=1}^{N_{new}} \lambda_i(s_{new})] m(s_{new})\end{aligned}$$

   The unknown local mean value is filtered from the linear estimator when the kriging weights sum to 1.

2. The variance of the estimator is:

$$\begin{aligned}var[\hat{Z}(s_{new}) - Z(s_{new})] &= \sum_{i=1}^{N_{new}} \sum_{j=1}^{N_{new}} \lambda_i(s_{new}) \, \lambda_j(s_{new}) \, C(s_i, s_j) + C(0) \\ &- 2 \sum_{i=1}^{N_{new}} \lambda_i(s_{new}) \, C(s_i, s_{new})\end{aligned}$$

   The minimization of this error variance under the non bias condition leads to the following system of linear equations:

$$\begin{aligned}&\sum_{i=1}^{N_{new}} \lambda_i(s_{new}) \, \gamma(s_i, s_j) - m(s_{new}) = \gamma(s_i, s_{new}), \; i = 1, \ldots, N_{new} \\ &\sum_{i=1}^{N_{new}} \lambda_i(s_{new}) = 1\end{aligned}$$

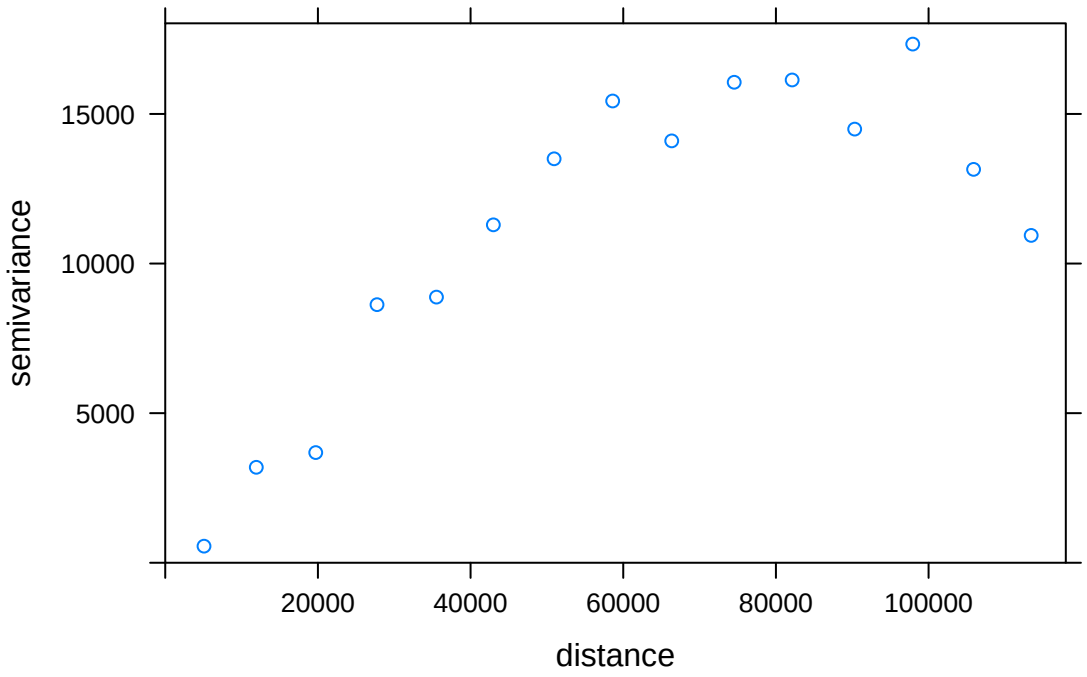


Figure 6.6: Semivariogram obtained with sic obs dataset

6.2.7 Example: Ordinary kriging map with SIC97 dataset

6.2.7.1 Experimental Variogram

Remember that each point of the experimental semivariogram represents half the variance between two observed sites separated by a specified distance. As the distance increases the variability between sites is assumed to increase.

Package *gstat* do the automatic computation of the experimental variogram.

```
# package gstat
v <- gstat::variogram(rainfall ~ 1, data= sic_obs)
plot(v)
```

We can see on figure 6.6 that for adjacent sites, the nugget effect which is half the variance (or semivariance) seems null. For a distance above 80000 the semivariance looks stabilized and has attained a sill of almost 15000.

Comments on the results *v*. The experimental variogram of the rainfall for `sic_obs` dataset is computed with the following formula

$$2\hat{\gamma}(h) = \frac{1}{\mid N(h) \mid} \sum_{i,j \in Neigh(h)} (Z_{s_i} - Z_{s_j})^2$$

where $N(h) = \{(i,j), dist(s_i, s_j) = h\}$ is the set of pairs of points separated by a distance h and $\mid N(h) \mid$ is the number of elements in $N(h)$. When `v` is printed, only values at some specific distances are given. Practical computation consists in cutting the interval of observed distances ($[h_{min}, h_{max}]$) into bins of the same length and then,

1. For all pairs (s_i, s_j) compute $V_{i,j} = (Z(s_i) - Z(s_j))^2$
2. Plot $V_{i,j}$ according to the distance $dist(s_i, s_j)$
3. Compute the mean value of all the points in each bin to get the experimental variogram point

```
head(v)
```

```
   np      dist     gamma dir.hor dir.ver   id
1  15  5078.697   554.700       0       0 var1
2  68 11926.084  3190.882       0       0 var1
3 111 19714.898  3683.126       0       0 var1
4 132 27743.181  8626.913       0       0 var1
5 142 35528.553  8879.391       0       0 var1
6 191 42984.622 11295.016       0       0 var1
```

The experimental variogram is sensitive to the choice of breaks and to h_{max}. This option can be set in function `variogram` with option `boundaries` or `width`. Often h_{max} is not the maximum distance but half its value to prevent border effects in the experimental variogram. For function `variogram` it is by default, *the length of the diagonal of the box spanning the data divided by three.*

Eventually, the experimental variogram can be computed along one or several direction(s), when the processus Z_s is spatially anisotropic (option `alpha` for function `variogram`).

Calibration of the Model Variogram

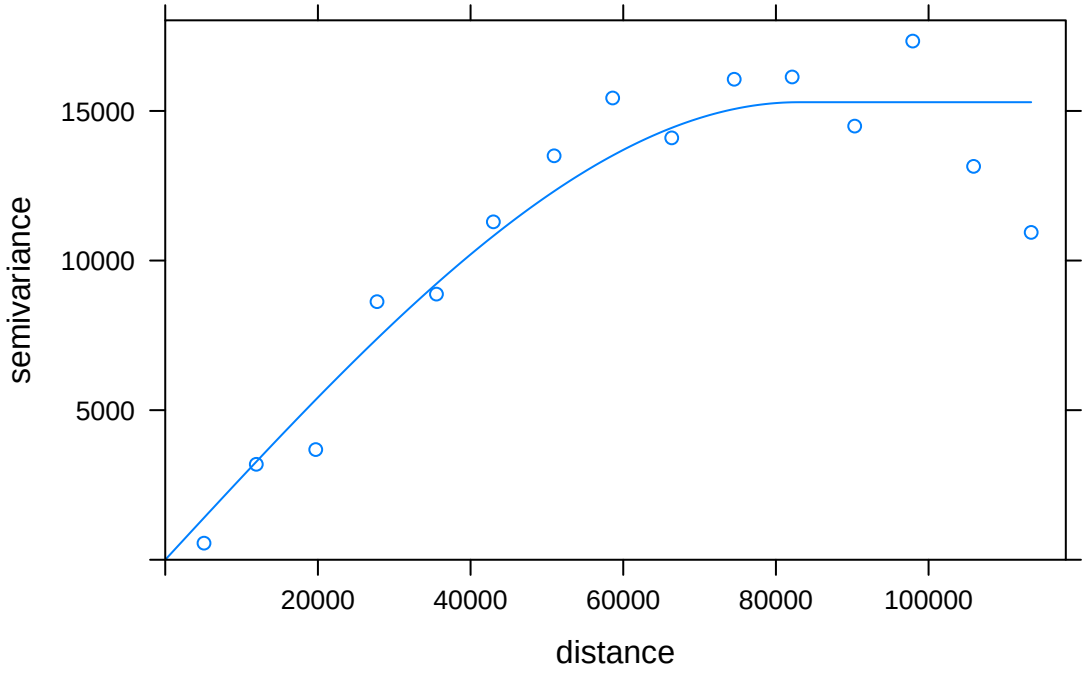


Figure 6.7: Experimental Variogram Fitted with a Spherical Model

```
# automatic adjustment of a spherical model
# to the experimental variogram v
m.fit <- gstat::fit.variogram(v, vgm("Sph"))
m.fit
plot(v,m.fit)
```

```
  model    psill    range
1   Nug     0.00     0.00
2   Sph 15292.38 82946.36
```

To choose the best model between the list of possible models, a visual inspection is often enough but some statistical criteria like AIC or the weighted Sum of Squares (WSS) are also used. Some `R` functions like `fit.variogram` do an automatic WSS fit (see figure 6.7).

to go further, *WSS mathematical definition*:

$$WSS = \sum_{k=1}^{K} w(h_k) \left[\hat{\gamma}(h_k) - \gamma(h_k)\right]^2$$

where $2\hat{\gamma}(h_k)$ and $2\gamma(h_k)$ are respectively the experimental and the model variogram values for sites separated by a lag/distance h_k. The weight, $w(h_k)$, is usually proportional to the number of site pairs at lag h_k. For example, for function `fit.variogram` it is equal to the number of point pairs divided by the distance h_k.

6.2.7.2 Ordinary Kriging Map

```
NF.kriged = gstat::krige(rainfall ~ 1, sic_obs,
                  mydata.grid, model = m.fit,nmax=20)
g1 <- sp::spplot(NF.kriged["var1.pred"], main="Prediction")
g2 <- sp::spplot(NF.kriged["var1.var"],main="Variance")
gridExtra::grid.arrange(g1,g2,nrow=2)
```

```
[using ordinary kriging]
```

Kriging gives two results, a prediction and the variance of the prediction. Therefore, we can draw two distinct maps (see figure 6.8). Remember that kriging is an exact interpolator, therefore the variance of the prediction is null on observed sites.

6.2.7.3 Cross Validation

If we want to check the kriging goodness of fit we must calculate residuals (the difference between the observed and the estimated values at a site). In order to be unbiaised, we recommend using residuals obtained by cross validation to respect the following rule: an observation at a site must not participate in the construction of the model which will predict/estimate the value at this site.

The function krige.cv is implemented in `R` to perform cross validation for simple, ordinary or universal point (co)kriging, kriging in a local neighbourhood.

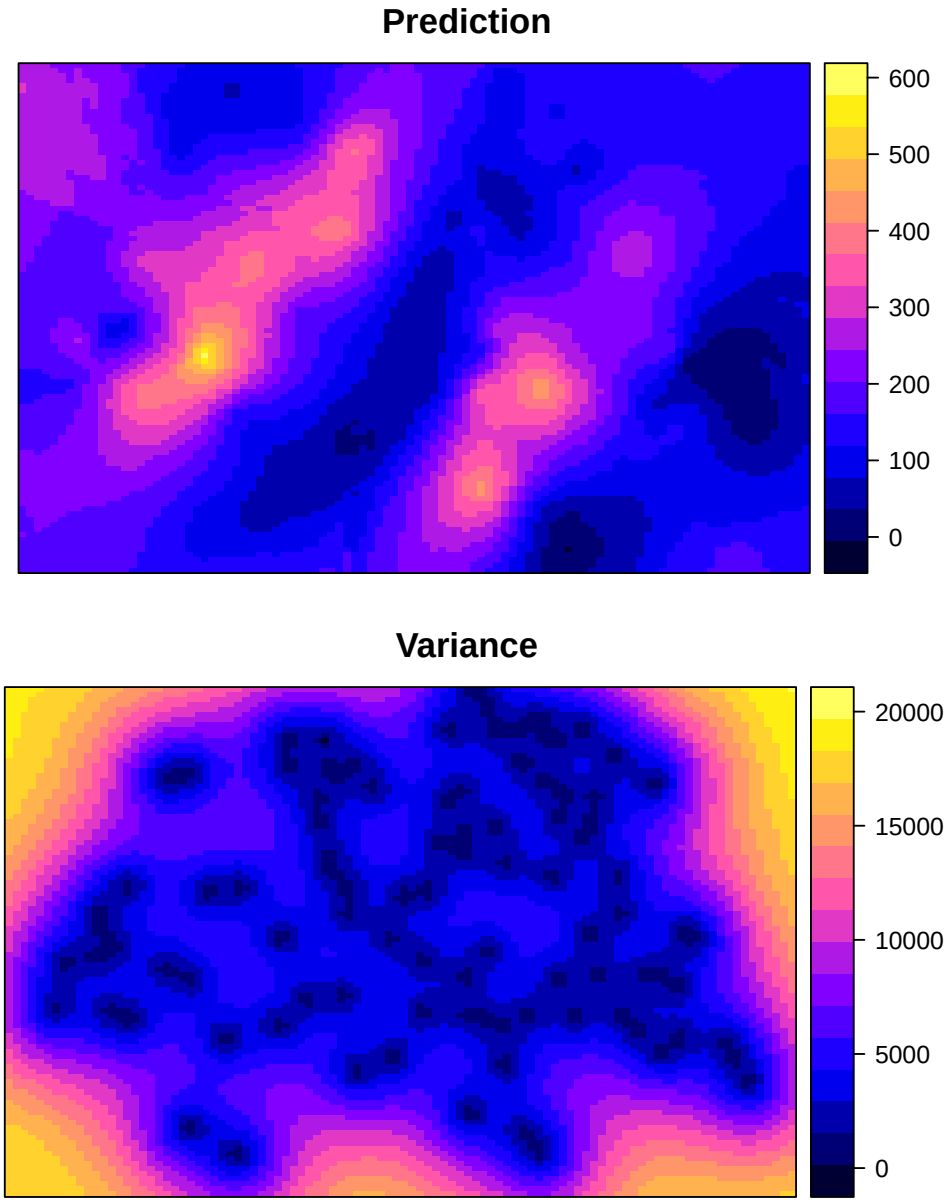


Figure 6.8: Ordinary Kriging Map Interpolation

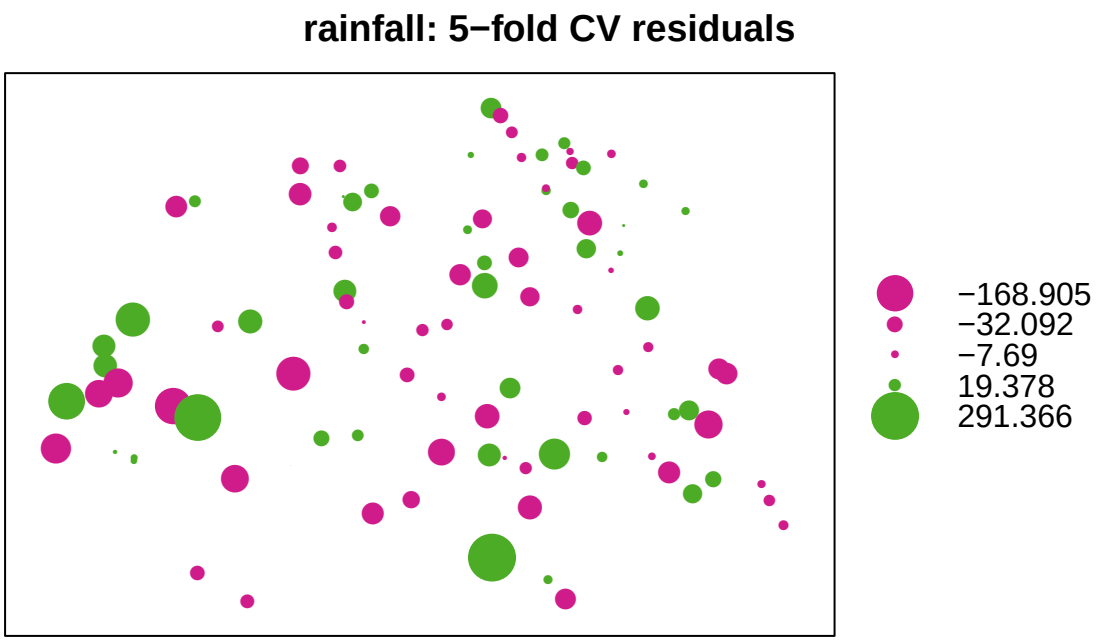


Figure 6.9: Map of the cross validation residuals of the ordinary kriging model fit on sic obs dataset

There are two options for the cross validation (leave one out or k-fold). On figure 6.9 we can see the results for 5-fold. There is still a spatial structure in the residuals (on the left and at the bottom with clusters of high absolute values for residuals). Figure 6.9 gives more information (bias in the prediction) than the variance map 6.8 (expected variability of the prediction due to the spatial structure).

```
NF.kriged.cv = gstat::krige.cv(rainfall ~ 1, sic_obs,
                          model = m.fit, nmax=20, nfold=5)
sp::bubble(NF.kriged.cv, "residual",
       main = "rainfall: 5-fold CV residuals")
```

```
  |
  |                                                                      |   0%
  |
```

```
  |================                                                      |  25%
  |
  |================================                                      |  50%
  |
  |================================================                      |  75%
  |
  |======================================================================| 100%
```

from R documentation: Leave-one-out cross validation (LOOCV) visits a data point, and predicts the value at that location by leaving out the observed value, and proceeds with the next data point. (The observed value is left out because kriging would otherwise predict the value itself.) N-fold cross validation makes a partitions the data set in N parts. For all observation in a part, predictions are made based on the remaining N-1 parts; this is repeated for each of the N parts. N-fold cross validation may be faster than LOOCV.

6.2.7.4 Universal Kriging or Kriging with a Trend

Universal Kriging assumes a linear or quadratic trend (where spatial coordinates could be used as explanatory variables).

$$Z(s_i) = T(s_i) + R(s_i)$$

with

1. Linear trend $T(s_i) = \beta_0 + \beta_1 x_i + \beta_2 y_i$
2. Quadratic trend $T(s_i) = \beta_0 + \beta_1 x_i + \beta_2 y_i + \beta_3 x_i^2 + \beta_4 y_i^2 + \beta_5 x_i\, y_i$

In the rainfall example, ordinary kriging gives poor results (because there is still spatial structure in the residuals). So, we can try the universal kriging.

```
NF.kriged.UK.cv = gstat::krige.cv(rainfall ~ X+Y, locations=sic_obs,
                          model = m.fit, nfold=5,nmax=20)
#residuals=sic_nobs$rainfall-NF.kriged.UK$var1.pred
#NF.kriged.UK@data <- data.frame(NF.kriged.UK@data,residuals)
sp::bubble(NF.kriged.UK.cv,'residual',
       main = "rainfall: residuals from Universal Kriging")
```

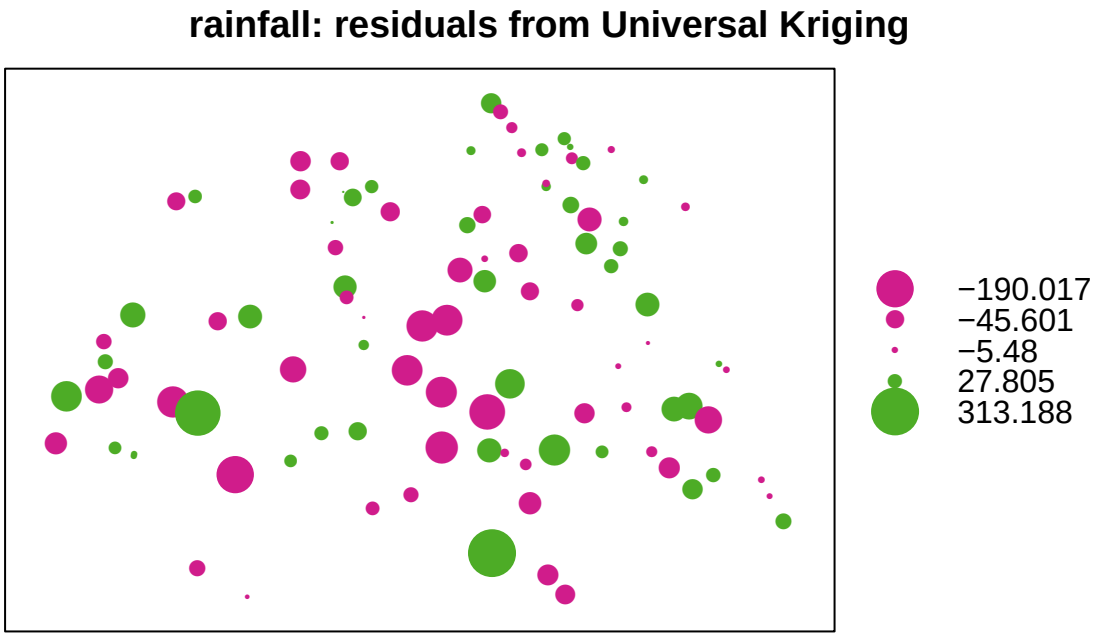


Figure 6.10: Residuals of the universal kriging model fit on sic obs dataset

```
  |
  |                                                                      |   0%
  |
  |================                                                      |  25%
  |
  |================================                                      |  50%
  |
  |================================================                      |  75%
  |
  |======================================================================| 100%
```

Figure 6.10 shows the residuals obtained with universal kriging. This approach brings no improvement. The remaining spatial structure was not due to a spatial unknown linear trend.

Practical recommendation:

Do a Universal kriging or Estimating the trend and computing simple kriging predictions of the residuals is equivalent to Universal Kriging when a linear trend is assumed (Cressie [1], 1993, section 3.4.5).So:

1. Estimate the trend with a regression.
2. Compute the residuals
3. Carry out the variogram estimation and kriging on the residuals but use the **Simple Kriging**!
4. Add the trend to the kriging estimates

6.2.7.5 Comparison with IDW Approach

The neighborhood can be defined in the same way for kriging and IDW. But:

- **IDW**: each site has a weight inversely proportional to the distance to the site to predict (s_{new}).

- **Kriging**: The weighting is built through the variogram model and the spatial pattern of the sites.

For both interpolators, the value on a new site is estimated by a weighted linear combination of the neighboring sites. For kriging, a variance of the prediction can be computed in addition.

6.2.7.6 Using Ordinary Kriging to Predict the Rainfall Data (SIC97)

Kriging can be used to produce interpolation maps but also predictions. We will use the kriging estimator established on the `sic_obs` dataset to predict the rainfall at the remaining 367 locations (`sic_nobs` dataset).

Histograms of observed and predicted values We can compare the results of the kriging prediction to the observation.

```
# nobs dataset
sic_nobs <- sic_full[-(1:100),]
# kriging predictions
NF.kriged.nobs = gstat::krige(rainfall ~ 1, sic_obs,
                      model = m.fit, newdata=sic_nobs,nmax=20)

[using ordinary kriging]

# summary of the observation and the prediction
summary(data.frame(sic_nobs$rainfall,
                   NF.kriged.nobs@data$var1.pred))

 sic_nobs.rainfall NF.kriged.nobs.data.var1.pred
 Min.   :  0.0     Min.   : -1.695
 1st Qu.: 90.0     1st Qu.: 90.027
 Median :171.0     Median :172.192
 Mean   :186.7     Mean   :184.582
 3rd Qu.:270.5     3rd Qu.:264.987
 Max.   :585.0     Max.   :585.000
```

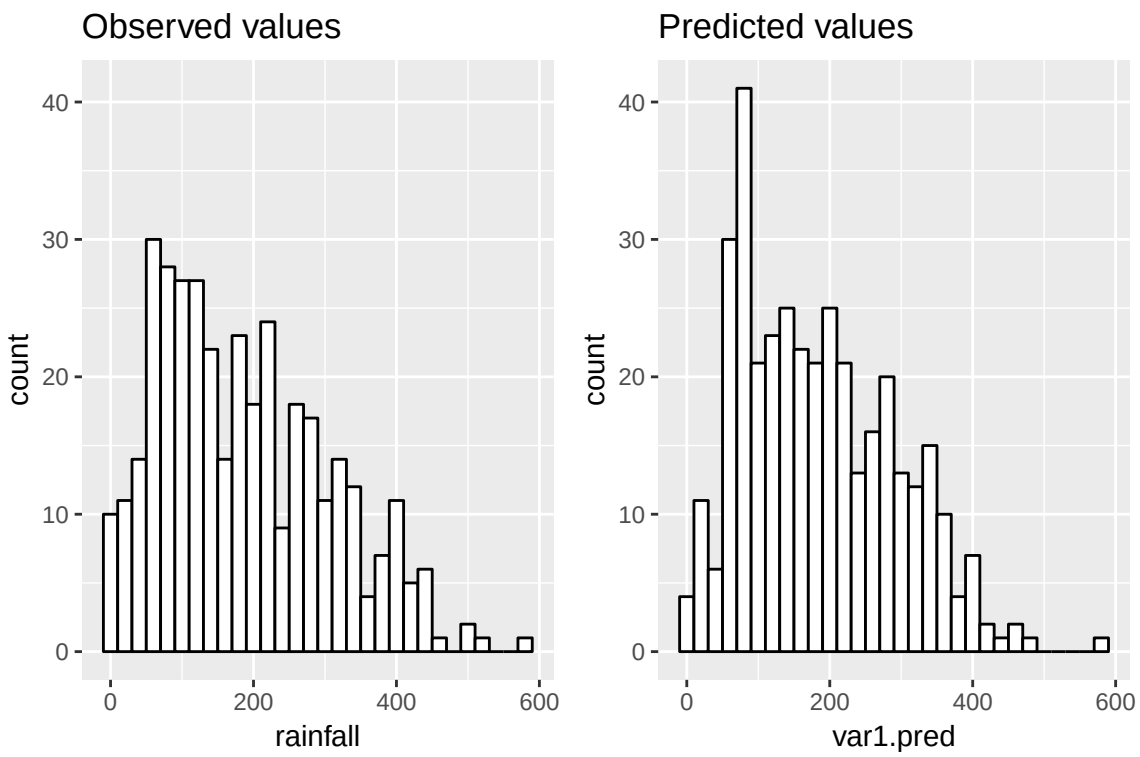


Figure 6.11: Histogram

If we look at the output of the summary above but also at the histograms (figure 6.11) we can conclude that kriging gives a smooth prediction with less dispersion of the rainfall values.

```
# histogram of observations and predictions

g1 <- ggplot(data=as.data.frame(sic_nobs),aes(x=rainfall))+
  geom_histogram(binwidth = 20, color="black", fill="white")+
  labs(title="Observed values")+
  ylim(0,41)
g2 <- ggplot(data=NF.kriged.nobs@data,aes(x=var1.pred))+
  geom_histogram(binwidth = 20, color="black", fill="white")+
  labs(title="Predicted values")+
  ylim(0,41)
gridExtra::grid.arrange(g1, g2, ncol=2)
```

Correlation between observed and predicted values If the predicted values are identical to the observed values, then their values should be on a 1:1 line. We can check this point by computing the correlation and by doing a linear regression of prediction over observation.

```
# Computation of the residuals
residuals=sic_nobs$rainfall-NF.kriged.nobs$var1.pred
# data handling
NF.kriged.nobs@data <- data.frame(NF.kriged.nobs@data,
                                  rainfall=sic_nobs$rainfall)
NF.kriged.nobs@data <- data.frame(NF.kriged.nobs@data,residuals)
# correlation between observation and prediction
cor(NF.kriged.nobs@data$rainfall,NF.kriged.nobs@data$var1.pred)

[1] 0.8936177

# variance of the residuals
var(NF.kriged.nobs@data$residuals)

[1] 2858.929

#
coef <- coef(lm(var1.pred~rainfall,data=NF.kriged.nobs@data))
```

The scatterplot in figure 6.12 with the regression line (in green) and the 1:1 line (in red) indicates a slight bias of the prediction.

```
ggplot(NF.kriged.nobs@data,aes(x=rainfall,y=var1.pred)) +
  geom_point() +
  geom_abline(slope =1, intercept = 0, col="red",size=2) +
  geom_abline(slope =coef[2], intercept = coef[1],
              col="green",size=2)
```

Residuals of the rainfall prediction Finally, we can have a look at the residuals in figure 6.13 of the prediction (we are in a special case where we know). Again, we observe a spatial structure in the residuals.

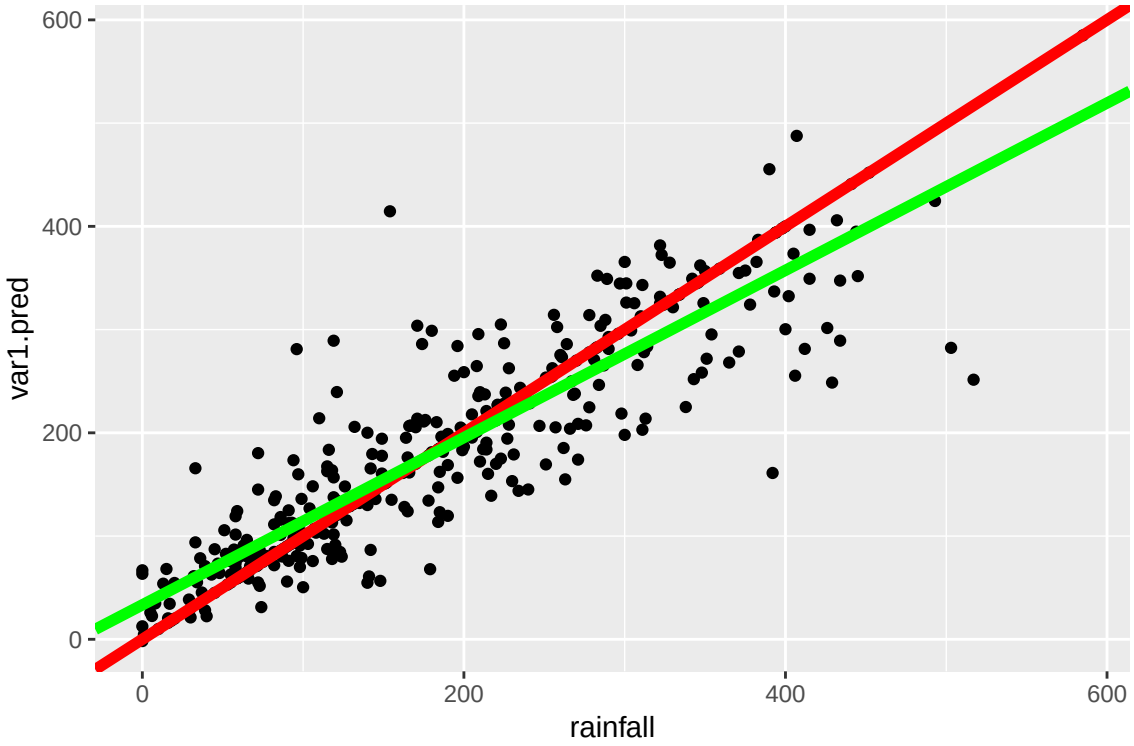


Figure 6.12: Scatterplot of the predictions versus observations with regression and 1:1 lines

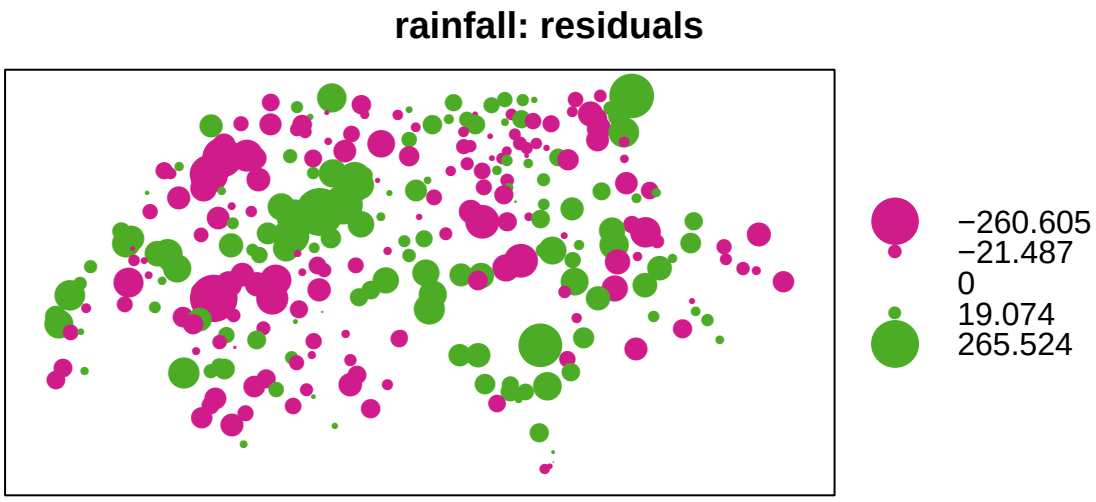


Figure 6.13: Residuals of the prediction with ordinary kriging

```
sp::bubble(NF.kriged.nobs, "residuals", main = "rainfall: residuals")
```

6.3 Sequential Gaussian Simulation

Goal: Estimate a characteristic or parameter of the RP Z_s, for example a probability map.

Principle

kriging gives an estimate of both the mean value and standard deviation of the normal (Gaussian) variable at each grid node.

Sequential Gaussian Simulation replaces the kriging mean value by a random draw from this normal distribution.

More details can be found in the book from Goaverts [?].

Normal Score Transformation for the Rainfall Example When data are not Normally distributed, the data can be transformed into normal scores before doing the sequential

Gaussian simulation on the normal scores. First, the rainfall dataset is normalized (centered and divided by its standard deviation).

```
myrain <- data.frame(rainfall=(mydata$sic_full.rainfall -
                                 mean(mydata$sic_full.rainfall))/
                        sd(mydata$sic_full.rainfall))
```

We can see from figure 6.14 that a rainfall of 210 millimeters corresponds to a scaled value of $(210 - 184)/112 = 0.23$ and a probability of 0.62. The normal quantile (or Normal Scaled Score) for this probability is 0.30. This empirical quantile (210) to normal quantile (0.30) transformation preserves the rank of an observation and therefore the probability level.

```
g1 <- ggplot(data=myrain,aes(x=rainfall))+
  geom_histogram(aes(y=..density..), binwidth = 1, color="white",
                 fill=rgb(0.2,0.7,0.1,0.4)) +
  xlim(-3.5,3)+
  stat_ecdf(geom = "step", pad = TRUE) +
  geom_segment(aes(x=0.23, xend=0.23, y=0, yend=0.62),
               size=1.5, col="red")+
  geom_segment(aes(x=0.23,xend=3,y=0.62,yend=0.62),
               size=1.5, col="red",
               arrow = arrow(length = unit(0.3, "cm"),type="closed"))

normdata <- data.frame(Normal.Scores=rnorm(n=467,mean=0,sd=1))

g2 <- ggplot(data=normdata,aes(x=Normal.Scores))+
  geom_histogram(aes(y=..density..),binwidth = 1, color="white",
                 fill=rgb(0.2,0.7,0.1,0.4))+
  xlim(-3.5,3)+
  stat_ecdf(geom = "step", pad = TRUE)+
    geom_segment(aes(x=-3.5, xend=qnorm(0.62), y=0.62, yend=0.62),
                 size=1.5, col="red")+
  geom_segment(aes(x=qnorm(0.62),xend=qnorm(0.62),y=0.62,yend=0),
               size=1.5, col="red",
               arrow = arrow(length = unit(0.3, "cm"),type="closed"))

gridExtra::grid.arrange(g1, g2, ncol=2)
```

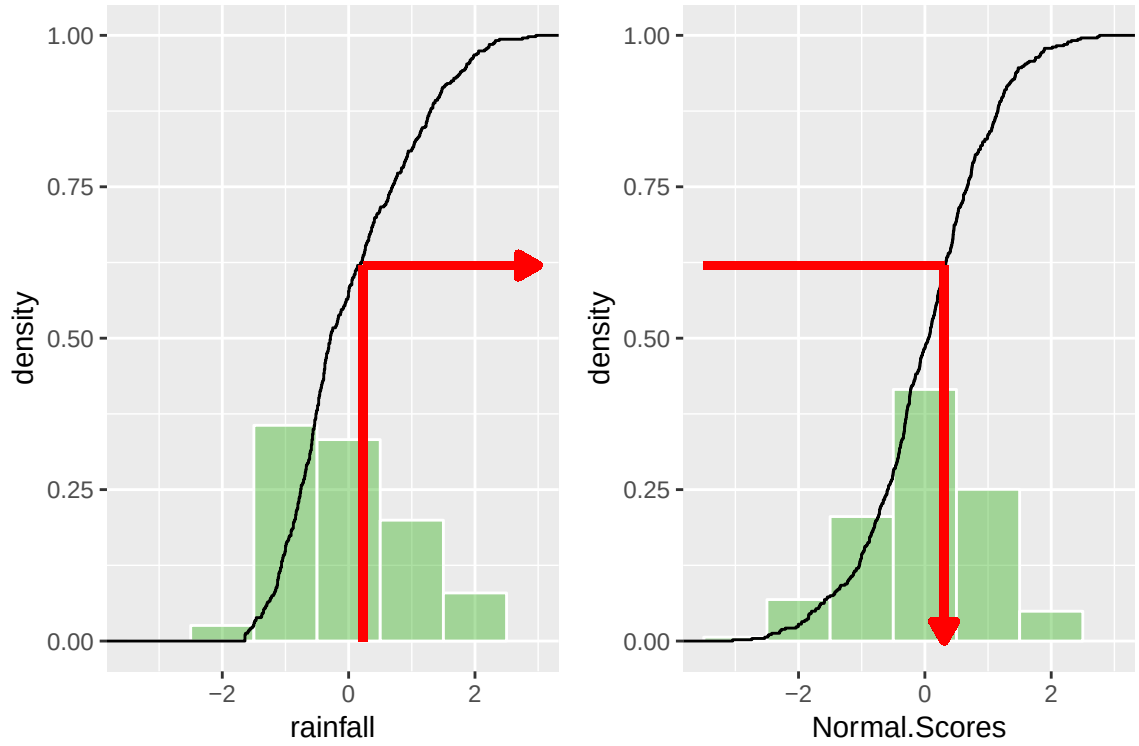


Figure 6.14: Illustration of the Normal Score Transformation for the Rainfall dataset

Algorithm of the SGS

1. Transform data to normal scores.
2. Perform a variogram analysis on the normal scores
3. Create a grid and generate a random path through the grid nodes.
4. Use kriging to estimate a mean value and standard deviation at the first node.
5. Set the variable value at that node from the random draw.

   Imagine that ordinary kriging gave a mean estimate of 0.23 with a standard deviation of 0.5 for the first node of the grid. Then the random draw for normal score is:

```
normal.score.scaled <- rnorm(n=1,mean=0.23, sd =0.5)
normal.score <- 184 + 112*normal.score.scaled
print(normal.score)

[1] 231.7708
```

6. Repeat for the next nodes, including previously simulated nodes as data values in the kriging process.

The previously simulated grid nodes are included as *data* in order to preserve the proper covariance structure between the simulated values.

Examples of simulated maps are given in figure 6.15. If we repeat this simulation not 4 but a hundred times then we can compute a probability map. For each node of the grid, we calculate the number of times the simulated rainfall values were above a certain limit (500 millimeters for example).

```
NF.kriged.sim = gstat::krige(rainfall ~ 1, sic_full,
                      mydata.grid, model = m.fit, nmax=20, nsim=4)
sp::spplot(NF.kriged.sim, main = "four conditional simulations")
```

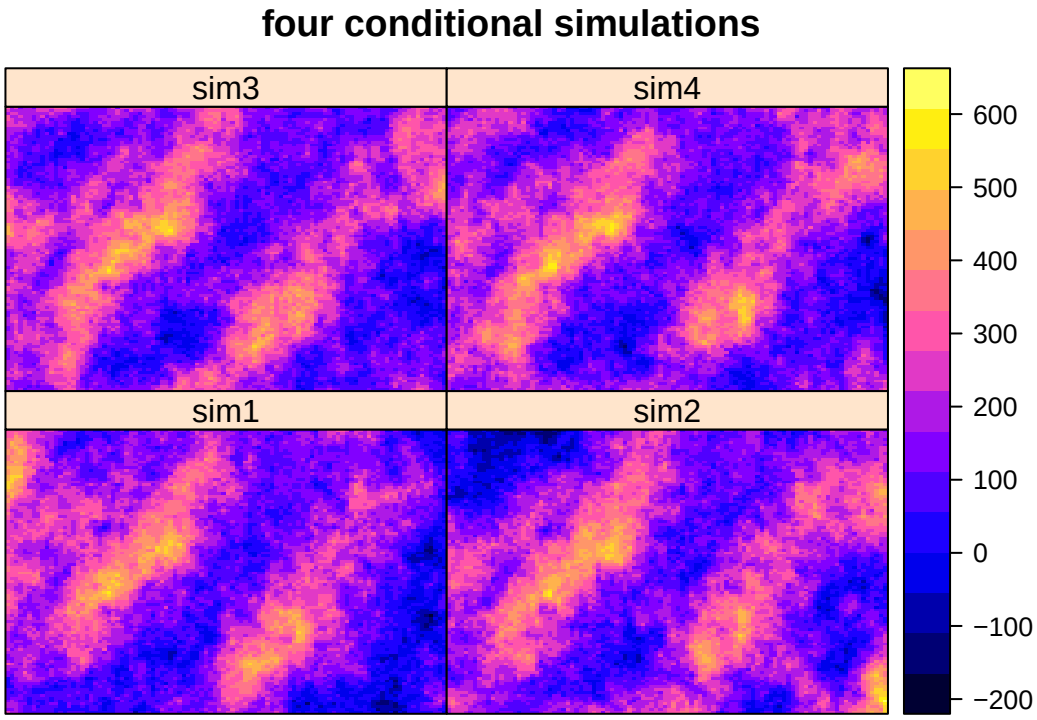


Figure 6.15: Sequential Gaussian Simulation for the Rainfall dataset

```
drawing 4 GLS realisations of beta...
[using conditional Gaussian simulation]
```

6.4 Co-Kriging

When non exhaustive secondary information is available, it can be incorporated into the estimator using the cokriging approach. This approach takes into account the secondary variables (from the RP $Z_{j,s}\, N_j,\, j = 2, ..., p$) and their spatial cross correlation with the primary variable (from the RP $Z_{1,s}$). It is possible for the secondary data to be at different sites.

$$\begin{aligned}\hat{Z}_1(s_{new}) - T_1(s_{new}) = & \sum_{i=1}^{N_{1,new}} \lambda_{1,i}(s_{new}) \left(Z_1(s_{1,i}) - T_1(s_{1,i})\right) \\ & + \sum_{j=1}^{p} \sum_{i=1}^{N_{j,new}} \lambda_{j,i}(s_{new}) \left(Z_j(s_{j,i}) - T_j(s_{j,i})\right)\end{aligned}$$

All cokriging estimators are required to be unbiased $E[\hat{Z}_1(s_{new}) - Z_1(s_{new})] = 0$ and to minimize the error variance $Var[\hat{Z}_1(s_{new}) - Z_1(s_{new})]$.

Each RP $Z_{j,s}$ is decomposed into a residual and a trend components:

$$Z_{j,s} = R_{j,s} + T_j(s),\ j = 1, \ldots, p$$

The residual component $R_{j,s}$ is modeled as a stationary RP with zero mean value and:

1. Covariance function: $Cov[R_j(s), R_j(s+h)] = C_j(h)$
2. Cross covariance function: $Cov[R_j(s), R_k(s+h)] = C_{jk}(h)$

In this section we will use the notion of cross-variograms (see section 4 for more details).

6.4.1 Example of Co-Kriging for Rainfall Data (SIC97)

6.4.1.1 Georeferencing of the Rainfall dataset

Read data set from gstat.

```
data(sic97)
suisse<-sic_full
class(suisse)

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

We need to declare a reference point: Geneve (index 435).

```
lat.geneve<-46.2 #N
lon.geneve<-6.1667 #E
x.geneve<-sp::coordinates(suisse)[435,1] # m
y.geneve<-sp::coordinates(suisse)[435,2] # m
```

Create a new CRS: LAEA projection at the location of Geneve (it could be a different projection!).

```
mycrs<-"+proj=laea +lat_0=46.2 +lon_0=6.1667
+x_0=0 +y_0=0 +ellps=GRS80 +units=m +no_defs"
```

Create new `SpatialPointsDataFrame` with a `CRS` for suisse. The reference point (Geneve) must have coordinates x=0, y=0

```
suisse2<-sp::SpatialPointsDataFrame(
  coords=cbind(coordinates(suisse)[,1]-
  proj4string=CRS(mycrs),data=suisse@data)
```

Check that it is the correct location with an interactive map (map not shown).

```
mapview::mapview(suisse2)
```

Locations in longitude/latitude to be able to download the correct elevation tiles (SRTM data).

```
suisse.lonlat<-sp::spTransform(suisse2,CRS("+init=epsg:4326"))
suisse.lonlat@bbox # extension in lon/lat


               min      max
coords.x1  6.166622 10.50516
coords.x2 45.797633 47.73390
```

Download elevation directly from internet with R commands from package **utils**. The R commands **download.file** and then **unzip** are run only once. The files will unzip and save in your working directory. The next time, you won't need this R commands and that's why they appear as comments (with # at the beginning) in the following script.

```
elev<-NULL
for (LONG in c("006","007","008","009","010"))
  {
  for (LAT in c("45","46","47"))
  {
    TILE<-paste0("N",LAT,"E",LONG)
#adress <- "http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/Eurasia/"
#urlzip<-paste0(adress,TILE,".hgt.zip")
#download.file(url=urlzip,destfile=paste0(TILE,".hgt.zip"),mode="wb")
#unzip(zipfile=paste0(TILE,".hgt.zip"))
    srtm<-raster::raster(paste0("datasets/",TILE,".hgt"))
    if (is.null(elev)) elev<-srtm
    if (!is.null(elev)) elev<-merge(elev,srtm)
  }
}
# Check that elev is a raster layer
elev


class       : RasterLayer
dimensions  : 3601, 6001, 21609601  (nrow, ncol, ncell)
resolution  : 0.0008333333, 0.0008333333  (x, y)
extent      : 5.999583, 11.00042, 44.99958, 48.00042  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
data source : in memory
```

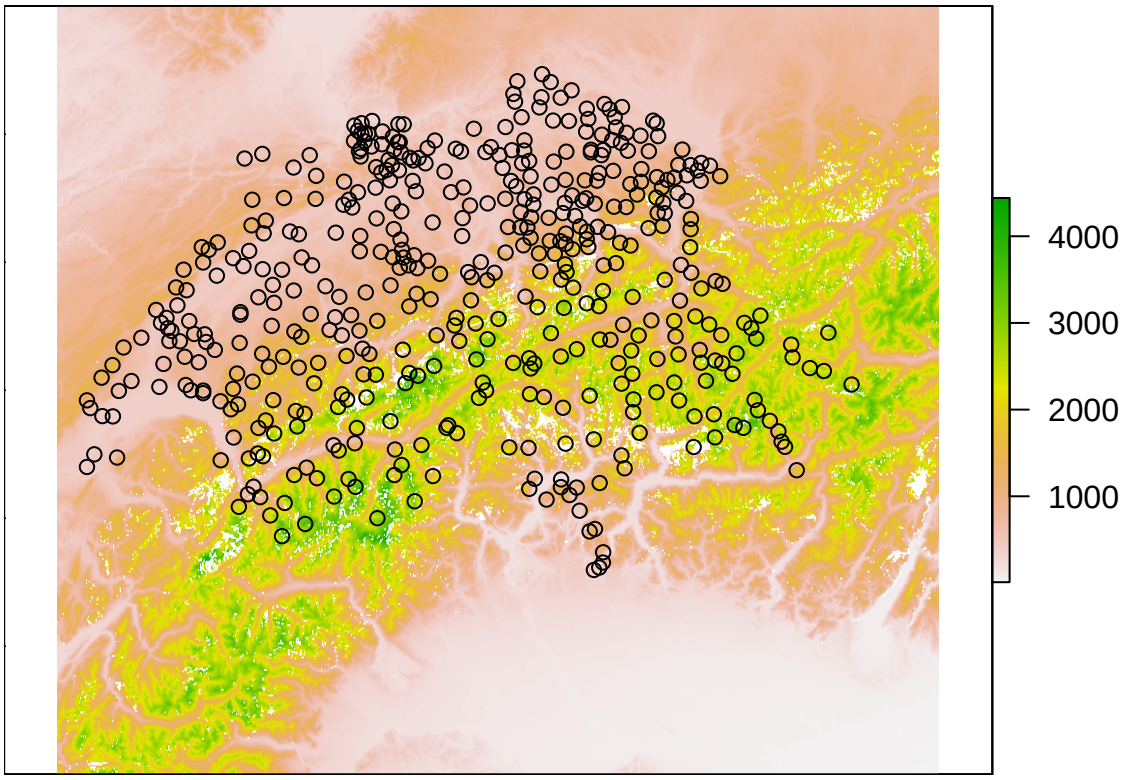


Figure 6.16: Elevation map from SRTM data with sample points of the SIC97 dataset

```
names       : layer
values      : -10, 4672  (min, max)
```

We can map the tiles and add our sample points on the map to check in figure 6.16 that the tiles cover all the sampled area.

```
# confirm that the DEM data covers all our precipitation dataset:
par(mfrow=c(1,1),mar=rep(0,4))
plot(elev)
xy<-coordinates(suisse.lonlat)
points(xy)
```

Extract elevation values from DEM at our locations

```
e<-raster::extract(elev,suisse.lonlat)
suisse$elev <- e
```

NOTE: there are NA values in elevation: possibly because the SRTM has missing values at some locations creating an undersampled problem for the co-kriging.

In the cokriging (multivariate case) the only known model is the "linear coregionalization model". A cross variogram model must be related to a specific pair of variograms. Cross variograms might be symmetric or not, but the linear corregionalization model forces the cross variograms to be symmetric.

In our example, the set of observed data should be the first hundred values of *suisse* `SpatialPointsDataFrame`. It may be that, the sample size is not enough to have correct experimental variograms. Therefore we use all the available data to model and fit the variograms and cross variogram.

```
sum(is.na(suisse@data$elev)) # 9 NA values


[1] 9


# to remove those locations from the data set:
suisse_obs<-suisse[!is.na(suisse@data$elev), ]
# needs enough sample size... cannot use
#suisse_obs <- suisse_obs[1:100,]
```

6.4.2 Co-kriging

Building a dataset for experimental variogram

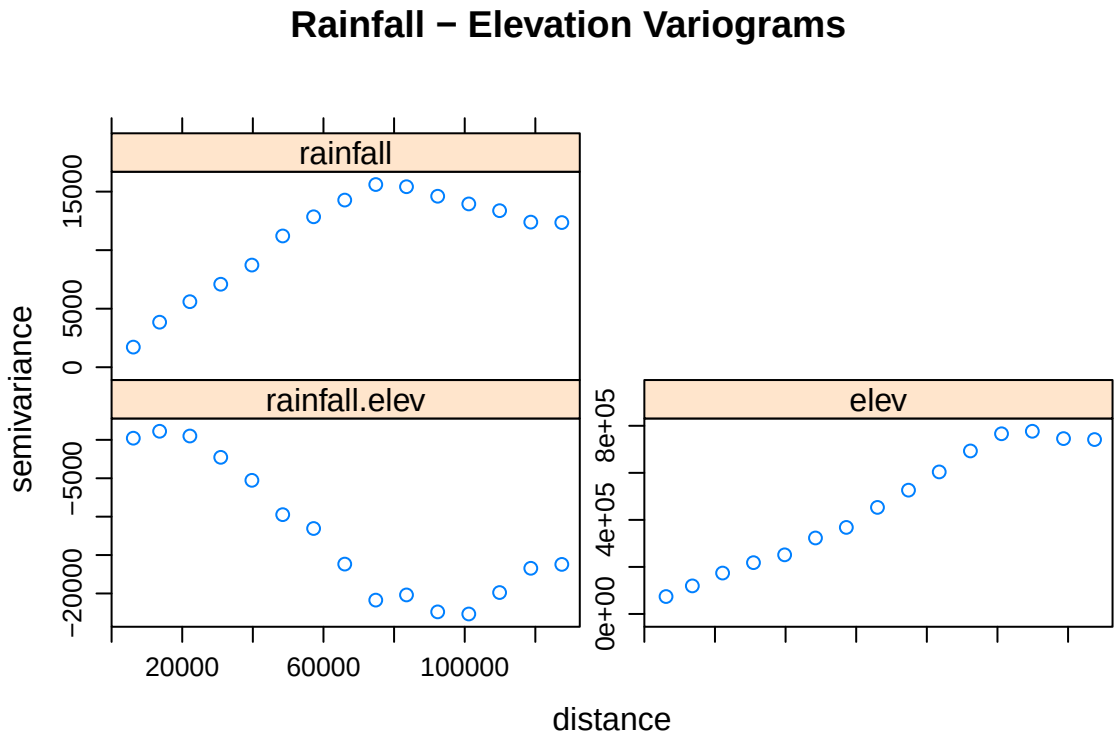
```
suisse.gs <- gstat(NULL,id="rainfall", formula = rainfall~1,
                   set = list(nocheck = 1), data=suisse_obs)
suisse.gs <- gstat(suisse.gs,id="elev", formula = elev~1,
                   set = list(nocheck = 1), data=suisse_obs)
```

The *set* option allows to go on with the prediction even if the coregionalisation model fails (warning message *non-positive definite coefficient matrix*). In our case, it seems that the

failure comes from the bad accuracy of the projection and therefore we see some incorrect elevation values (needs more reference points to improve the geolocalization).

Variograms

```
suisse.vg <- gstat::variogram(suisse.gs)
plot(suisse.vg, main='Rainfall - Elevation Variograms')
```

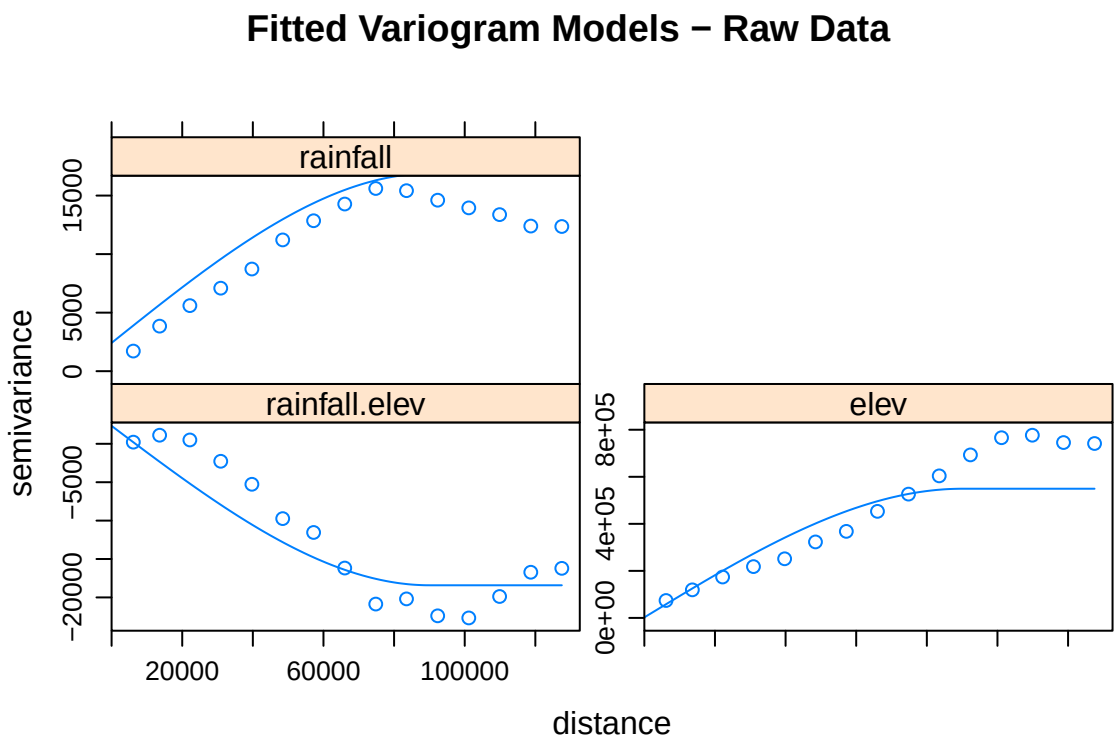


If we use a previous estimation for the rainfall variogram.

```
# Guess at a variogram model for each
# add the model to the gstat object
suisse.gs <- gstat(suisse.gs, model = vgm("Sph", nugget = 200,
                                          range = 90000, psill=15000),
                   fill.all=TRUE, set = list(nocheck = 1))
```

If we use the automatic fit and next a personal modification (from visual inspection of the experimental variograms).

```
# Cross-Variograms
suisse.fit <- gstat::fit.lmc(suisse.vg, suisse.gs, fit.lmc=TRUE)
plot(suisse.vg, model=suisse.fit,
     main="Fitted Variogram Models - Raw Data")
```




```
print(suisse.fit)


data:
rainfall : formula = rainfall`~`1 ; data dim = 458 x 3
elev : formula = elev`~`1 ; data dim = 458 x 3
variograms:
                 model      psill range
rainfall[1]        Nug   2417.400     0
rainfall[2]        Sph  14466.727 90000
elev[1]            Nug   2244.901     0
elev[2]            Sph 546966.257 90000
rainfall.elev[1]   Nug   2329.554     0
```

```
rainfall.elev[2]   Sph -20748.585 90000
set nocheck = 1;

# to modify
# suisse.fit$model$elev$psill[2] <- 600000      # nugget for elevation
# suisse.fit$model$elev$range[2] <- 100000       # psill for elevation
#etc.
```

Prediction and Interpolation: cokriging usually improves the error of prediction. In this example, be careful because we used all the data to model the variogram which induces underestimation of the error of prediction.

```
suisse_nobs <- sic_full[-(1:100),]

# predict to the non observed points
cok <- predict(suisse.fit, newdata=suisse_nobs,
               set = list(nocheck = 1))


non-positive definite coefficient matrix in structure 1Now checking for Cauchy-Schwar
variogram(var0,var1) passed Cauchy-Schwartz
[using ordinary cokriging]

# summarize predictions and their errors
summary(cok$rainfall.pred); summary(cok$rainfall.var)


   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0    91.0   171.0   186.9   270.5   585.0
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     0.0     0.0   114.3     0.0  6593.1
```

The rainfall prediction of the non observed points is given in figure 6.17. We can also do an interpolation map with cokriging.

```
# Interpolation map with mydata.grid
cok2 <- predict(suisse.fit,mydata.grid, set = list(nocheck = 1))
#Interpolation and Prediction
gridExtra::grid.arrange(spplot(cok2["rainfall.pred"]),
          spplot(cok["rainfall.pred"]),
             nrow=2)
```

```
non-positive definite coefficient matrix in structure 1Now checking for Cauchy-Schwar
variogram(var0,var1) passed Cauchy-Schwartz
[using ordinary cokriging]
```

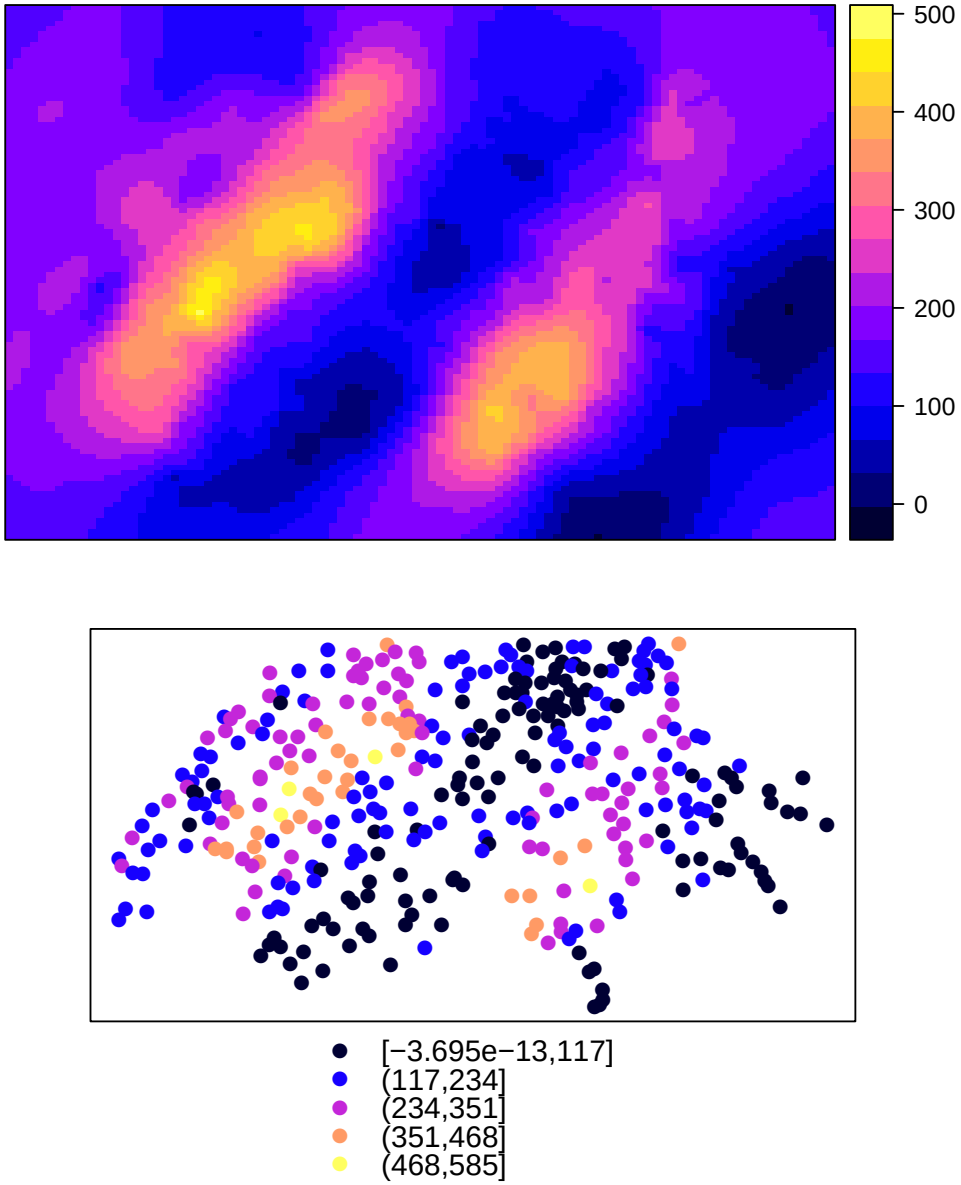


Figure 6.17: Interpolation Map and Prediction of the Rainfall Values with Rainfall and Elevation Co-Kriging

Chapter 7

Pattern Recognition for Spatial Data

Some parts of this chapter are inspired from the book *'Spatial Data Analysis In Ecology and Agriculture using R'*. R.E. Plant, CRC Press, 2012.

7.1 Introduction

7.1.1 Definitions

Pattern recognition has numerous definitions, with variations amongst authors, fields of application, origins, ...

We will use here a very general one, and will define it as *any method aiming at the recognition of patternds, regularities and hidden structures in data.*

This definition encompasses different goals, from the visualization, formalization and explaination of the pattern, to its extraction, prediction and application to new data. For the latter, it's linked to *machine learning*, with the supervised and unsupervised learning methods.

Unsupervised learning methods try to extract structures hidden in the data by finding similarities, relations, links between individuals, based on their observed features. They lead to the construction of groups, or subpopulations, where individuals are more closely related to the other individuals belonging to the same group than to the other groups.

Supervised learning methods aims at detecting pre existing structures and building predictors that can apply those structures to new data. The pre exisiting structure is generally a set of class or categories to which each individuals belong. Results of those methods can be for

example diagnostic tools that can identify the affection of a patient based historical records of diagnosis and patients medical data, or land use maps from existing records and satellite imagery.

"Machine learning" expression can be somewhat misguiding as most of those methods needs ans heavy human expertise input in their differents steps, from the selection of the training data sets to the choice of the methods and their parameters. So that, asn most of the data modelling techniques, pattern recognition stands between art and science.

Those choices will be guided by the data and the objectives of the analysis. Useful data in pattern recognition are often higly multidimensionnal, and available tools for pattern recognition in spatial data mostly the same than for "classical data". In one hand, the spatial information they include help supporting human decisions during the analysis and add new insights for the interpretation of the results. On the other hand, spatial data raises new questions about the nature of *individuals* which are the elementary data unit of most pattern recognition methods.

7.1.2 Important spatial data features for pattern recognition

Classical pattern recognition tools search for structures among data units, often called *individuals*. But the definition of an individual is particular with spatial data, as spatial data can be agregated to an arbitrary level (e.a. district, town, region, country, ...) or resolution. And the information linked to those data changes with the chosen unit. This is called the **modifiable areal unit problem**.

The modifiable areal unit problem can be illustrated by two effects : zonation and resolution.

7.1.2.1 Zonation effect

Let's take an example to illustrate the zonation effect. In forest science it's very common to evaluate the size of trees by measuring their circonference at breast height. Let's assume that this as be done for a portion of forest, and that the histogram in Figure 7.1 illustrate the frequency distribution of the circonference of the trees in that part of the forest.

As we can see, this distribution shows a nice exponential decrease, with a lot of small (and young) trees, and fewer and fewer trees as their size increase. Without any spatial information, foresters will often interpret this as a stand managed following selection cuttings, in equilibrium.

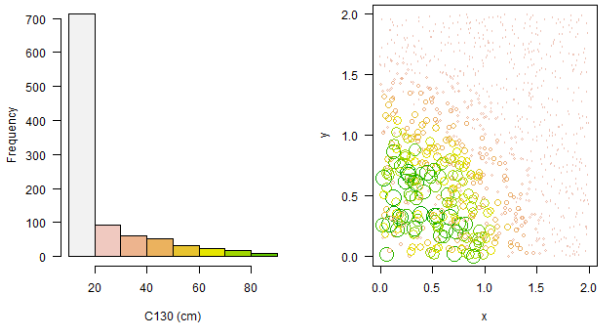


Figure 7.1: Left panel : Frequency distribution of the circonferences of trees in a hypothetical forest plot. Right panel : Spatial location of those trees in the plot (circle size is proportional to circonference)

But if we add the information about the location of each tree in the stand (Figure7.1, right panel), this illusion disappear, and we can see that this stand is in fact an edge between an old and a young even-aged forest stands.

This error of interpretation derive from the association of the trees' circonferences distribution to the whole forest stand, assuming the related information was homogeneous throughout this spatial unit. If it's not the case, the related information will highly depend on the level of aggregation of our data, as we can see in Figure 7.2, illustrating the average circonference of trees on pixels of various resolution, or Figure 7.3, showing the same histogram as before, but for two smaller spatial unit of the forest stand, which now clearly point the even-aged structure of those smaller units.

As we can see, the zonation effect arise each time we link information to a spatial unit of arbitrary size, assuming it is valid homogenously for its whole extent.

7.1.2.2 Resolution effect

The resolution effect is more directly related to raster data. When working with this type of data, resolution has two consequences. First, on the computation time, as the number of pixels is proportional to the squared resolution. This can lead to a rapid growth in computing time, as pattern recognition algorithms mostly have a complexity higher than linear. Second, on the information itself, as the decrease in resolution leads to a smoothing effect by averaging the information on a greater extent. The choice of a particular resolution,

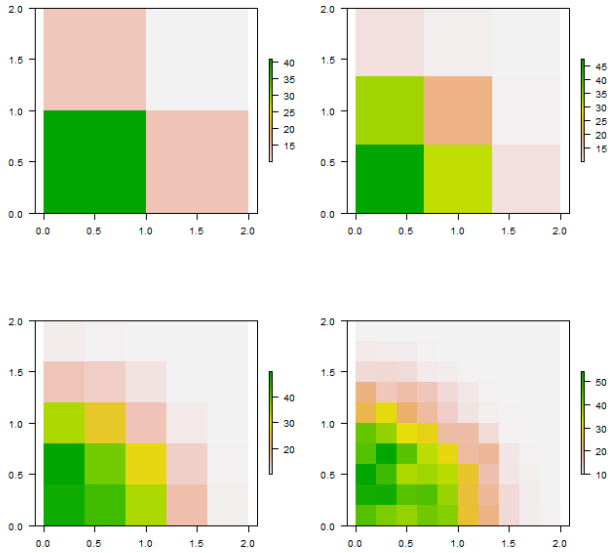


Figure 7.2: Rasters of average circonference of trees, with increasing resolution from top left to bottom right, revealing the edge structure of the forest plot

and the strength of the smoothing effect it generates, will strongly affect the results.

In classification problems this smooting effect can be valuable, as shown in the example of Figure 7.4. In this artificial example, four square regions have been generated from two populations with distinct means and a constant random noise. On the first panel, both populations are hard to separate because of the high overlap of their distribution. From left to right, resolution is halved at each step, and the resulting smoothing effect gets stronger. As a result, the distinction between the two populations gets clearer, the average difference staying constant, while the background noise decrease due to the smoothing effect.

As a consequence, an higher resolution is not always the best choice when starting a pattern recognition on raster data, and the smoothing effect is another argument to introduce in the final product resolution when crossing different data sources. Ideally, this resolution should be adjusted on the size of the object to classify, being high enough to describe the objects without blurring them with their background, and not to high to keep the computing time at an affordable level and benefitting from the smoothing effect.

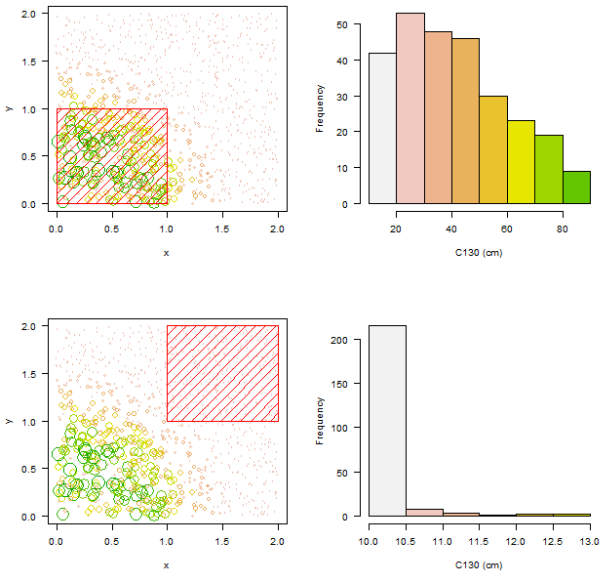


Figure 7.3: Frequency distribution of the circonferences of trees in subplots. Up : old forest subplot. Botton : young forest subplot)

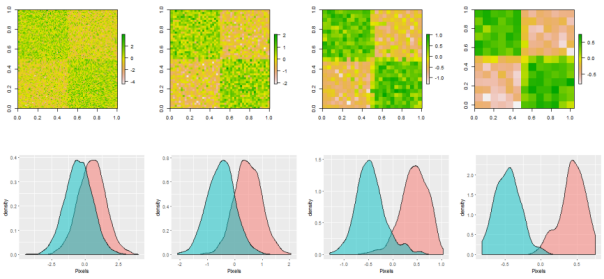


Figure 7.4: Artificial populations represented by rasters of decreasing resolution (each step halves the resolution). Lower panels, distribution of the values of the pixels of the two populations.

7.1.2.3 Data selection

Those considerations show that the choice of the data type is all but trivial in pattern recognition, and will strongly affect available data sources and the corresponding results. Will we manipulate vector objects, or raster pixels ? At which level of agregation/resolution ?

Once this choices have been made, linked data can be extracted to fuel the pattern recognition methods themselves.

In the following sections, we will discuss methods gathered around three global objectives :

1. Visualise and explore data
    - **Principal Component Analysis**
2. Find unrevealed structures
    - **Numerical classification methods**
3. Predict structures
    - **Discriminant analysis**

Appendix A

Simulation code for the AR(1) model

The R code used to simulate the AR(1) time autocorrelation model 2.8 is given below. It should be noted that the more compact matrix/vector versions of the model are not used, to save memory and CPU time.

```
> simulAR1 <- function(n, lambda, times, transient=0, mu=0, sigma=1){
tot=transient + n
simul <- matrix(nrow=times, ncol=n+2)
colnames(simul) <- c(paste("Y_", 1:n, sep=""), "Ybar", "S2")
if (transient > 0) lambvec <- lambda^((transient-1):0)
for (j in (1:times)){
eps <- rnorm(tot, sd=sigma)
eta <- eps[1]
if (transient > 0) eta <- sum(lambvec*eps[1:transient])
for (i in 1:n){
simul[j,i] <- mu + eta
eta <- eta*lambda + eps[transient+i]
}
simul[j, n+1] <- mean(simul[j,1:n])
simul[j, n+2] <- var(simul[j,1:n])
}
simul
}
```

The simulations discussed in Subsections 2.2.4 and 2.2.5 were obtained with the following function calls:

```
> simul.n10k <- simulAR1(n=10000,lambda=0.7,times=10000, transient=1000)
> simul.n1k <- simulAR1(n=1000,lambda=0.7,times=10000, transient=1000, mu=10, sigma=3)
```

Due to the random nature of the errors, results for new calls will, of course, differ.

Appendix B

Further elements on coordinate reference systems

A **map projection** is a method to produce all or part of a spheroid (ellipsoid of revolution) on a flat surface. More specifically, it transforms latitudes and longitudes of locations from the surface of a sphere or an ellipsoid into locations on a plane. Even if map projections are not in general perfect geometric projections, it is convenient to classify them according to the most similar geometric projection, which can be **azimuthal**, **conic** or **cylindrical** as illustrated in Figure B.1. Of course, any of those planar surfaces is placed relatively to the spheroid, which determines the points or lines over the surface that will remain undistorted under the projection.

A given shape over the spheroid has properties like distance between points, perimeter, area and so on. Therefore, it is crucial to know which properties, if any, are preserved under a given map projection. The main properties that are of interest for real applications are local shape (**conformal projection**), area (**equal-area projection**) and distance (**equidistant projection**). Since the spheroid is not a planar surface, it is known that no map projection can be both conformal and area preserving.

R, as many other open source applications, uses the PROJ syntax, developed under the PROJ.4 project, to describe coordinate reference systems. For instance, consider the PROJ description:

```
+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0  +units=m +no_defs
```

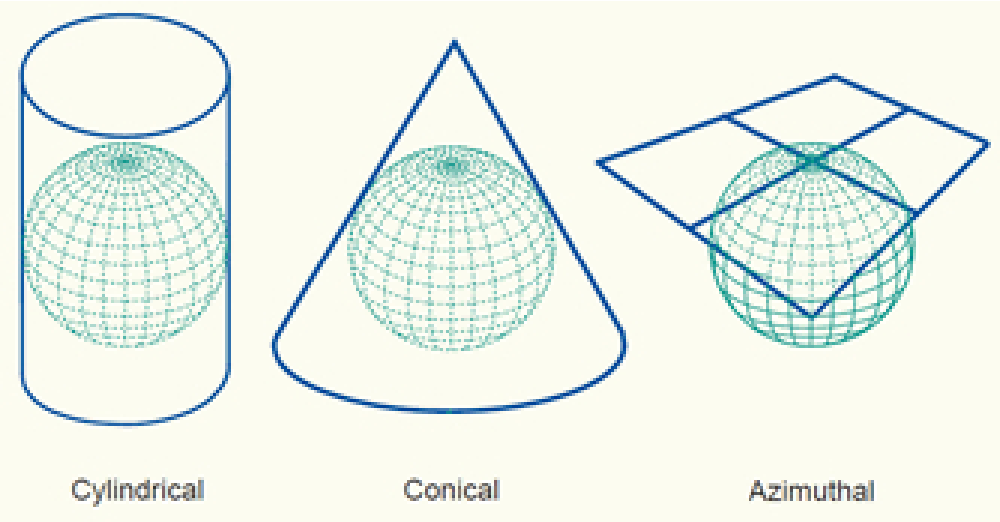


Figure B.1: Three basic geometric projections (Knippers, R. (2009, August). Geometric Aspects of Mapping. Retrieved from kartoweb.itc.nl/geometrics/).

The parameters of the CRS above have the following meaning:

1. `+proj=laea` indicates that it is a Lambert azimuthal equal-area map projection;
2. `+lat_0=52 +lon_0=10` are geographic coordinates of the origin of the projection, where the distortions vanish;
3. `+x_0=4321000 +y_0=3210000` are called *false easting* and *false northing* and they are the distances (m) from the origin $(x = 0, y = 0)$ of the cartographic coordinates to the origin of the projection;
4. `+ellps=GRS80` is the ellipsoid name;
5. `+towgs84=0,0,0,0,0,0,0` is the datum transformation parameters to WGS84 (see Figure **??**);
6. `+units=m` are the units in which are expressed the cartographic coordinates that are defined by the projection.

Most of the CRS in use have an EPSG name, which facilitates the identification of the CRS. For instance, the CRS in the example above can be identified by its EPSG code `epsg:3035` and the PROJ description reduces to just `+init=epsg:3035`.

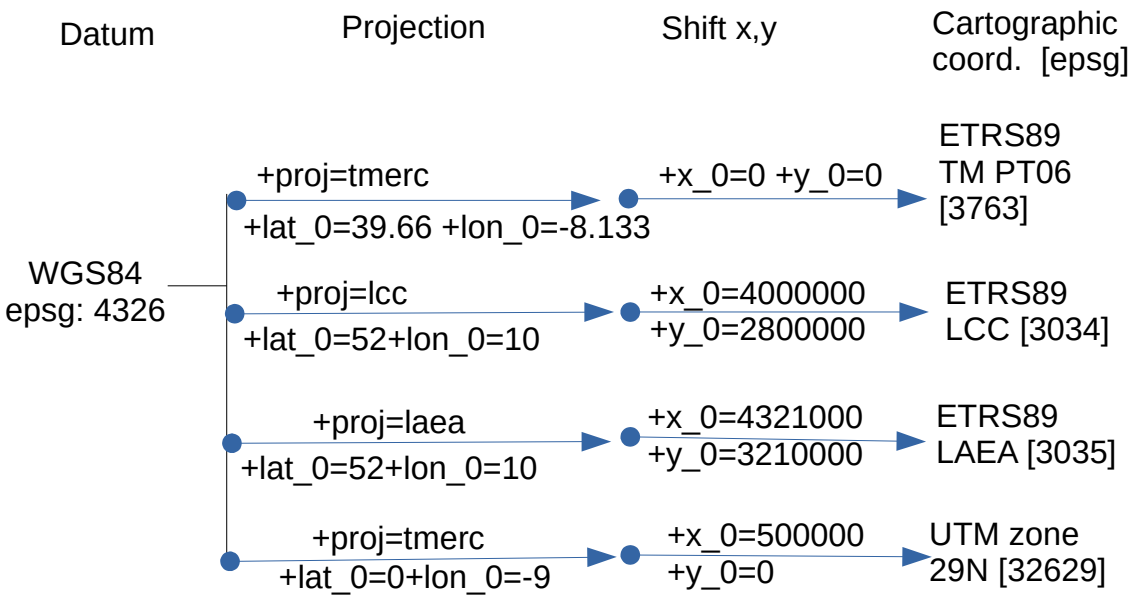


Figure B.2: A few CRS that are used in modern cartography for the EU and for Portugal in particular. ETRS89-TM-PT06 is the Portuguese zone of the ETRS89-TM family of CRS used in the EU, where TM indicates that it uses the transverse Mercator projection. LCC stands for the Lambert conformal conic projections and LAEA stands for Lambert azimuthal equal-area projection. Since this later one preserves areas, it is used for representing statistical data for the EU. The last CRS is zone 29 of the Universal Transverse Mercator (UTM) family of coordinate reference systems.

For the same regions of the world, different CRS can be used. For instance, three cartographic CRS were adopted in 2006 as official coordinate reference systems for Portugal following the recommendations of the EU. Figure B.2 describes those CRS and also describes the Universal Transverse Mercator (UTM) zone which includes Portugal. UTM is a family of CRS that is widely used for global data since distance distortions in each zone are always lower than 0.5%.

Appendix C

Maps and colors

Maps, like the ones produced by `plot` or `mapview` have default colors. Tipically, and among other fetures, one wishes to select a given palette of colors and intervals of values that correspond to each color.

R provides different ways of defining vectors of colors: those are just strings that represent colors using an hexadecimal notation color-hex. For instance, color *red* corresponds to code `#FF0000`, and *yellow* has code `#FFFF00`.

There are many ways of defining vectors of colors, as exemplified below. Parameter `alpha` in [0,1] indicates transparency, with opaque corresponding to `alpha=1`. One can visualize the resulting palette in Figure C.1.

```
library(randomcoloR) # color palettes
library(viridisLite) # colors (used in mapview)
mycolors<-c("red","yellow","green","blue")
mycolors<-colorRampPalette(c(rgb(0,0,1,alpha=1), rgb(0,0,1,alpha=0)), alpha = TRUE)(8
mycolors<-viridisLite::inferno(n=10,alpha=1,begin=0.2,end=1,direction= -1)
mycolors


 [1] "#FCFFA4FF" "#F5DC4DFF" "#FCAF13FF" "#F8850FFF" "#E8602CFF"
 [6] "#CF4446FF" "#AE305CFF" "#8B226AFF" "#66166EFF" "#420A68FF"
```

Note that if the `alpha` parameters is used, tranparency is encoded in the 7 and 8-th hexadecimal digits, with maximum value `FF` for opaque and minimum value `00` for transparent.

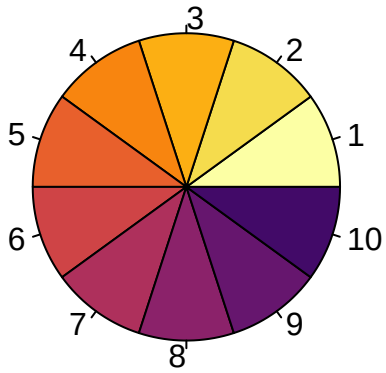


Figure C.1: Visualizing a set of colors with `pie`.

To associate k colors to the data, we need to define $k+1$ break values, with each color being assigned to the respective interval. In `mapview::mapview` this is done with argument `at` for the breaks and `col.regions` for the colors.

For instance, if we want to display the `ndvi` map in Section 3.6 with 4 classes between 0.1 and 1 then we need to choose at least 4 colors and the respective intervals as in the following example. A description of options for `mapview` can be found in mapview reference. If the number of colors is lower than the number of intervals, colors are recycled.

```
mycolors<-colorRampPalette(c(rgb(1,1,0,0.5), rgb(0,1,0,0.5)), alpha = TRUE)(4)
ndvic<-crop(ndvi,ndvi@extent/20)
```

For vector data, an in particular `sf` objects, some of the most commonly used options are `zcol` to indicate the attribute to be rendered, `col.regions` for the color palette (as above), `at` for the breakpoints (as above), `alpha.regions` for the opacity of the fills (0 is transparent and 1 is opaque), and `lwd` for the line width around the fills (0 for no line). One can add

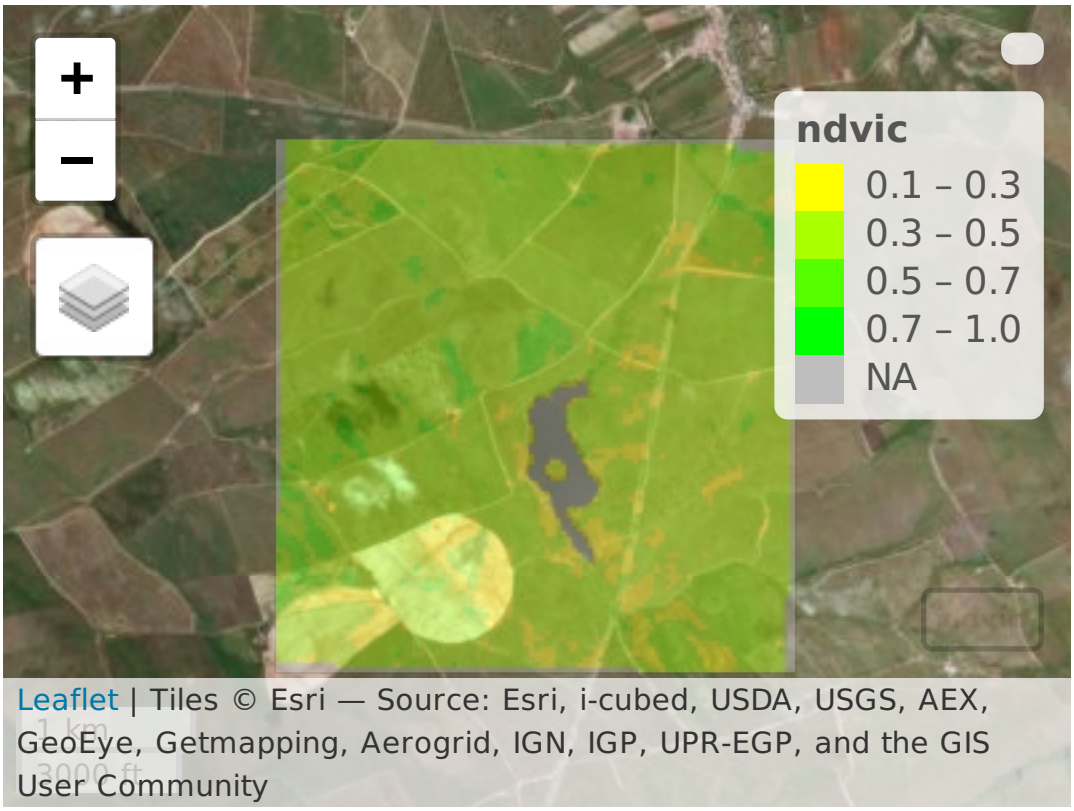


Figure C.2: NDVI image classified in 4 classes, using 4 selected colors.

features to mapview maps with `mapview::addFeatures`, and a variety of functions available in package `leaflet`, like `leaflet::addMarkers` exemplified below.

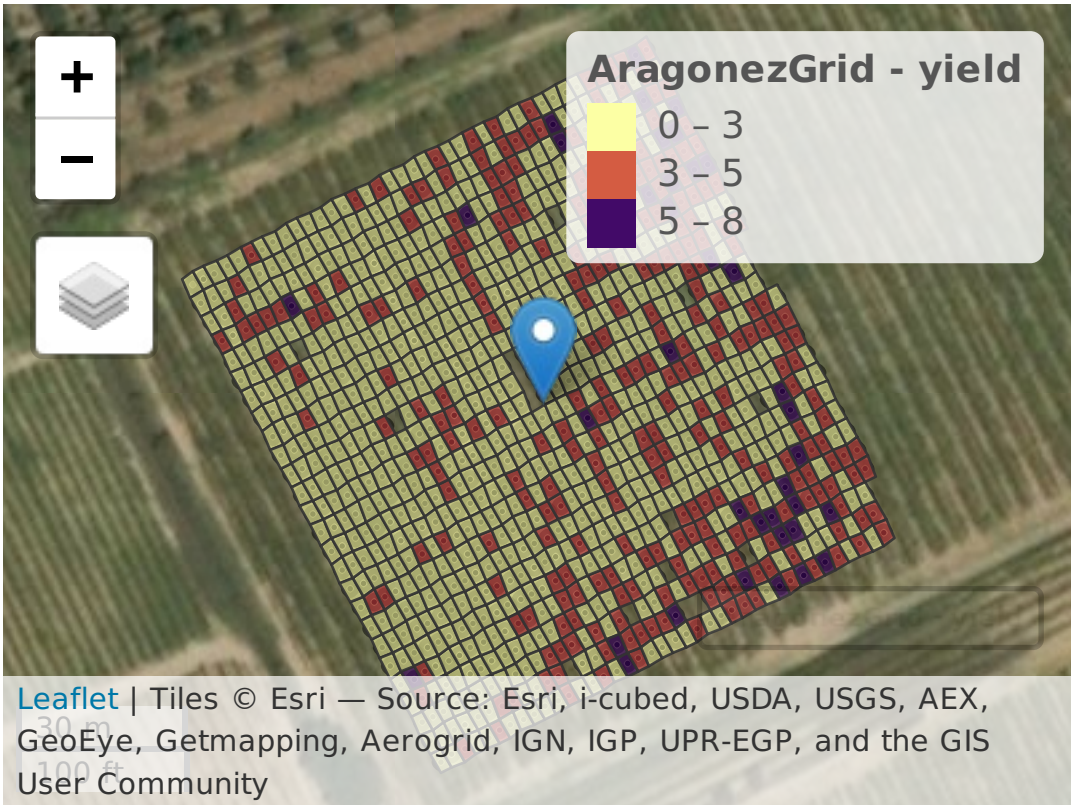


Figure C.3: Map of polygon vector data set with selected colors and additional features.

Appendix D

Effects of two-dimensional spatial autocorrelation

The effects of two-dimensional spatial autocorrelation on standard statistical methods are similar to those for one-dimensional autocorrelation discussed in Chapter 2. The presence of autocorrelation decreases the effective sample size, as there are no longer n *independent* sources of information. Thus, the standard statistical techniques which are derived under the assumption of independence will provide mistaken significance levels and p-values, as well as mistaken confidence levels for confidence intervals.

This can be seen by again considering the autocorrelated error model introduced in (4.4) or in (5.11), which extends the AR(1) autocorrelated error model discussed in Chapter 2. We *first re-write the AR(1) model*, as given in equations (2.12), using the random vector $\mathbf{Y}$ of n observations of a temporal process. Note that assuming *independent* Normal errors, with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ for all i, is equivalent to assuming that the random error vector $\boldsymbol{\epsilon}$ has a Multinormal distribution, with mean vector $E[\boldsymbol{\epsilon}] = \mathbf{0}$ and variance-covariance matrix $V[\boldsymbol{\epsilon}] = \sigma^2 \mathbf{I}_n$, where $\mathbf{I}_n$ is the $n \times n$ identity matrix. Hence, model (2.12) can be re-written as:

$$\begin{cases} \mathbf{Y} &= \mu \mathbf{1}_n + \mathbf{L}\boldsymbol{\epsilon} \\ \boldsymbol{\epsilon} &\sim \mathcal{N}_n(\mathbf{0}, \sigma^2 \mathbf{I}_n) \ . \end{cases} \tag{D.1}$$

where $\mathbf{1}_n$ denotes a vector of n ones, $\boldsymbol{\epsilon}$ denotes the vector of n random errors ϵ_i, and $\mathbf{L}$ is a (lower triangular) $n \times n$ matrix, whose i-th row is the vector $\boldsymbol{\lambda}_i$, that is, $\mathbf{L}$ is the matrix:

$$\mathbf{L} \quad = \quad \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ \lambda & 1 & 0 & 0 & \dots & 0 & 0 \\ \lambda^2 & \lambda & 1 & 0 & \dots & 0 & 0 \\ \lambda^3 & \lambda^2 & \lambda & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \lambda^{n-1} & \lambda^{n-2} & \lambda^{n-3} & \lambda^{n-4} & \dots & \lambda & 1 \end{bmatrix} \tag{D.2}$$

Vector $\mathbf{Y}$ is therefore a linear transformation of a Multinormal random vector $\boldsymbol{\epsilon}$. Such linear transformations preserve Multinormality, although the expected vector and (co-)variance matrix of $\mathbf{Y}$ are different from those of $\boldsymbol{\epsilon}$. These new distribution parameters can be calculated from the standard properties for the expected vectors and variance-covariance matrix of a linear transformation of a random vector. In fact, for any given random vector $\mathbf{X}$, constant vector $\mathbf{a}$ and constant matrix $\mathbf{B}$, we have:

$$E[\mathbf{a} + \mathbf{B}\mathbf{X}] \quad = \quad \mathbf{a} + \mathbf{B}\, E[\mathbf{X}] \tag{D.3}$$

$$V[\mathbf{a} + \mathbf{B}\mathbf{X}] \quad = \quad \mathbf{B}\, V[\mathbf{X}]\, \mathbf{B}^t \tag{D.4}$$

In our context, $\mathbf{X} = \boldsymbol{\epsilon}$, $\mathbf{B} = \mathbf{L}$ and $\mathbf{a} = \mu \mathbf{1}_n$, and so, taking into account (D.1):

$$\begin{array}{rcccccl} E[\mathbf{Y}] & = & E[\mu\mathbf{1}_n + \mathbf{L}\boldsymbol{\epsilon}] & = & \mu\mathbf{1}_n + \mathbf{L}\, E[\boldsymbol{\epsilon}] & = & \mu\mathbf{1}_n. \\ V[\mathbf{Y}] & = & V[\mu\mathbf{1}_n + \mathbf{L}\boldsymbol{\epsilon}] & = & \mathbf{L}\, V[\boldsymbol{\epsilon}]\, \mathbf{L}^t & = & \sigma^2 \mathbf{L}\mathbf{L}^t\ . \end{array}$$

Thus, the AR(1) time autocorrelation model (2.12) can be re-written in a single line:

$$\mathbf{Y} \ \sim \ \mathcal{N}_n(\mu\mathbf{1}_n, \sigma^2\, \mathbf{L}\mathbf{L}^t)\ . \tag{D.5}$$

With a transient period of length t, the size of all vectors would be $t + n$ and matrix $\mathbf{L}$ would be $(t + n) \times (t + n)$, but only the last n elements of the vectors, and the lower-right $n \times n$ submatrix of $\sigma^2\mathbf{L}\mathbf{L}^t$, would be of interest. Calling this $n \times n$ submatrix $\boldsymbol{\Sigma}$, and using the approximate post-transient expressions (2.17) for the variances and covariances between sample elements Y_{t+i} and Y_{t+j}, we have:

$$\mathbf{Y} \ \sim \ \mathcal{N}_n(\mu\mathbf{1}_n\, , \boldsymbol{\Sigma})\ , \tag{D.6}$$

where

$$\boldsymbol{\Sigma} \;=\; \frac{\sigma^2}{1-\lambda^2}\begin{bmatrix} 1 & \lambda & \lambda^2 & \lambda^3 & \dots & \lambda^{n-2} & \lambda^{n-1} \\ \lambda & 1 & \lambda & \lambda^2 & \dots & \lambda^{n-3} & \lambda^{n-2} \\ \lambda^2 & \lambda & 1 & \lambda & \dots & \lambda^{n-4} & \lambda^{n-3} \\ \lambda^3 & \lambda^2 & \lambda & 1 & \dots & \lambda^{n-5} & \lambda^{n-4} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \lambda^{n-1} & \lambda^{n-2} & \lambda^{n-3} & \lambda^{n-4} & \dots & \lambda & 1 \end{bmatrix} \tag{D.7}$$

Now we *consider again the 2-dimensional spatial process* introduced in equations (4.4) or in (5.11). This model can also be re-written using a vector/matrix notation. Denoting the random vector with the observed process Z_i as $\mathbf{Z}$, the vector of the autocorrelated process $\{\eta_i\}_{i=1}^{n}$ as $\boldsymbol{\eta}$ and the vector of the random errors as $\boldsymbol{\epsilon}$, we have an alternative model formulation:

$$\begin{cases} \mathbf{Z} \;=\; \mu\mathbf{1}_n + \boldsymbol{\eta} \\ \boldsymbol{\eta} \;=\; \lambda\mathbf{W}\boldsymbol{\eta} + \boldsymbol{\epsilon} \\ \boldsymbol{\epsilon} \;\sim\; \mathcal{N}_n(\mathbf{0}, \sigma^2\mathbf{I}_n) \end{cases} , \tag{D.8}$$

With a transient period of length t, $\mathbf{W}$ is a $(t+n)\times(t+n)$ matrix and only the post-transient part of the process Z is of interest. With no transience, matrix $\mathbf{W}$ is $n \times n$. In any case, the second equation in model (D.8) can be re-written (assuming the matrix inverse exists) as:

$$(\mathbf{I}_n - \lambda\mathbf{W})\boldsymbol{\eta} \;=\; \boldsymbol{\epsilon} \qquad \Leftrightarrow \qquad \boldsymbol{\eta} \;=\; (\mathbf{I}_n - \lambda\mathbf{W})^{-1}\boldsymbol{\epsilon}\ . \tag{D.9}$$

So the model under consideration, with spatially autocorrelated errors, becomes:

$$\begin{cases} \mathbf{Z} \;=\; \mu\mathbf{1}_n + (\mathbf{I}_n - \lambda\mathbf{W})^{-1}\boldsymbol{\epsilon} \\ \boldsymbol{\epsilon} \;\sim\; \mathcal{N}_n(\mathbf{0}, \sigma^2\mathbf{I}_n) \end{cases} . \tag{D.10}$$

This model for spatially autocorrelated errors is an extension of the (one-dimensional) AR(1) model (2.8), where matrix $(\mathbf{I}_n - \lambda\mathbf{W})^{-1}$ replaces matrix $\mathbf{L}$. In the AR(1) model each observation only depends on the observation that immediately preceded it, and so the spatial weights matrix $\mathbf{W}$ in the AR(1) model has all elements equal to zero, except for the sub-diagonal immediately beneath the main diagonal, where all elements would be 1. The inverse of the resulting matrix $\mathbf{I}_n - \lambda\mathbf{W}$ then has the form for $\mathbf{L}$ given in equation (D.2).

This 2-D error autocorrelation model can again be written in a single line, since it states that the observed vector $\mathbf{Z}$ has a Multinormal distribution, with parameters given by expressions (D.3) and (D.4). Specifically, the expected vector is $E[\mathbf{Z}] = \mu\mathbf{1}_n$, and the (co-)variance

matrix is given by:

$$V[\mathbf{Z}] = (\mathbf{I}_n - \lambda\mathbf{W})^{-1} \cdot \sigma^2\mathbf{I}_n \cdot \left[(\mathbf{I}_n - \lambda\mathbf{W})^{-1}\right]^t = \sigma^2(\mathbf{I}_n - \lambda\mathbf{W})^{-1}\left[(\mathbf{I}_n - \lambda\mathbf{W})^t\right]^{-1} \quad \text{(D.11)}$$

$$= \sigma^2\left[(\mathbf{I}_n - \lambda\mathbf{W})^t(\mathbf{I}_n - \lambda\mathbf{W})\right]^{-1} = \sigma^2\left[\mathbf{I}_n - \lambda\left(\mathbf{W} + \mathbf{W}^t\right) + \lambda^2\mathbf{W}^t\mathbf{W}\right]^{-1} \quad \text{(D.12)}$$

Thus:

$$\mathbf{Z} \sim \mathcal{N}_n\left(\mu\mathbf{1}_n\ ,\ \sigma^2\left[\mathbf{I}_n - \lambda\left(\mathbf{W} + \mathbf{W}^t\right) + \lambda^2\mathbf{W}^t\mathbf{W}\right]^{-1}\right) \quad \text{(D.13)}$$

Using this vector/matrix notation, the sample mean can be written as $\overline{Z} = \frac{1}{n}\mathbf{1}_n{}^t\mathbf{Z}$, since for any vector $\mathbf{Z}$, the inner product $\mathbf{1}_n{}^t\mathbf{Z}$ gives the sum of elements of $\mathbf{Z}$. Like any linear combination of the elements of a Multinormal vector, it will have a Normal distribution. Using the properties for expected values and variances, we have :

$$E[\overline{Z}] = \frac{1}{n}\,\mathbf{1}_n{}^t \cdot \mu\mathbf{1}_n = \mu\,\frac{1}{n}\,\mathbf{1}_n{}^t\mathbf{1}_n = \mu \quad \text{(D.14)}$$

$$V[\overline{Z}] = \frac{1}{n^2}\,\mathbf{1}_n{}^t\,V[\mathbf{Z}]\,\mathbf{1}_n\ . \quad \text{(D.15)}$$

Since for any matrix $\mathbf{B}$, the quadratic form $\mathbf{1}_n{}^t\mathbf{B}\mathbf{1}_n$ gives the sum of all elements in $\mathbf{B}$, under the model D.13, the sample mean $\overline{Z}$ has the following distribution:

$$\overline{Z} \sim \mathcal{N}\left(\mu\,,\,\frac{sum(V[\mathbf{Z}])}{n^2}\right)\ , \quad \text{(D.16)}$$

where $sum(V[\mathbf{Z}])$ indicates the sum of all elements in the variance-covariance matrix given in equation (D.12). For $\lambda = 0$ (no spatial autocorrelation) this expression for $V[\overline{Z}]$ reverts back to $\frac{\sigma^2}{n}$, the result for independent samples. The above expression for the spatial autocorrelation model will depend on both λ and the weights matrix $\mathbf{W}$, but is in general different from the variance for independent samples. Equation (D.16) can be used to build confidence intervals for $\overline{Z}$, when λ and σ^2 are known.

This and other spatial correlation models will be further explored in Chapter 5.

Appendix E

Exercises

E.1 Exercices on geographical data sets with R

E.1.1 Create a simple polygon `sf` object from scratch

This simple exercise illustrates how to create a `sf` object from scratch. This helps to understand what is its data structure. We define some simple polygons and build a couple of `sf` objects using those polygons. Polygons are just 2-column matrices of coordinates where each row represents a vertex. One can also define a list of polygons: the first polygon in the list will be the exterior ring of the spatial region, and the following polygons in the list define holes. We suppose that the holes are in the interior of the region enclosed in the external ring and, therefore, each list represents a *spatially connected* region on the plane as illustrated in Figure E.1.

First, 5 polygons are defined and included in list, where each list is going to describe the geometry of one feature of the `sf` object.

```
p1 <- rbind(c(0,0),c(4,0),c(4,4),c(0,4),c(0,0)) # polygon
hole1 <- rbind(c(1,1),c(3,1),c(3,3),c(1,3),c(1,1)) # polygon
P1<-list(p1,hole1) # list of polygons
p2 <- rbind(c(5+0,0),c(5+4,0),c(5+4,4),c(5+0,4),c(5+0,0))
hole2 <- rbind(c(5+1,1),c(5+3,1),c(5+3,3),c(5+1,3),c(5+1,1))
P2<-list(p2,hole2)
p3 <- rbind(c(5+1,4+1),c(5+3,4+1),c(5+3,4+3),c(5+1,4+3),c(5+1,4+1))
P3<-list(p3)
```

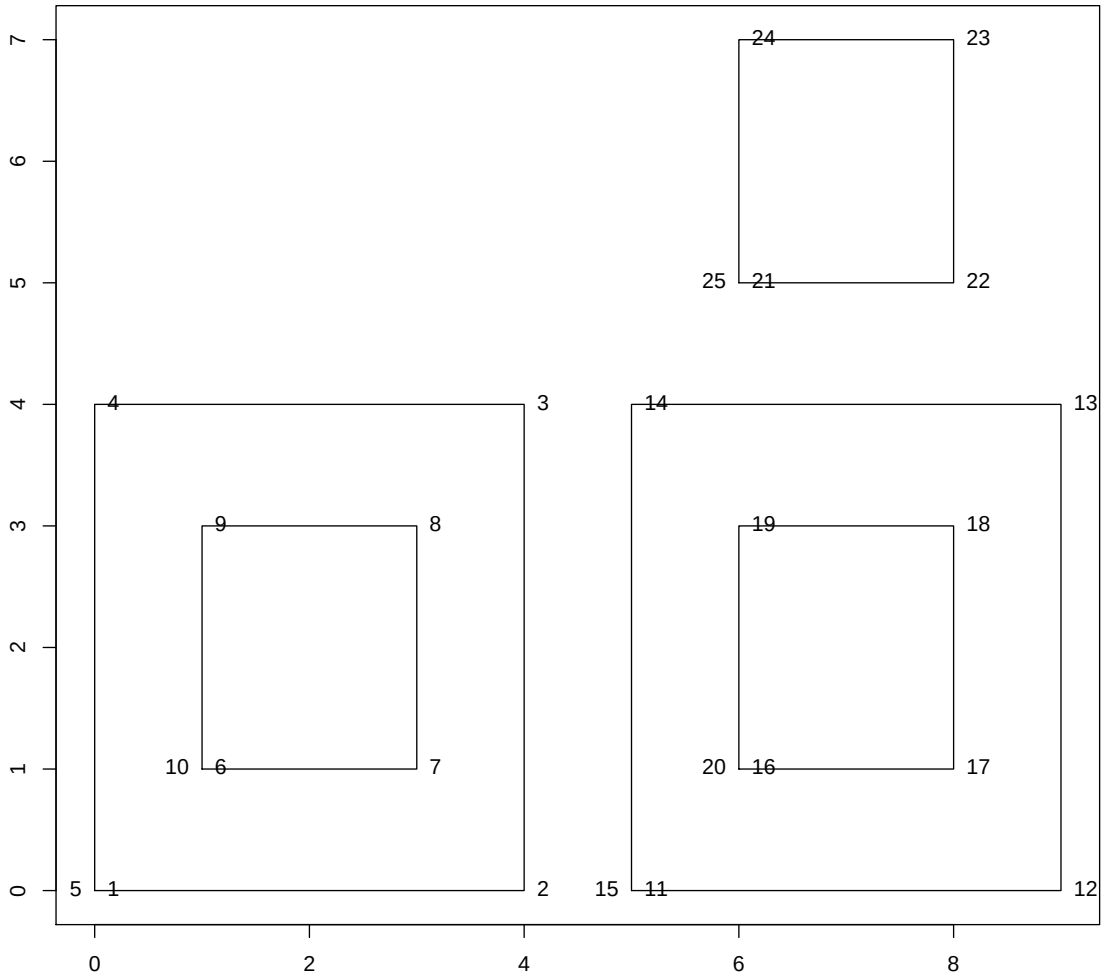


Figure E.1: `sf` object of geometry type POLYGON or MULTIPOLYGON to be created from scratch. The numbers indicate the indices of the vertices needed to define the geometries.

Let's now define a first `sf` object with three *features*. Features 1 and 2 have one hole, while feature 3 has just an exterior ring. Moreover, each feature can have *attributes* with values. Here we consider just one attribute named `code`.

```
L<-list(st_polygon(P1),st_polygon(P2), st_polygon(P3))
mysf <- st_sf(code = c(1:3), geometry = st_sfc(L)) # sf object
mysf

Simple feature collection with 3 features and 1 field
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 0 ymin: 0 xmax: 9 ymax: 7
epsg (SRID):    NA
proj4string:    NA
  code                       geometry
1    1 POLYGON ((0 0, 4 0, 4 4, 0 ...
2    2 POLYGON ((5 0, 9 0, 9 4, 5 ...
3    3 POLYGON ((6 5, 8 5, 8 7, 6 ...
```

The geometry can be displayed in "well known text" format with `st_as_text`.

```
st_as_text(st_geometry(mysf))

[1] "POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 3 1, 3 3, 1 3, 1 1))"
[2] "POLYGON ((5 0, 9 0, 9 4, 5 4, 5 0), (6 1, 8 1, 8 3, 6 3, 6 1))"
[3] "POLYGON ((6 5, 8 5, 8 7, 6 7, 6 5))"
```

One main feature of `sf` objects is that they are organized as tables (it is called a *tidy* format) where each row corresponds to one single feature and with a special column called `geometry` that determines the geometry of each feature.

The column `geometry` of `mysf` holds the spatial geometry of the features. This is a `sfc` (sf column) object, which is a list of `sfg` (sfg stands for sf geometry) objects with a given geometry. In short, three classes are used to represent simple features: `sf` for the table (a `data.frame`), `sfc` for the list-column set of geometries, and `sfg` for the geometry of each individual feature.

In the example above, all features have the same geometric type with is `POLYGON`, which implies, in particular, that every feature is spatially connected. Function `sf::st_coordinates` not only returns the coordinates of all vertices, but also indicates the index of the feature (`L2`) and the index of the ring (`L1`) for each one of them.

```
st_coordinates(mysf)[c(5,6,10,11),]

     X Y L1 L2
[1,] 0 0  1  1
[2,] 1 1  2  1
[3,] 1 1  2  1
[4,] 5 0  1  2
```

Now, let's consider a diffrent way of representing the same region, but now with only two features, where the second feature represents both objects P2 and P3. This new feature is not spatially connected anymore, since P2 and P3 do not intersect: it is of geometric type MULTIPOLYGON, which is a collection of objects of type POLYGON.

```
L<-list(st_polygon(P1),st_multipolygon(list(P2,P3)))
mymsf <- st_sf(value = c(1:2),geometry = st_sfc(L))
st_geometry_type(mymsf)

[1] POLYGON      MULTIPOLYGON
18 Levels: GEOMETRY POINT LINESTRING POLYGON ... TRIANGLE

st_as_text(st_geometry(mymsf))

[1] "POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 3 1, 3 3, 1 3, 1 1))"
[2] "MULTIPOLYGON (((5 0, 9 0, 9 4, 5 4, 5 0), (6 1, 8 1, 8 3, 6 3, 6 1)), ((6 5, 8 5
```

Since `mymsf` is of mixed type (we say that it is a COLLECTION), we cannot extract all its coordinates at once. However, `sf` contains an extremely useful function called `st_cast` that casts the objects from one geometric type to another one. This can be used to convert `mymsf` into a new object of single type MULTIPOLYGON.

```
newsf<-st_cast(mymsf,to="MULTIPOLYGON")
st_geometry_type(newsf)

[1] MULTIPOLYGON MULTIPOLYGON
18 Levels: GEOMETRY POINT LINESTRING POLYGON ... TRIANGLE

st_coordinates(newsf)[c(5,6,20,21),]

     X Y L1 L2 L3
[1,] 0 0  1  1  1
[2,] 1 1  2  1  1
[3,] 6 1  2  1  2
[4,] 6 5  1  2  2
```

Note that there are now three levels for each vertex: the ring to which it belongs (L1), the part (L2) and the feature (L3).

To complete the definition of the `sf` objects and to be able to combine them with other geographical data sets, we need to associate a coordinate reference system (CRS) to the data. Since this toy data set does not represent a real entity on the Earth surface, this is not going to match with an existing feature. Nevertheless, one can georeference it at an arbitray location. Here we choose a longitude and a latitude, and apply a straithforward azimuthal projection at that point. The final dimensions (in meters) correspond to the vertices' coordinates in `newsf`.

More specifically, the custom CRS below is an azimuthal projection (`laea` stands for Lambert Azimuthal Equal Area) at the point with coordinates `lon_0=-9.13464` and `lat_0=38.70769`.

```
st_crs(newsf)<-"+proj=laea +lon_0=-9.13464 +lat_0=38.70769 +x_0=0 +y_0=0 +ellps=WGS84
```

Since the data set now has a CRS, then it can be displayed with `mapview`.

```
mapviewOptions(basemaps="Esri.WorldImagery")
mapview(newsf)
```

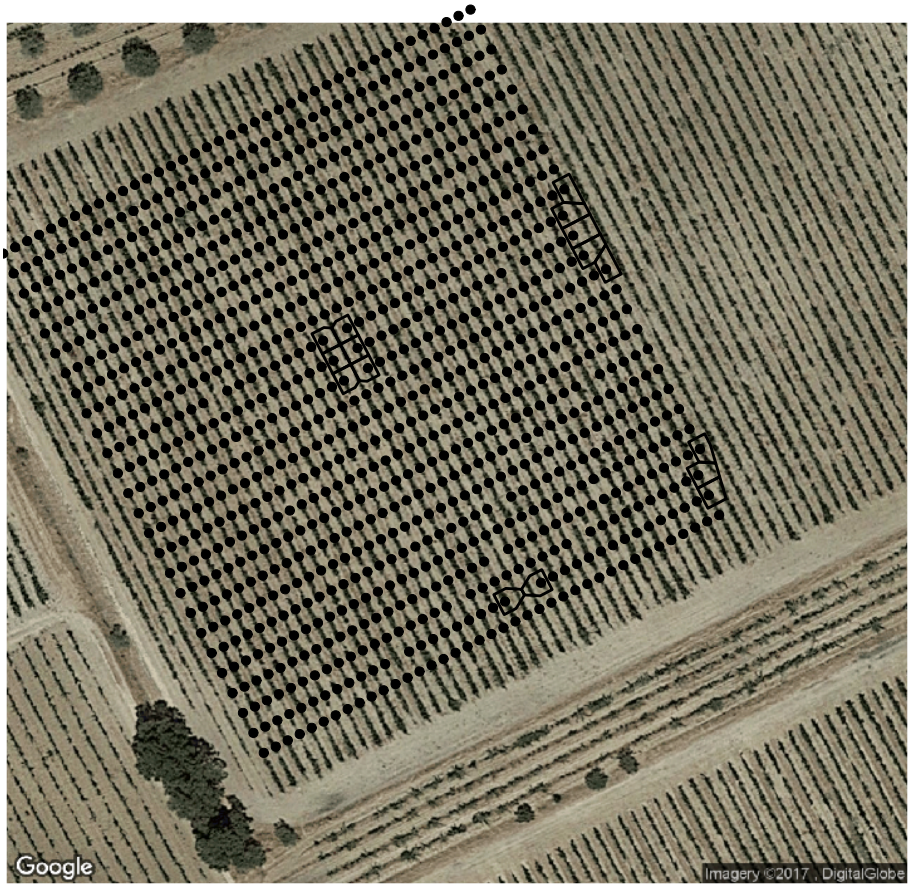


Figure E.2: Voronoi polygons with largest area for the Aragonez data set.

Finally, `newsf` can be exported as *shapefile*, *geopackage*, *kml* or other vector format with `st_write`. Available drivers can be listed with `st_drivers()`.

```
st_write(newsf, "newsf.shp") # the format is guessed from the file extension
```

E.1.2 Explore polygons that represent the Aragonez dataset

Consider the Voronoi polygons for the Aragonez data set determined in Section 3.1. Create a `sf` object with the polygons with the largest areas (due to missing neighbors). The result should be similar to Figure E.2.

E.1.3 Download, mosaic and analyze raster images

Elevation data for most locations on the Earth is readily available to download as seen in Section 3.7. For instance, SRTM3 elevations for Eurasia can be downloaded from site .

Those data sets are typically distributed in tiles of $1^o \times 1^o$. Suppose that you want to create a digital elevation model (DEM) for Continental Portugal, using the relevant tiles depicted in the figure bellow:

1. Write a R script to download all necessary tiles and merge them together to obtain an elevation model for Portugal;

2. Determine the location where the slope is steepest according to the DEM, and observe a high resolution image of that location.

Suggestions:

- To generate automatically a vector of tile names, one may want to consider the function `formatC`. For instance, `formatC(7,width = 3, flag = "0")` returns the string `"007"`;

- To merge two `RasterLayer` objects into one single one, one can use function `raster::merge`;

- To compute the slope, one can use function `raster::terrain`, as in Section 3.7.

E.1.4 Create a custom Buffer function

Function `sf::st_buffer` applied to an `sf` object returns a new `sf` with a modified geometry, where geometric features are expanded by a given distance (which can be a positive or negative, and can be constant or a vector of values). For instance, if it is applied to an approximately circular feature with radius 100 m (and area equals to 31415.93 m^2) and distance 10 m, the output will be a circular feature with radius 110 m and area equals to 38013.27 m^2.

Define a new function `myBuffer` which returns for each feature of the spatial object a new feature which area is *twice* the area of the original feature.

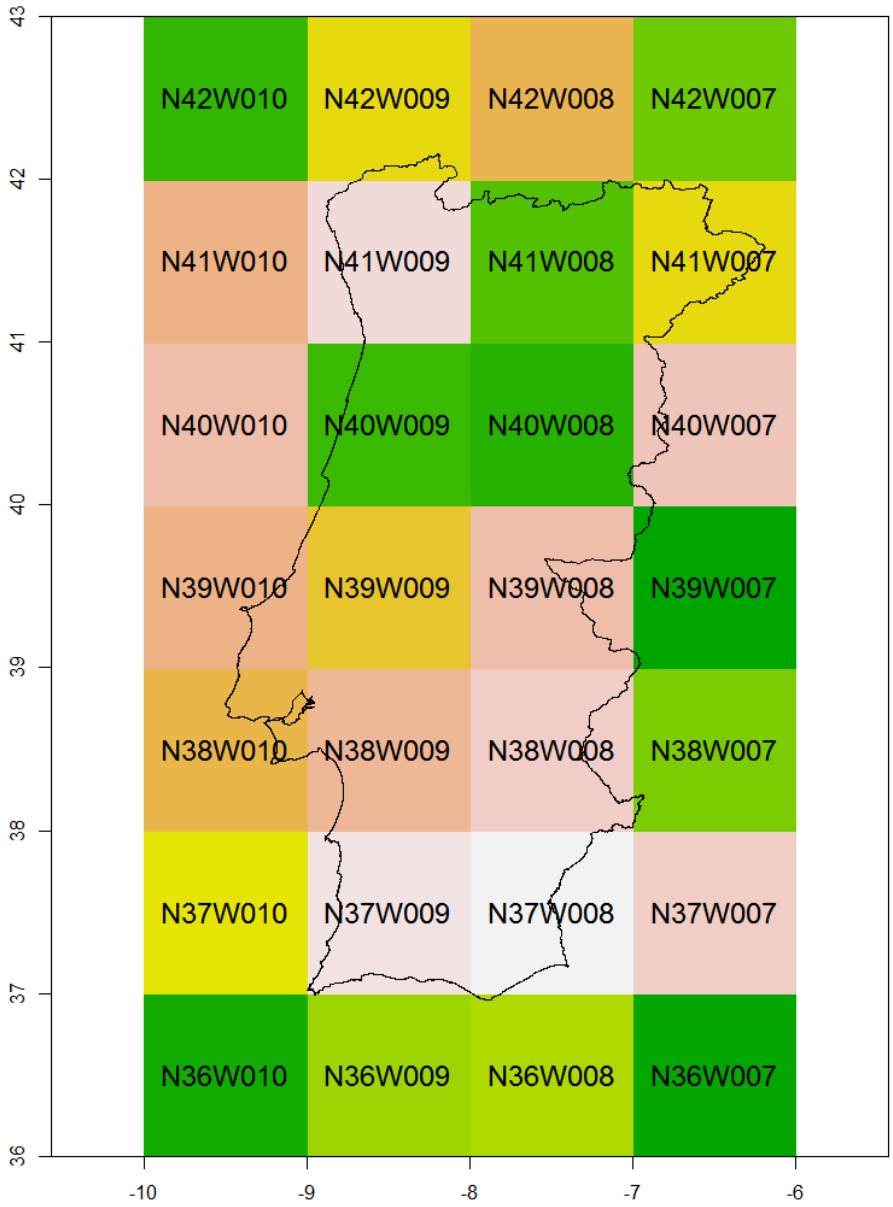


Figure E.3: STRM tiles for Continental Portugal

E.1.5 Create a neighbor data structure for a polygon spatial data set

Consider the regularly gridded Aragonez data set `AragonezGrid` created in Section 3.5. Define neighborhood relations for that grid according to some geometric/distance criteria, and create an object of class `knn` as described in Section 3.2 that represents the neighbors for `AragonezGrid`.

E.2 Further exercises on tools for spatial autocorrelation

E.2.1 The Arinto dataset

Arinto is a very popular Portuguese variety of grapes for white wines. The `Arinto` data frame, which can be downloaded from the course website, is a dataset that is similar to the Aragonez dataset. A field trial was set up in the Azeitão area of the Setúbal Peninsula, to the South of Lisbon. A vineyard trellis was set up, again with wires running in an approximately N-S direction. The 19 wires, numbered 52 to 70 and henceforth called columns, were separated by $2.75m$. Each wire was subdivided into "rows", that is, rectangles of height $3.2m$. The irregular contour of the field trial means that there are different numbers of rows in different parts of the trial fields, but the row numbers range from 1 to 61, with equal-numbered rows being adjacent in two adjacent columns. As with the Aragonez dataset, the main variable of interest is the `yield` of each rectangular cell, in what is approximately a rectangular lattice. For each lattice cell, the variables in the data frame are:

Name	Description
`genotype`	Genotype of the vines in the grid cell
`block`	The experimental design block to which the grid cell belongs
`col`	The column number (52 to 70)
`row`	The row number (1 to 61)
`colm`	The distance (in m) of the column to the reference point
`rowm`	The distance (in m) of the row to the reference point
`yield`	the yield (in kg/cell) of the grid cell.

As with the Aragonez dataset, genotypes and blocks will be ignored.

The overall purpose is to repeat the steps taken to analyse the yields in the Aragonez dataset, comparing the results in both cases. In particular:

1. Load the `Arinto.RData` file into an `R` session. You should have a data frame called `Arinto` available.

2. Create three new columns in the data frame, with the detrended yields that result from:

   - a constant trend;
   - a linear trend on the `col` (x) and `row` (y) coordinates;
   - a quadratic trend (second-degree polynomial) on the `col` (x) and `row` (y) coordinates.

3. Create an `sf` object using `colm` and `rowm` as x and y coordinates, respectively. Do not specify a `proj4string` argument (it will be defined as `NA`, but will not prevent the use of most `R` functions).

4. Create a `SpatialPointsDataFrame` object with data variables given by the original, and the three types of detrended yields.

5. Create bubble plots of the three types of detrended yields. Comment your results. Does there seem to be spatial autocorrelation, in each of the plots?

6. Use the `plot` or `spplot` command to simultaneously plot the three types of detrended yields. Compare the spatial patterns of the deviations from the three kinds of trends defined above. Comment your results.

7. Create an object of class `SpatialPolygonsDataFrame` with the yields and the detrended yields as data. Use the `spplot` function to view:

   (a) The three detrended yield variables;

   (b) The four (detrended or not) yields.

   Why is the second option not a good idea?

8. Using an appropriate `R` command, create a list of neighbours for each observed point, where neighbours are defined as all points at a distance no greater than 10 meters from each observed point. Taking into consideration the row-wise and column-wise separations between observed points, how many neighbours should there be for a typical point (where 'typical' means that it is not on the edges of the trial field)? Is this coherent with the number of non-zero links that is displayed by the `R` command that created your neighbour list?

9. Use Moran's I and Geary's c to decide whether spatial autocorrelation exists, using the neighbours defined above, and:

   (a) a binary weight matrix;

   (b) a row-normalized weight matrix.

10. Create a list of neighbours, using the $k=8$ nearest neighbours criterion. Based on the resulting `nb` list, plot Moran's correlograms of order 10, for the three types of detrended yields. Comment your results.

11. Consider again question 10, but now using a Geary's correlogram. Are the results coherent?

12. Compute and plot the empirical semi-variograms for the detrended yields, using the functions in the `gstat` package. Comment on the values obtained for the sill, nugget, partial sill and range, in each case. How do these results relate to those obtained above?

13. Choose a variogram model that you consider appropriate for the linearly detrended yields and fit it on the appropriate empirical semi-variogram. How good is the fit?

14. Repeat the previous question, but using the quadratically detrended yields.

15. Study the possible existence of anisotropy, using the `alpha` argument in the `variogram` function. Use the values 0 and 90 for the angles that define each direction. Why does the variogram associated with 90° drop off at a distance of about 50, while the variogram for 0° remain steady at a sill of approximately 0.27? Does anisotropy appear to exist?

E.2.2 The meteorological dataset

Consider the meteorological dataset described in Subsection 4.8.2 and which is made available on the course website, in a file called `meteo.RData`, which has a data frame called `meteo`.

1. Create a new data frame with units that are better suited for human interpretation of the variables: degrees Celsius for the three temperatures, hours for sunshine duration, and millimetres for total precipitation. Call the new data frame `meteo2`. Does this change affect any of the tools for spatial analysis that have been discussed so far? If so, in what ways?

2. It is natural to expect both a North-South gradient for variables such as temperatures, and an East-West gradient which to some extent coincides with a transition from Maritime to Continental weather. Create new columns in the `meteo2` data frame, in which each of the meteorological variables is detrended. Consider a linear trend on the geographical coordinates.

3. Create bubble plots of the linearly detrended variables. What conclusions are suggested by these bubble plots?

4. Create the empirical semi-variograms for each of the five detrended variables. Plot them, and comment your results.

5. Create the cross-variograms for all pairs of variables. Comment your results.

E.2.3 Working with NetCDF data

NetCDF stands for Network Common Data Format. It is a set of machine-independent data formats commonly used with scientific data. It is used by many websites that provide large datasets in areas such as climatology and oceanography, among them the ECMWF (European Centre for Medium-range Weather Forecasts) and NOAA (National Oceanic and Atmospheric Adminstration, in the US). There is an `R` package called `ncdf4` which provides an interface between NetCDF files and `R`.

1. Install the `ncdf4` package on your machine.

2. The NOAA website provides meteorological datasets that result from the processing of satellite data, often called *reanalysis data*. Go to NCEP-DOE Reanalysis 2: Gaussian Grid webpage. Read the general information on that page. Download the `air.2m.mon.mean.nc` NetCDF file, with information on monthly mean air temperatures (at 2m) for 473 months, on a world-wide grid of 192 longitude locations and 94 latitude locations.

3. Use the commands in the `R` package `ncdf4` to convert the NetCDF file, first into a 3-dimensional array in `R`, and then into a `SpatialPointsDataFrame` object, with longitude and latitude as coordinates and a data frame whose columns correspond to different months. In particular, explore the following commands:

   (a) `ncdf4::nc_open`, to open (attach to the `R` session) the NetCDF file that you downloaded. Save the result of your command in an `R` object called `air2m.nc`. Explore

its class and structure and familiarize yourself with the names that NetCDF file has for each of its variables.

(b) `ncdf4::ncvar_get`, to select and save variables from the NetCDF file as `R` objects. Explore the `start` and `count` arguments in the `ncvar_get` command, in order to select only a manageable subset of the full dataset, with a selected longitudes, b selected latitudes and all $a \times b$ air temperatures for each of c months (choose small values for a, b and c). In particular: (i) save the variable `air` (air temperatures) as an object called `air2m.air`, exploring its class and structure (it should be a three-dimensional array); (ii) save variable `lon` (the longitudes) as a vector called `air2m.lon`; (iii) save variable `lat` (the latitudes) as a vector called `air2m.lat`.

(c) `expand.grid`, a command from the base distribution of `R`, to create a two-column (ab rows) matrix with all the combinations of your selected longitudes and latitudes.

(d) `apply`, to create a matrix with ab rows and c columns, storing the ab values of air temperature for each given month in one of the columns. Use the `apply` command to convert each of the c elements in dimension 3 to a vector (using the `as.vector` command), that will be stored in each of the columns of the new matrix.

4. create a `SpatialPointsDataFrame` object with all the elements that you have created. Use the `CRS("+init=epsg:4326")` shorthand to specify `@proj4string` slot (with the longitudes and latitudes as coordinates in the WGS84 coordinate reference system).

5. view your `SpatialPointsDataFrame` object using `mapview::mapview`.

E.3 Mini-Project on Linear Model and Model Selection

The dataset has been provided by Bruno Tisseyre from Montpellier SupAgro. The data consist of physical and biochemical measurements taken on vines in a vineyard. For each measured vine the latitude and the longitude are known.

We are interested by the quality of the grapes for winemaking. An indicator of this quality can be the sugar contained in the grapes, and it is measured using degrees Brix (1 degree Brix is 1 gram of sucrose in 100 grams of solution).

The question of interest is whether or not this quality can be explained and/or estimated using yield, water status or other physical or biochemical variables.

You can load the data in R, using the file `exo_viticulture.csv`. Check the importation and the classes of the different variables. You can then specify that the variables `x` and `y` are

spatial coordinates. Your dataset will then be considered as a `SpatialPointsDataFrame`.

```
mydata <- read.table(file = "datasets/exo_viticulture.csv", header = TRUE,
                     sep = ";",dec=',')
str(mydata)
library(maptools)
coordinates(mydata) <- c("x", "y")
```

E.3.1 Graphical Representation and Summary of the Data

A first step in the analysis is to represent and summarize our data.

1. We can first summarize the dataset.
2. One of the simplest spatial representations is a bubble plot. The vines are represented by colors and bubbles whose size is proportional to measured degree Brix.

```
library(RColorBrewer)
spplot(mydata, "Brix", col.regions=brewer.pal(9,"Blues"),cex=0.3*(1:5),
       aspect=1, key.space="bottom", main="Brix")
```

3. Another spatial representation is a perspective plot. The vines are distributed on 15 `place` and 5 `row` (we can note that 2 `place` are empty), and the Brix values observed for these vines are represented from a perspective view. To do that, the Brix values observed are entered into a matrix. Then we decide to multiply by 3 the values of the `place` and `row` to have a better display.

```
Brix <- matrix(nrow = 15, ncol = 5)
for (i in 1:49){
  Brix[mydata$place[i],mydata$row[i]] <- mydata$Brix[i]
}
place <- 3 * 1:15
row <- 3 * 1:5
persp(place, row, Brix, theta = 40, phi = 20,
      zlim = c(16,25), scale = FALSE,cex.lab=1,ticktype="detailed",main="Brix")
```

4. Finally we can represent the possible scatterplots between the different variables, to examine possible correlations between these variables.

```
pairs(mydata@data[,1:13])
```

E.3.2 A First Linear Model

We want to explain and to estimate the quality of grapes (the `Brix` variable), using the 12 other physical and biochemical variables.

Fit the full model, that is the model with all the 12 explanatory variables included in the model. What do you think of this model?

```
mod1 <- lm(Brix ~ Pot_hydr + Berry_weight + pH + Acidity + IPT
           + Nb_berry_vinestock + Yield + Antho_grape + Antho_berry
           + SFEp + Circum + Vigour, data=mydata)
summary(mod1)
```

E.3.3 Model Selection to Explain the Quality of the Grapes

You can select a model using a model selection procedure. For instance, using the backward procedure and Fisher's tests (function `drop1`).

```
drop1(mod1, .~.,test="F")
```

E.3.4 Model Checking

We will now keep the model selected using backward procedure and Fisher's tests. To be able to test the different components of this model, some assumptions should be checked.

1. Write the equation of the model kept, and the associated assumptions on the residuals.
2. The first step to check the assumptions is to plot some figures dedicated to model diagnosis. Interpret these figures.

3. To check the homoscedasticity of the residuals, we can plot the residuals against the fitted values and against every possible explanatory variable.

4. We also need to check the independance of the residuals. What do you propose to check that?

E.3.5 Interpretation of the Model

If the assumptions of the model have been verified, you can examine the estimated coefficients of the kept model, and interpret them.

E.4 Practical work on regression models for spatially autocorrelated data

The data are explained and are available in the book *'Spatial Data Analysis In Ecology and Agriculture using R'*. R.E. Plant, CRC Press, 2012.
Data was collected as part of a four year study initiated in Winters, California, which is located in the Central Valley. The objective here is to determine if we can establish which of the measured explanatory variables influenced the observed yield variability in two fields. Here we will focus on the first year of experimentation, and on the first field. During this year, the first field was planted with wheat in December 1995 and harvested in May 1996. It was harvested with a harvester equipped with a yield monitor so a yield map of the field is available. Soil and plant data were collected by sampling on a square grid 61m in size, or about two sample points per hectare (hence 86 points).
The climate of California's Central Valley is Mediterranean, with hot, essentially rain-free summers and cool, wet winters. However, spring wheat was planted in an irrigated cropping system. Furthermore, the field was managed by a highly skilled farmer who used the best practices as recommended by the University of California.
The original analysis of this data set was carried out by Plant et al. (1999), and can be found in the book *'Spatial Data Analysis in Ecology and Agriculture using R'*. R.E. Plant, CRC Press, 2012.

The variables in this dataset are given in Table E.1.

To perform a statistical analysis on this data set, we have to take into account the fact that not all the variables are on the same resolution scale, and to take into account the spatial information:

Variable	Quantity represented	Type	Spatial resolution
`Sand`	Soil sand content (%)	Exogenous	Low
`Silt`	Soil silt content (%)	Exogenous	Low
`Clay`	Soil clay content (%)	Exogenous	Low
`SoilpH`	Soil pH	Exogenous	Low
`SoilTOC`	Soil total organic C (%)	Exogenous	Low
`SoilTN`	Soil total nitrogen (%)	Exogenous	Low
`SoilP`	Soil phosphorous content	Exogenous	Low
`SoilK`	Soil potassium content	Exogenous	Low
`Weeds`	Weeds level (1-5)	Exogenous	Low
`Disease`	Disease level (1-5)	Exogenous	Low
`CropDens`	Crop density (1-5)	Endogenous	Low
`leafN`	Leaf nitrogen content	Endogenous	Low
`FLN`	Flag leaf N content	Endogenous	Low
`SPAD`	Minolta SPAD reading	Gauge	Low
`GrainProt`	Grain protein (%)	Response	Low
`Yield`	Yield (kg ha^{-1})	Response	High
`Easting`	Easting coordinate	Coordinate	High
`Northing`	Northing coordinate	Coordinate	High

Table E.1: Variables in data set.

1. The variables `Yield` and the other variables are not on the same resolution scale. Concerning the `Yield` we have measurements on a map with 33183 points. Concerning the other variables they were collected by sampling on a square grid 61m in size, hence on 86 points. To be able to explain the yield with the other variables, the yield hase been interpolated on these 86 sample points. The method used is the interpolation with inverse weighted distance. The interpolated yield has been added to our dataset containing the low resolution variables. It has been scaled.

2. `Easting` and `Northing` variables are considered as spatial coordinate variables, the data set is a `SpatialPointsDataFrame` object.

You can load the dataset into R.

```
library(maptools)
load("datasets/Plant/Plant_dataset.Rdata")
```

E.4.1 Graphical Representation and Summary of the Data

A first step in the analysis is to represent and summarize our data.

1. We can first summarize the dataset.
2. We can give a rapid description of the dataset, for instance concerning the variables `Clay`, `SoilP`, `Weeds` and `Yield`, using boxplots or histograms.
3. We can take into account the spatial information. Figure E.4 shows a map of the soil types in the field. This map was constructed by R.E. Plant by downloading a shapefile of Soil Survey Geographic (SSURGO) soil classification data from the natural Ressource Conservation Service (NRCS) Soil Data Mart http://soildatamart.nrcs.usda.gov and clipping this shapefile in ArcGIS with that of the field boundary.
   The northernmost soil type is Capay silty clay (Ca). This soil is characterized by a low permeability. The soil type in the center is Brentwood silty clay loam (BrA), which is characterized by a moderate permeability. The soil in the south is Yolo silt loam (Ya), which is characterized as moderately permeable and well drained.

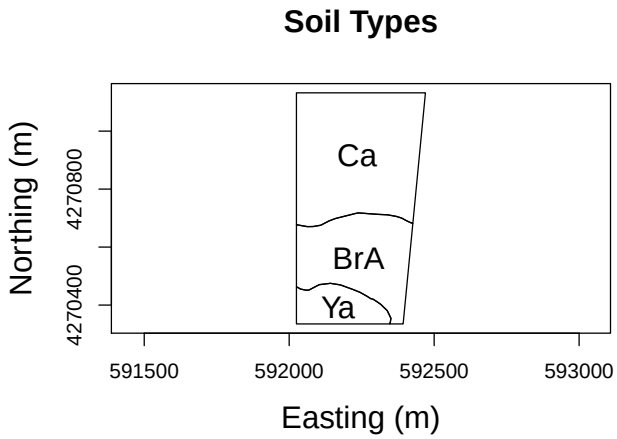


Figure E.4: Soil types in the field

We can take into account the spatial information for the low resolution variables (point sample data), see for instance Figure E.5 representing the variables `Sand`, `Clay`, `SoilP` and `Weeds`.

Figure E.6 represents the yield map (high resolution). The southern part of the field has a higher yield overall, with the southwest corner having the highest yield. The

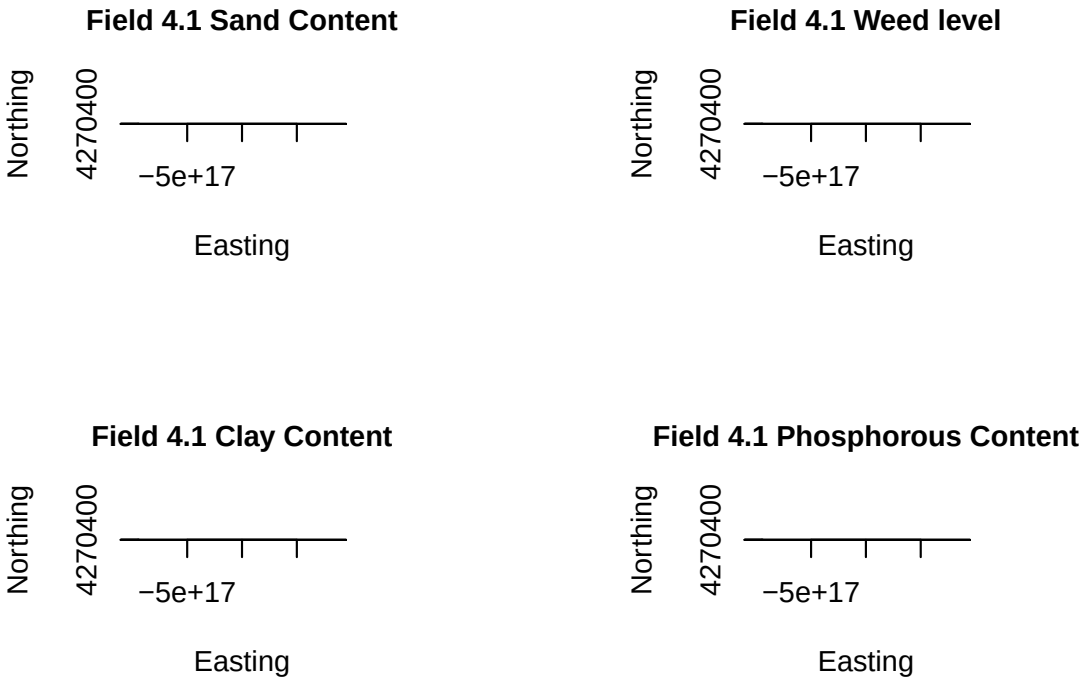


Figure E.5: `Sand`, `Clay`, `SoilP` and `Weeds` in the field

yield then generally appears fairly smooth except for two anomalously low areas: a triangular shaped region on the western edge, and the entire eastern edge.

Using `dataset`, you can represent the yield map in low resolution, using a bubble plot (Figure E.7).

```
library(RColorBrewer)
spplot(dataset, "Yield", col.regions=brewer.pal(9,"Oranges")[4:9],
       cex=0.5*(1:5),scales = list(draw = TRUE),xlab = "Easting",
       ylab = "Northing", main = "1996 Yield (low resolution)")
```

4. The preceding Figure suggests a spatial correlation of the yield. A way to visualize this correlation is to use a semi-variogram. In order to use a semi-variogram, the random spatial process studied (here the yield) must be second-order stationary. This

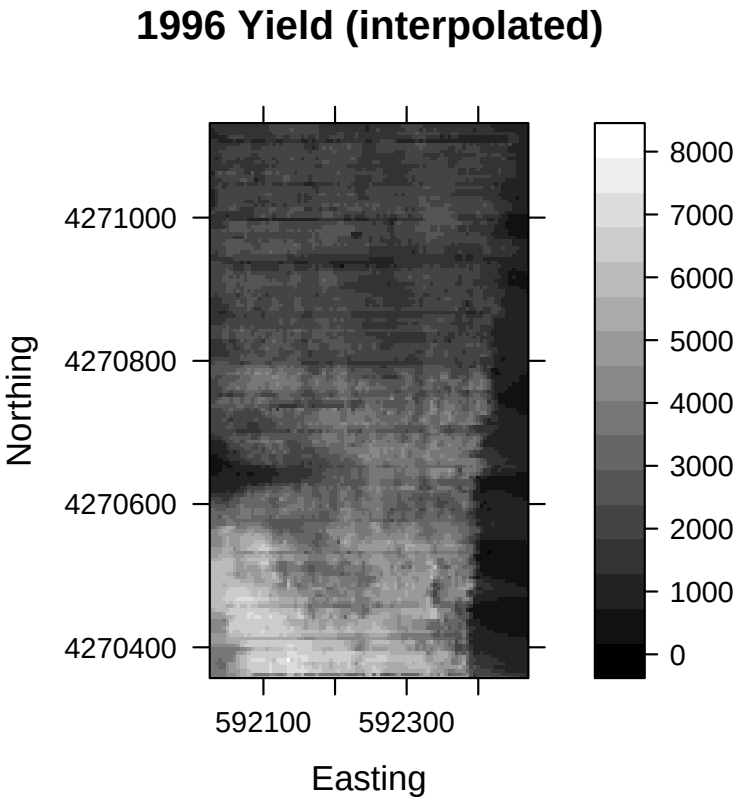


Figure E.6: Map of the yield

is obviously not the case here, hence detrended data must be used. We then need to detrend the yiel, that is to remove the spatial deterministic trend. What to you think of this trend and of the detrended data ?

```
TrendYield <- lm(Yield ~ I(Easting^2) + I(Northing^2) + I(Easting * Northing)
                 + Easting + Northing, dataset)
dataset$trend <- predict(TrendYield, dataset)
dataset$detrended <- residuals(TrendYield)
```

```
spplot(dataset, "trend", col.regions=brewer.pal(9,"Oranges")[4:9]
       ,cex=.5*(1:5), main="Trend of yield",key.space="right")
spplot(dataset, "detrended", col.regions=brewer.pal(9,"Oranges")[4:9]
       ,cex=.5*(1:5), main="Detrended yield",key.space="right")
```

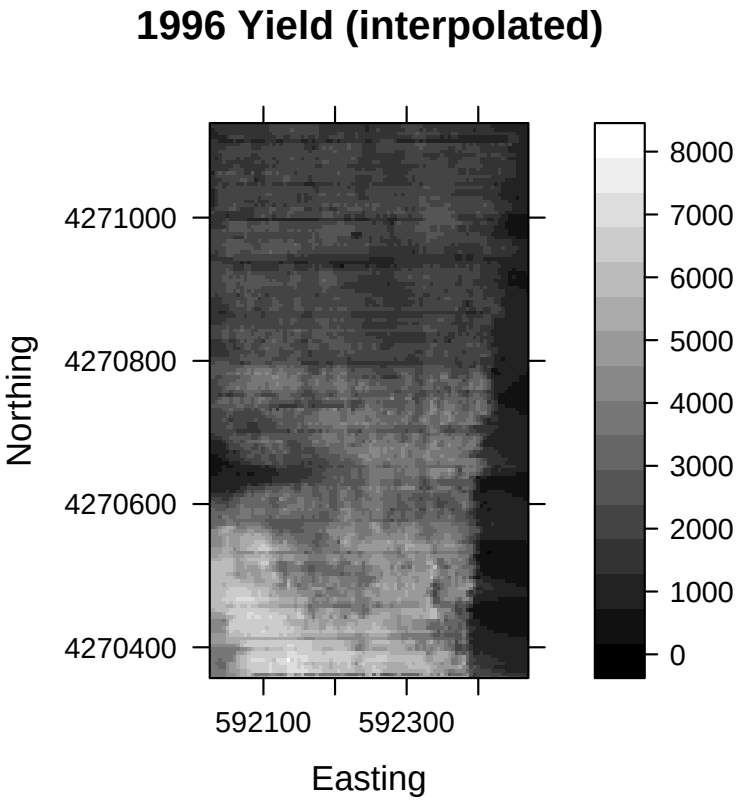


Figure E.7: Map of the yield (low resolution).

5. Then, we can compute semi-variograms for the detrended yields, and interpret it. Here we compute the semi-variograms in four different directions, to check for isotropy. What is your conclusion?

```
summary(dataset@coords)
library(gstat)
plot(variogram(detrended ~ 1, dataset, alpha = c(0, 45, 90, 135)),
     main="Variograms for detrended yield")
```

E.4.2 Model selection to explain the yield

We do not consider `SoilTOC` for inclusion in the model, as it is quite correlated to `SoilTN`. Moreover, the soil texture components sum to 100 and `Sand` has a stong negative association

with `Clay`. Hence we do not consider `Sand` for inclusion in the model. Scatterplots show that the endogenous variables (`CropDens`, `LeafN` and `FLN`) all share some degree of association with each exogenous variable. Therefore we do not consider them for inclusion in the model. Finally, We consider the following explanatory variables for inclusion in the model: `Silt`, `Clay`, `SoilpH`, `SoilTN`, `SoilP`, `SoilK`, `Weeds` and `Disease`.

1. We can represent the possible scatterplots or boxplots between the Yield and the explanatory variables, to get an idea of the relationships. Give an interpretation for all the following plots.

```
par(mfrow=c(2,2))
plot(Yield ~ Silt, dataset)
plot(Yield ~ Clay, dataset)
plot(Yield ~ SoilpH, dataset)
plot(Yield ~ SoilTN, dataset)
par(mfrow=c(2,2))
plot(Yield ~ SoilP, dataset)
plot(Yield ~ SoilK, dataset)
plot(Yield ~ Weeds, dataset)
plot(Yield ~ Disease, dataset)
```

2. Several arguments are in favor of the inclusion of interactions in the model. First, the counterintuitive observed associations with yield as discussed before. The preceding scatterplot show relationships that indicate an interaction between some of the explanatory variables associated with yield. Indeed, some yield values trend in one direction with an explanatory variable, and some are either not associated or trend in another direction. This may indicate that yield has a different relationship with these variables in different parts of the field. (In fact we can make other plots and see that there are different linear relationships between yield and the explanatory variables in the north and in the south of the field. If we do not want to include interactions, we should do two analysis: one for the north of the field, and one for the south.)
   A lot of interactions can be considered, but we restrict the analysis to those which have a biophysical or an empirical sense. We think that mineral nutrients and pH can interact with soil texture, hence we consider the interaction terms of `Clay` with `SoilpH`, `SoilTN`, `SoilP` and `SoilK`. We also consider another interaction justifiable on

a biochemical basis, the interaction between SoilP and SoilpH.
The full model is coded as follows:

```
mod.full <- lm(Yield ~ Silt + Clay + SoilpH + SoilTN + SoilP + SoilK
               + Weeds + Disease + Clay*SoilpH + Clay*SoilTN + Clay*SoilP
               + Clay* SoilK + SoilpH*SoilP, data=dataset)
```

Using a selection procedure, select a model to explain the yield.

E.4.3 Model Checking

Using the preceding selected model, the error terms are assumed to be independent, to follow the Gaussian distribution and to be homoscedastic. In particular, no spatial correlation of the error term is assumed. To validate these assumptions, we have to perform several tests and figures.

1. Write the equation of the model kept, and the associated assumptions on the residuals.
2. The first step to check the assumptions is to plot some figures dedicated to model diagnosis. Interpret these figures.
3. To check the homoscedasticity of the residuals, we can plot the residuals against the fitted values and against every possible explanatory variable.
4. We now want to check the spatial independence of the residuals. The first step is to represent the residuals on a map. Based on this map, what do you think of the spatial distribution of the residuals?
5. We can represent the variogram of the residuals. Interpret its shape.
6. We can also represent the Moran correlogram of the residuals. You first need to define a list of neighbors for each location. For instance, you can define that the neighbors of a location are its 4-nearest neighbors. Interpret the shape of the Moran correlogram.
7. Finally after all these plots, we can perform a statistical test to test if the residuals are spatially correlated or not. Give the name of this test, the associated hypotheses, and perform it. What is your conclusion?

E.4.4 Regression models for spatially autocorrelated data

After fitting a linear model to explain the yield, we find that the assumptions on the residuals were not verified. In particular, the residuals were found to be spatially correlated (we checked that we were not in presence of heteroscedasticity). Therefore, we cannot rely on classical tests on the coefficients of this model, hence we cannot interpret this model. Indeed, the type I error rate could be increased, or the coefficient' estimates could be biased.

We need to adapt our methodology and to use models taking into account spatial effects by using specific autocorrelation structures. We will use regression models specifically designed for spatial data (spatial lag and spatial error models).

E.4.4.1 Fitting spatial lag and spatial error models

1. For these models, we need to define a spatial weights matrix: we define a list of neighbors for each location (for instance the 4-nearest neighbors), then we define a spatial weights matrix (for instance a row-standardised spatial weights matrix).

```
library(spdep)
nlist <- knn2nb(knearneigh(dataset,k=4))
W <- nb2listw(nlist,style="W")
```

2. Give the equation of the spatial lag model, the associated assumptions, and interpret it.

3. Fit the spatial lag model. We first need to specify the formula of the model. Concerning the explanatory variables, we keep those which were selected in the classical linear case.

```
library(spatialreg)
myformula <- as.formula("Yield ~ Clay + SoilP + Weeds + Clay*SoilP")
mod.lag <- lagsarlm(myformula, data=dataset, listw=W)
summary(mod.lag)
```

4. Give the equation of the spatial error model, the associated assumptions, and interpret it.

5. Fit the spatial error model, in the same way you fitted the spatial lag model.

```
mod.err <- errorsarlm(myformula, data=dataset, listw=W)
summary(mod.err)
```

E.4.4.2 Comparison between spatial lag and spatial error models and model selection

1. Using likelihood ratio tests, choose between the spatial lag model and the spatial error model. For each test performed, specify the hypotheses.

```
LR.sarlm(model.lm,mod.lag)
LR.sarlm(model.lm,mod.err)
```

2. Using AIC criteria and pvalues, choose between the spatial lag model and the spatial error model.

```
AIC(mod.lag,mod.err)
```

The p-values indicate that we prefer the spatial lag model (this model is not rejected). The same conclusion is obtained if we use the AIC criteria.
We can fit a SAC model and compare it to the spatial lag model.

```
mod.sac <- sacsarlm(myformula, data=dataset, listw=W)
LR.sarlm(mod.sac,mod.lag)
AIC(mod.lag,mod.sac)
```

We prefer to keep the spatial lag model instead of the spatial SAC model. This means that the yield values are directly associated with each other, as opposed to being associated with unmeasured, spatially autocorrelated processes that are loaded into the error. It seems counterintuitive, since it is unlikely that wheat plants would influence each other at a distance of 61 meters. Each of the explanatory variables is likely to be autocorrelated. The values of `SoilP` and `Clay` are highly influenced by soil forming processes, and `Weed` is likely to be influenced by the weed seed bank, which likely display interactive autocorrelation. `Yield` response to these autocorrelated explanatory

variables, as well as to autocorrelated variables not included in the model, may cause it to display a level of positive autocorrelation sufficient for the lag model to provide the best fit to the data. This is especially true if the uncorrelated errors ϵ tend to be larger in magnitude than the correlated errors η.

3. Try to remove explanatory variables and interactions between them to improve the fitting of your model, and to include other variables that were not already included.
4. Have a look at the residuals of the final model chosen.
5. Predictions using this spatial lag model can be done. Make predictions for the data on which the model was fitted.

```
pred <- as.data.frame(predict.sarlm(mod.lag))
head(pred)
dataset$pred <- pred[,1]
spplot(dataset, "pred", col.regions=brewer.pal(9,"Oranges")[4:9],
       cex=.5*(1:5), main="Fitted values of mod.lag")
```

6. But It is also possible to make predictions for new data. For instance, we can make predictions for yield if the soil phosphorous content is increased by one unit (using fertilizer).

```
dataset2 <- dataset
dataset2@data$SoilP <- dataset@data$SoilP + 1
newpred <- as.data.frame(predict.sarlm(mod.lag, newdata=dataset2@data,
                                       listw = W, pred.type="TS"))
head(newpred)
dataset@data$prev <- newpred$fit
spplot(dataset, "prev", col.regions=brewer.pal(9,"Oranges")[4:9],
       cex=.5*(1:5), main="Prevision of yield when SoliP is increased by 1")
```

E.4.4.3 Extended linear model

We now want to use an extended linear model, which is a generalization of the spatial lag and spatial error models.

Choosing the form of the variance-covariance matrix of the error term is equivalent to choose a model for the semi-variogram.

1. You can have a look to the form of the empirical semi-variogram for the residuals.
2. As we do not feel confident to choose a model from the form of the semi-variogram, we prefer to choose the model of the semi-variogram using the AIC criteria.

```
Error in glsEstimate(object, control = control):  computed "gls" fit is singular
rank 1

Error in `coef<-.corSpatial`(`*tmp*`, value = value[parMap[, i]]):  NA/NaN/Inf
in foreign function call (arg 1)
```

3. Once you have chosen a model, you can compare it to the classical linear model, using the `anova` fonction.
4. You then need to check that the chosen model solve the problem of spatial dependency, by looking at the semi-variogram of the studentized residuals, and by looking at these Studentized residuals on a map.
5. Make predictions of yield.

```
pred <- predict(modSpher, newdata=dataset)
head(pred)
```

6. If this is relevant, make some predictions in case the values of some explanatory variables are increased or decreased (imagine relevant scenarios).

```
dataset2 <- dataset
dataset2@data$SoilP <- dataset@data$SoilP + 1
newpred <- predict(modSpher, newdata=dataset2)
head(newpred)
dataset@data$prev2 <- as.numeric(newpred)
spplot(dataset, "prev2", col.regions=brewer.pal(9,"Oranges")[4:9],
       cex=.5*(1:5), main="Predicted yield when soilP is increased by 1")
```

Appendix F

Bibliography

[1] N.A.C. Cressie. *Statistics for spatial data.* Wiley series in probability and mathematical statistics: Applied probability and statistics. J. Wiley, 1993.

[2] R.E. Plant. *Spatial Data Analysis in Ecology and Agriculture Using R.* Taylor & Francis, 2012.

[3] Robin Lovelace, Jakub Nowosad, and Jannes Muenchow. *Geocomputation with R.* Chapman and Hall CRC, 2019.

[4] Edzer J. Pebesma and Roger S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9–13, November 2005.

[5] Robert J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2017. R package version 2.6-7.

[6] Luc Anselin. Spatial effects in econometric practice in environmental and resource economics. *American Journal of Agricultural Economics*, 83(3):705–710, 2001.

[7] Roger S. Bivand, Edzer Pebesma, and Virgilio Gomez-Rubio. *Applied spatial data analysis with R, Second edition.* Springer, NY, 2013.

[8] J. Lesage and R. Pace. *Introduction to Spatial Econometrics.* Chapman and Hall CRC, 2009.

[9] R. Webster and M.A. Oliver. *Statistical methods in soil and land resource survey.* Spatial information systems. Oxford University Press, 1990.

[10] M. Arnaud and X. Emery. *Estimation et interpolation spatiale: méthodes déterministes et méthodes géostatistiques.* Hermes, 2000.