
Applied geostatistics

Exercise 4: Predicting from point samples (Part 1)

D G Rossiter

June 27, 2014

Contents

1	Introduction	3
2	Trend surfaces	3
3	* Design-based prediction	6
3.1	Answers	14
4	Ordinary kriging	15
4.1	Variogram modelling	15
4.2	Prediction at single points	16
4.3	Confidence limits	18
4.4	Prediction on a regular grid	20
4.5	* Predicting at a sample point	27
4.6	Answers	29
5	Saving the workspace	30
6	* Ordinary kriging with anisotropy	31
6.1	Computing the anisotropic variogram	31
6.2	Geometric anisotropy	32
6.3	Zonal anisotropy	40
6.4	Answers	45
7	* Insight into the OK system	45
7.1	The OK system at a sample point	52
7.2	Answers	55

Version 3.1. Copyright © 2006-2010 ITC, 2011-2013 University of Twente, 2014 D G Rossiter All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited. To adapt or translate please contact the author (d.g.rossiter@utwente.nl).

8 Self-test	55
9 Quitting R	57
References	58
Index of R concepts	59

不怕慢，只怕站
“Do not fear slowness, only fear stopping”
– Chinese proverb

1 Introduction

In this exercise you will learn several ways to **predict** at unsampled points, using data values from a set of sample points.

After completing this exercise you should be able to:

1. Predict attributes over a region with a trend surface;
2. (optionally) Predict attributes over a region using strata;
3. Construct a prediction grid over a region;
4. Predict attributes over a region using Ordinary Kriging (OK);
5. (optionally) Predict attributes over a region using OK with an anisotropic variogram model;
6. (optionally) Gain insight into the OK system.

This exercise includes several sections (marked with a *) with topics that are **optional** and not covered in the self-test. You should at least read the first paragraph of these sections to know their topic, since you may want to work through them later.

Task 1 : If R is not already running, start it. If you haven't already done so, load the **gstat** and **sp** libraries, as shown in the previous exercises. •

2 Trend surfaces

In a previous exercise (Exercise 3 §2) we computed a regional trend from a set of point observations. In a different exercise (Exercise 2 §6) we created a raster of the study area in which the observations are found, at a given resolution, and computed the predictions at all the grid cells of the raster.

The trend surface computed by the **krige** method of the **gstat** package is the same as that computed directly by the **lm** linear modelling method with coördinates as predictors; they both use least-squares (by default **unweighted**) to fit the surface.

There are several differences:

- **krige** does not report the trend surface coefficients;
 - However, the underlying **predict.gstat** method with the optional **BLUE=T** argument can be used for this.
- **lm** only predicts at the observation points (with the **fitted** access method; **krige** at any number of points.

- However, `predict.lm` can be used to predict at any set of points, using the `newdata` argument.

If you wish to review this, we repeat here the procedures for:

1. computing the trend surface equation with the `lm` function;
2. computing an interpolation grid;
3. computing a trend surface with the `krige` method; and
4. visualizing the result.

```
> load("tcp.RData")
> ls()

[1] "tcp"

> str(tcp)

'data.frame':      147 obs. of  4 variables:
 $ UTM_E : num  702638 701659 703488 703421 703358 ...
 $ UTM_N : num  326959 326772 322133 322508 322846 ...
 $ clay35: num   78 80 66 61 53 57 70 72 70 62 ...
 $ pH35  : num   4.8 4.4 4.2 4.54 4.4 ...
```

The trend surface equation, as computed by `lm`:

```
> summary(lm(clay35 ~ UTM_N + UTM_E, data = tcp))

Call:
lm(formula = clay35 ~ UTM_N + UTM_E, data = tcp)

Residuals:
    Min       1Q   Median       3Q      Max
-31.601  -5.106  -0.363   3.607  20.467

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.50e+02   5.19e+01  -4.83  3.5e-06 ***
UTM_N        -4.50e-04   9.24e-05  -4.88  2.8e-06 ***
UTM_E         6.51e-04   5.97e-05  10.91 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.11 on 144 degrees of freedom
Multiple R-squared:  0.506,    Adjusted R-squared:  0.499
F-statistic: 73.7 on 2 and 144 DF,  p-value: <2e-16
```

The interpolation grid, as a `SpatialPoints` object:

```
> coordinates(tcp) <- ~UTM_E + UTM_N
> gr <- expand.grid(UTM_E = seq(659000, 704000, by = 500),
+   UTM_N = seq(310000, 343000, by = 500), KEEP.OUT.ATTRS = F)
> coordinates(gr) <- ~UTM_E + UTM_N
> str(gr)
```

```

Formal class 'SpatialPoints' [package "sp"] with 3 slots
..@ coords      : num [1:6097, 1:2] 659000 659500 660000 660500 661000 ...
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : NULL
.. .. ..$ : chr [1:2] "UTM_E" "UTM_N"
..@ bbox        : num [1:2, 1:2] 659000 310000 704000 343000
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : chr [1:2] "UTM_E" "UTM_N"
.. .. ..$ : chr [1:2] "min" "max"
..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
.. .. ..@ projargs: chr NA

```

The interpolation from the trend surface, using `krige`:

```

> ts1 <- krige(clay35 ~ UTM_E + UTM_N, loc = tcp, newdata = gr,
+             model = NULL)

```

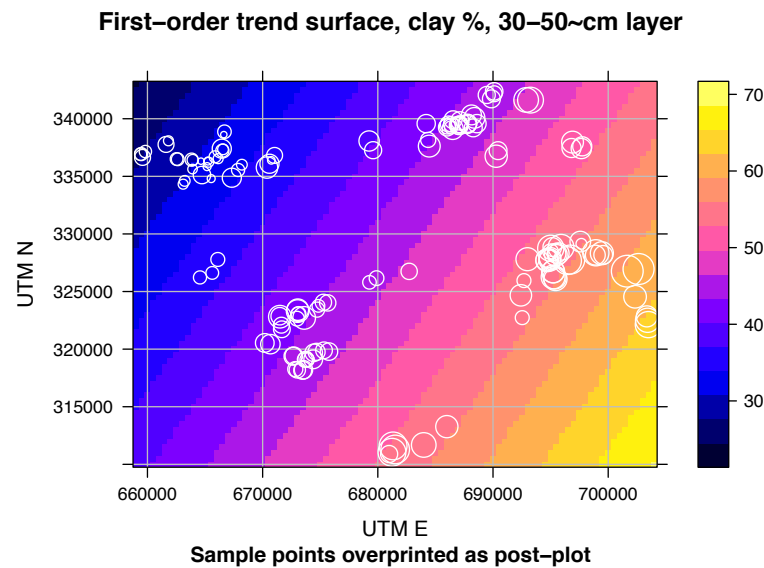
[ordinary or weighted least squares prediction]

Now we display the trend surface as a raster map. The `levelplot` function is part of the `lattice` graphics package; to make sure this is loaded we use the `require` command:

```

> require(lattice)
> print(levelplot(var1.pred ~ UTM_E + UTM_N, as.data.frame(ts1),
+               col.regions=bpy.colors(64), asp="iso",
+               main="First-order trend surface, clay %, 30-50~cm layer",
+               sub="Sample points overprinted as post-plot",
+               xlab="UTM E", ylab="UTM N",
+               panel=function(x,y,z, ...) {
+                 panel.levelplot(x, y, z, ...)
+                 panel.grid(h=-1,v=-1, col="gray", lty=1)
+                 panel.points(coordinates(tcp),
+                               cex=3*tcp$clay35/max(tcp$clay35),
+                               pch=1, col="white")
+               })

```



This should have refreshed your memory.

Task 2 : Clean up the local variables created in this section. •

```
> rm(tcp, gr, ts1)
```

3 * Design-based prediction

This is an **optional** section that shows a non-geostatistical method of spatial prediction that should be familiar from basic GIS. It is still useful when there are not enough points to establish a spatial structure. It is also applicable if there is no spatial structure, but there are difference between strata.

One method of “spatial” prediction is to divide the area into **strata** where the attribute to be predicted is supposed to be **homogeneous** within each stratum:

1. Same **expected value**;
2. Same **probability distribution** and its **parameters**, e.g. variance;
3. **No spatial dependence**.

This is essentially **reclassification** of a **classified** GIS layer; it should not be used if the structure of spatial dependence can be modelled. In case there are differences between strata *and* residual spatial structure, use Kriging with External Drift (KED), with the strata as the “drift” (see Exercise 5a, §6).

We illustrate this with the Jura dataset from the `gstat` package. This is the same dataset we used in the previous exercises, but here we use the built-in example dataset `jura`, which has both point observations and an interpolation grid.

Task 3 : Load the Jura example data set and examine its contents. •

Q1 : What objects are loaded with the Jura example dataset? What do they represent? *Jump to A1* •

Example datasets are loaded with the `data` method:

```
> data(jura)
> ls()

[1] "jura.grid"      "jura.pred"      "jura.val"
[4] "juragrid.dat"   "prediction.dat"  "transect.dat"
[7] "validation.dat"

> help(jura)
```

Notice that three dataframes were loaded: `jura.pred`, `jura.val`, and `jura.grid`.

Task 4 : Identify the attributes in the prediction points dataset that could be used for stratification. •

```
> str(jura.pred)

'data.frame':      259 obs. of  11 variables:
 $ Xloc   : num  2.39 2.54 2.81 4.31 4.38 ...
 $ Yloc   : num  3.08 1.97 3.35 1.93 1.08 ...
 $ Landuse: Factor w/ 4 levels "Forest","Pasture",...: 3 2 2 3 3 3 3 3 3 ...
 $ Rock   : Factor w/ 5 levels "Argovian","Kimmeridgian",...: 3 2 3 2 5 5 5 1 1 3
 $ Cd     : num  1.74 1.33 1.61 2.15 1.56 ...
 $ Co     : num  9.32 10 10.6 11.92 16.32 ...
 $ Cr     : num  38.3 40.2 47 43.5 38.5 ...
 $ Cu     : num  25.72 24.76 8.88 22.7 34.32 ...
 $ Ni     : num  21.3 29.7 21.4 29.7 26.2 ...
 $ Pb     : num  77.4 77.9 30.8 56.4 66.4 ...
 $ Zn     : num  92.6 73.6 64.8 90 88.4 ...
```

Q2 : Which attributes could be used for stratification? *Jump to A2* •

Task 5 : Determine which of these attributes are also in the interpolation grid dataset. •

```
> str(jura.grid)
```

```
'data.frame':      5957 obs. of  4 variables:
 $ Xloc   : num  0.3 0.35 0.35 0.4 0.4 0.4 0.4 0.4 0.4 0.4 ...
 $ Yloc   : num  1.7 1.7 1.75 1.7 1.75 1.8 1.85 1.9 2.1 2.15 ...
 $ Landuse: Factor w/ 4 levels "Forest","Pasture",...: 2 2 2 2 1 1 1 1 3 3 ...
 $ Rock    : Factor w/ 5 levels "Argovian","Kimmeridgian",...: 3 3 3 2 2 2 3 3 1 1
```

Q3 : Which of the attributes that could be used for stratification are also known over the interpolation grid? Jump to A3 •

Task 6 : Promote the prediction points and interpolation grid to spatial objects. •

Objects are promoted to spatial objects with the `coordinates` method, specifying which fields represent the coordinates:

```
> class(jura.pred)

[1] "data.frame"

> coordinates(jura.pred) <- ~Xloc + Yloc
> class(jura.pred)

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

We also promote the grid to a spatial object; by default a set of points is of class `SpatialPointsDataFrame`, but since this is on a grid we'd like it to be treated as `SpatialPixelsDataFrame`; one way to accomplish this is by using the `gridded` method to specify that the points are on a grid.

```
> class(jura.grid)

[1] "data.frame"

> coordinates(jura.grid) <- ~Xloc + Yloc
> class(jura.grid)

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

> gridded(jura.grid) <- TRUE
> class(jura.grid)

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

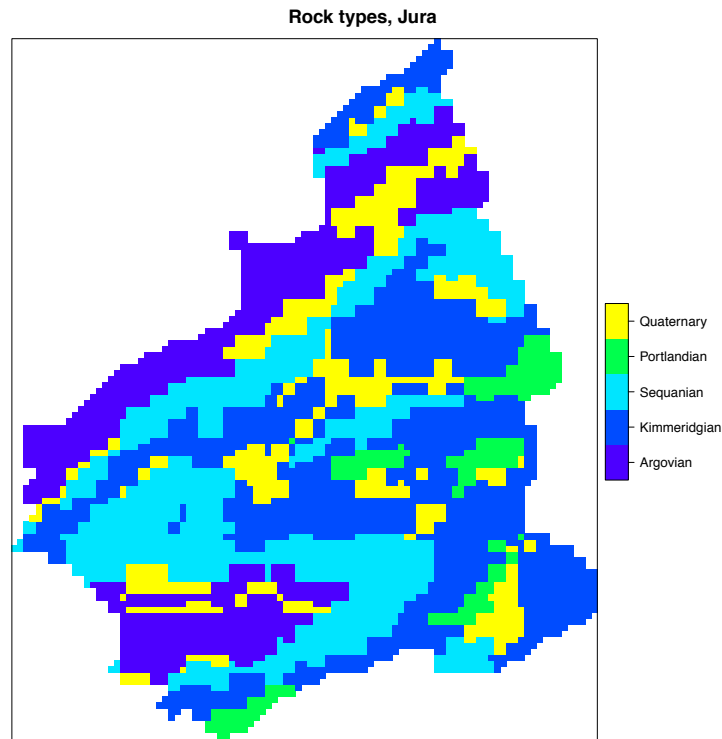
We will use **rock type** as a stratifying attribute for metal content. This is logical if the heavy metals in soils in this area are mainly from parent rock (and not from human activity).

Task 7 : Display a map of the rock type strata. •

One way to do this is with the `spplot` method. Since the `jura.grid` object is now a `SpatialGridDataFrame`, the `spplot` method automatically uses a square filled printing character (i.e., `pch=15`) to simulate a raster display. We also place the legend to the right with the `key.space` optional argument.

Note: We know from the summary above that the field `Rock` has five classes, so we could just write `topo.colors(5)` to specify a five-colour legend. It is more elegant to compute the number of classes automatically with the `nlevels` method, which returns the number of “levels” or classes in a classified field; so we write `topo.colors(nlevels(jura.grid$Rock))`.

```
> print(spplot(jura.grid, zcol="Rock",
+             col.regions=topo.colors(nlevels(jura.grid$Rock)),
+             key.space="right",
+             main="Rock types, Jura"))
```



Note: The `spplot` method is a front-end for various methods in the `lattice` graphics package, such as `xyplot` and `levelplot`.

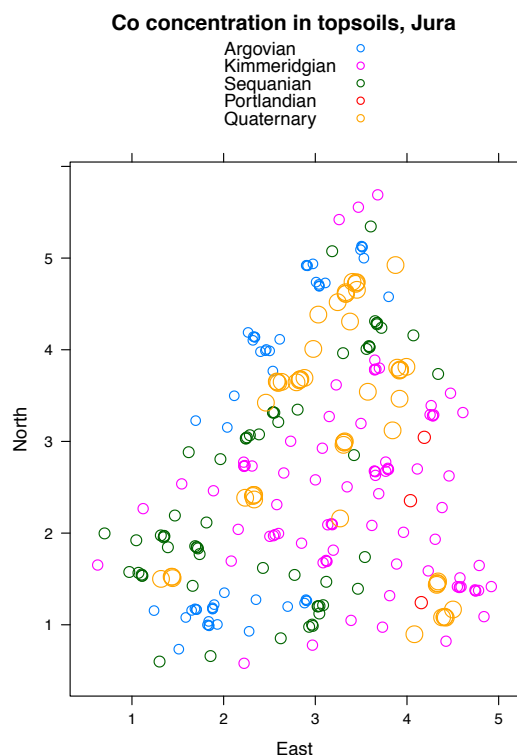
Task 8 : Display a postplot of the Co concentrations at the observation points colour-coded by their rock types. •

We use the `xyplot` method, specifying the axes as a formula `Yloc ~ Xloc` (since the vertical axis should have the North coordinate), the character size `cex` from the cobalt (Co) value, and the colour `col` from the rock type class.

Note that we must convert the factor `Rock` to a number with the `as.numeric` method.

Note also the use of the `groups` graphics argument to automatically colour the different rock types, and the `auto.key=T` graphics argument to show this in a legend.

```
> print(xyplot(Yloc ~ Xloc, as.data.frame(jura.pred),
+             groups=jura.pred$Rock,
+             cex=2*jura.pred$Co/max(jura.pred$Co), pch=1,
+             asp="iso", xlab="East", ylab="North",
+             main="Co concentration in topsoils, Jura",
+             auto.key=T)
+ )
```



Q4 : Does the *Co* content appear to be related to the rock types? [Jump to A4](#) •

We predict in the class based on the **analysis of variance** of a linear model, which gives the **mean** and **pooled standard deviation**.

Task 9 : Compute the one-way Analysis of Variance (ANOVA) of *Co* concentration as modelled by rock type; display the summary. •

We use the `lm` method and then summarize with `summary.lm`:

```
> m.co.rt <- lm(Co ~ Rock, data = jura.pred)
> summary(m.co.rt)
```

```

Call:
lm(formula = Co ~ Rock, data = jura.pred)

Residuals:
    Min       1Q   Median       3Q      Max
-9.497 -1.565  0.086  1.878  8.124

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      5.394      0.404   13.35 < 2e-16 ***
RockKimmeridgian  5.655      0.515   10.99 < 2e-16 ***
RockSequanian     4.581      0.548    8.36 4.2e-15 ***
RockPortlandian   3.979      1.745    2.28  0.023 *
RockQuaternary    4.202      0.566    7.42 1.7e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.94 on 254 degrees of freedom
Multiple R-squared:  0.334,    Adjusted R-squared:  0.324
F-statistic: 31.9 on 4 and 254 DF,  p-value: <2e-16

```

Q5 : *How much of the variation in Co content is explained by the model?* Jump to A5 •

Q6 : *What are the expected values for each rock type?* Jump to A6 •

These are the means for each class; we can find these with the `by` method:

```

> by(jura.pred$Co, jura.pred$Rock, mean)

jura.pred$Rock: Argovian
[1] 5.3939
-----
jura.pred$Rock: Kimmeridgian
[1] 11.049
-----
jura.pred$Rock: Sequanian
[1] 9.9752
-----
jura.pred$Rock: Portlandian
[1] 9.3733
-----
jura.pred$Rock: Quaternary
[1] 9.5959

```

Task 10 : Display a map of the predicted values of the Co content, with a colour ramp. •

The map of predicted values is just the map of strata **reclassified** with the predicted value for each class; we can get these with the `predict` generic method on the fitted model:

Note: We choose to specify the `data.frame` component (i.e., slot `@data`) of the target grid `jura.grid`, which is of class `SpatialPixelsDataFrame` as the `newdata` argument to `predict.lm`. This is in general not necessary since many methods extract the data frame component; however this syntax makes it explicit that the prediction is non-spatial, it only uses the attributes of the spatial object `jura.grid`.

```
> head(predict(m.co.rt, newdata = jura.grid@data))

      1      2      3      4      5      6
9.9752 9.9752 9.9752 11.0489 11.0489 11.0489
```

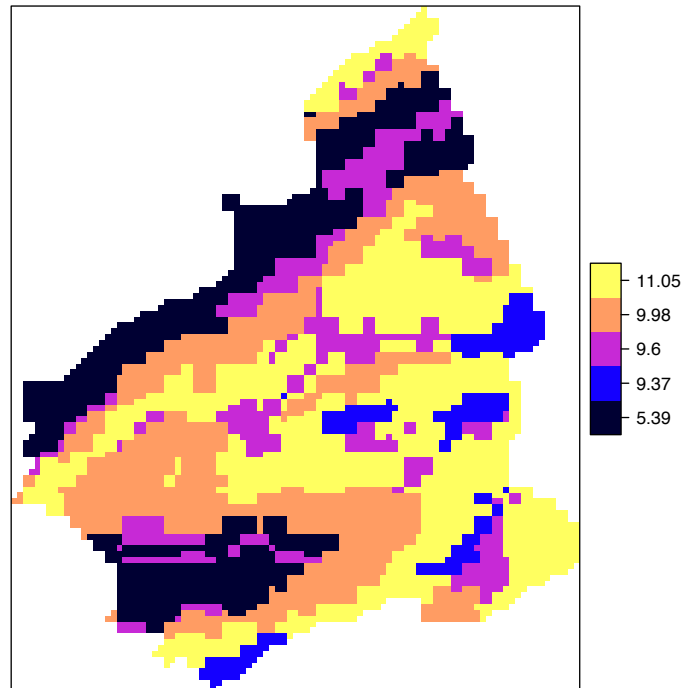
The difficulty here is that the ordered list of classes is not in sorted order of the predictions. For example, the second-listed class (`Kimmeridgian`) has the highest predicted concentration. So, we use the `as.ordered` method on the predicted values to sort the values in the proper order for a colour ramp display.

```
> co.reclass <-
+   data.frame(Rock=jura.grid$Rock,
+             Co=as.ordered(round(predict(m.co.rt,
+             newdata=jura.grid@data),2)))
> coordinates(co.reclass) <- coordinates(jura.grid)
> gridded(co.reclass) <- TRUE
```

Now we can show the prediction:

```
> print(spplot(co.reclass, zcol="Co",
+             col.regions=bpy.colors(nlevels(jura.grid$Rock)),
+             main="Predicted Co concentration in topsoils, Jura"))
```

Predicted Co concentration in topsoils, Jura



Q7 : Explain the abrupt spatial changes in predicted Co concentrations over the map. *Jump to A7 •*

We also would like to know the **confidence interval** for this prediction, i.e. the range within which we expect any new observation to fall, with a given probability of being wrong.

Task 11 : Compute the standard errors of prediction. •

Because of the uneven number of samples in the strata, we compute the standard deviation of the prediction for each class separately, directly from the samples, using the `sd` method.

We could use the `by` method, as we did for the means, above, but instead we'll use a more general method, splitting the data frame into separate lists for each rock type. To get these lists, we use the `split` method, specifying the rock type as the factor. Then we use the `sapply` method to **apply** a function, in this case `sd`, to each element in a list separately:

```
> co.split <- split(as.data.frame(jura.pred)$Co, jura.pred$Rock)
> sapply(co.split, sd)
```

Argovian	Kimmeridgian	Sequanian	Portlandian	Quaternary
2.0673	2.8064	2.1987	2.4509	4.3067

Q8 : What are the standard errors of prediction for each rock type? *Jump*

to A8 •

Once the standard errors of prediction are known, they can be used to compute confidence intervals, i.e., intervals in which we are fairly sure (to some user-specified probability) that the true value lies.

The confidence interval can be computed easily with the `t.test` method, again applied to all classes with `sapply`. The `t.test` method returns a large structure; we are only interested in the confidence interval:

```
> data.frame(sapply(co.split, t.test)["conf.int", ],
+   row.names = c("lower", "upper"))
```

	Argovian	Kimmeridgian	Sequanian	Portlandian	Quaternary
lower	4.8241	10.444	9.4215	3.2849	8.4316
upper	5.9637	11.654	10.5290	15.4617	10.7601

```
> data.frame(sapply(co.split, t.test, conf.level = 0.99)["conf.int",
+   ], row.names = c("lower", "upper"))
```

	Argovian	Kimmeridgian	Sequanian	Portlandian	Quaternary
lower	4.6346	10.247	9.2391	-4.6706	8.0454
upper	6.1531	11.851	10.7114	23.4173	11.1464

The first example is with the default 95% confidence level; in the second example we specify a 99% confidence level with the optional `conf.level` argument to the `t.test` command. Note that the original data must be approximately normally distributed within each class for this to be valid.

Note: We use of the `data.frame` function to restructure the list of two-element vectors into a data frame with named rows (using `row.names`) for a nicer display.

We will see in Exercise 5a how to combine the stratification with local prediction.

Task 12 : Clean up the local variables created in this section. •

```
> rm(co.reclass, co.split)
```

3.1 Answers

A1 : Seven R objects are loaded: `jura.grid`, `jura.pred`, `jura.val`, `juragrid.dat`, `prediction.dat`, `transect.dat` `validation.dat`. The help text shows that there are two versions of everything except a `transect`. The names like `jura.*` are preferred. *Return to Q1*

•

A2 : The classified attributes are `Landuse` (land use) and `Rock` (rock type). *Return to Q2* •

A3 : Both **Landuse** (land use) and **Rock** (rock type) are also in the interpolation grid and so may be used for design-based prediction. [Return to Q3](#) •

A4 : Somewhat; the first-listed rock type, Argovian, has the smallest symbols. [Return to Q4](#) •

A5 : 32.4% (see the adjusted R^2); so just under one-third of the variation in Co concentration is explained by the rock type. [Return to Q5](#) •

A6 : The prediction for each rock type is:

Argovian	Kimmeridgian	Sequanian	Portlandian	Quaternary
5.3939	11.0489	9.9752	9.3733	9.5959

[Return to Q6](#) •

A7 : The abrupt changes are at rock type boundaries. Within each rock type all locations have the same prediction. [Return to Q7](#) •

A8 : The standard error of prediction for each rock type is:

Argovian	Kimmeridgian	Sequanian	Portlandian	Quaternary
2.0673	2.8064	2.1987	2.4509	4.3067

[Return to Q8](#) •

4 Ordinary kriging

For this section of the exercise we continue with the Jura soil samples introduced in Exercise 2, §2. We saved this as an R object in Exercise 2, §3.2.

4.1 Variogram modelling

The tasks in this subsection are repeated from Exercise 3, §3, where you fit a variogram model to the Co content in the calibration dataset; if you already have these in your workspace there is no need to repeat them.

Task 13 : If the `jura.all` spatial object is not already in the workspace, load it from the saved image. •

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

Task 14 : Divide the Jura dataset into a **calibration** set `jura.cal`, made up of the first 259 observations from the full set, and a **evaluation** (also called **validation**) set `jura.val`, made up of the remaining 100 observations from the full set. •

This is just row selection from a matrix using the `[]` operator, which also works with the `@data` slot of an `sp` object:

```
> jura.cal <- jura.all[1:259, ]
> jura.val <- jura.all[260:359, ]
```

Task 15 : Compute the experimental variogram of the `Co` values in the **calibration** dataset, with a cutoff of 1.6 km, and model it. •

```
> v <- variogram(Co ~ 1, loc = jura.cal, cutoff = 1.6)
> (vmf <- fit.variogram(v, vgm(12.5, "Pen", 1.2, 1.5)))
```

```
      model    psill  range
1   Nug  1.3712 0.0000
2   Pen 12.9322 1.5239
```

Now we have a model of local spatial dependence, and can use it for prediction.

4.2 Prediction at single points

Kriging can be used to predict at specific points; it does not have to be used to produce a regular grid (for that, see §4.4). We illustrate this with the 100 observations which we held out as a **evaluation set**.

Note: The mathematics behind this are explored in optional §7, below.

Task 16 : Use the fitted variogram model to predict at the 100 evaluation points by Ordinary Kriging (OK). •

The `krige` method of the `gstat` package has the following required arguments:

1. a **formula** specifying the **model of spatial dependence** (the `formula` argument; if this comes first it does not need to be named);
2. a **spatial object** of **observation points** where the attribute value is known (the `loc` argument; if this comes second it does not need to be named);
3. a **variogram model** (the `model` argument)

and predicts at the points of a **second spatial object** where the attribute values are to be **predicted** (the `newdata` argument):

So, to predict at the evaluation points (`newdata = jura.val`), using the spatial object of calibration samples (`loc = jura.cal`), only based on local spatial separation (`formula = Co ~ 1`), using the variogram model just computed (`model = vmf`):

```
> k.val <- krige(Co ~ 1, loc = jura.cal, newdata = jura.val,
+               model = vmf)
```



```
[using ordinary kriging]

> summary(k.val)

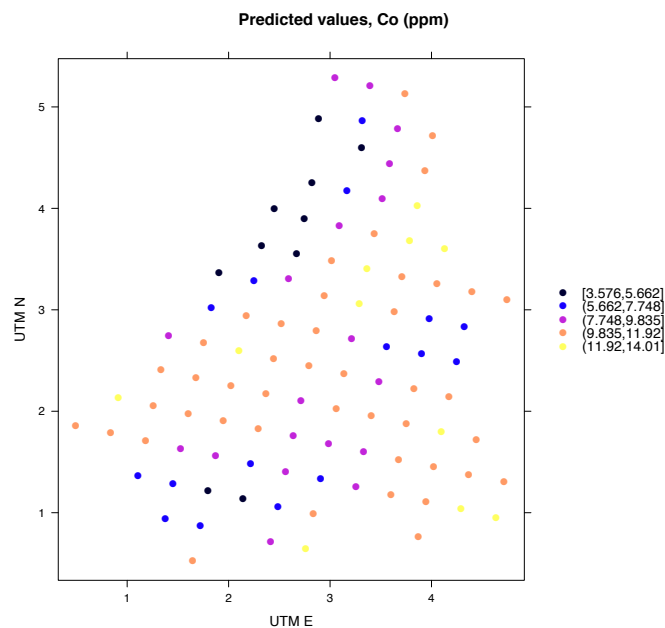
Object of class SpatialPointsDataFrame
Coordinates:
      min      max
X 0.491 4.745
Y 0.524 5.285
Is projected: NA
proj4string : [NA]
Number of points: 100
Data attributes:
      var1.pred      var1.var
Min.   : 3.58   Min.   :1.98
1st Qu.: 7.70   1st Qu.:4.35
Median :10.08   Median :4.84
Mean    : 9.46   Mean    :4.72
3rd Qu.:11.27   3rd Qu.:4.93
Max.    :14.01   Max.    :7.92
```

Q9 : What are the two data fields computed by *krige*? What are their units of measure? *Jump to A9* •

Task 17 : Plot the predictions and their variances. •

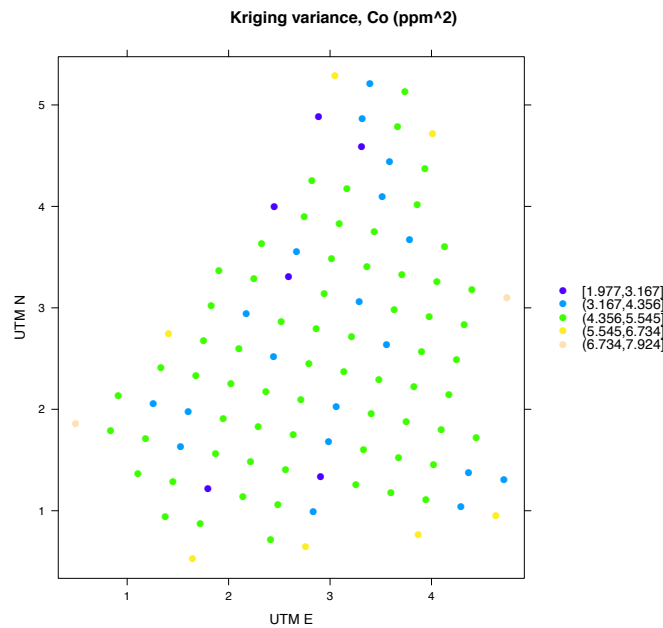
First the predictions:

```
> print(spplot(k.val, zcol = "var1.pred", col.regions = bpy.colors(64),
+   key.space = "right", main = "Predicted values, Co (ppm)",
+   xlab = "UTM E", ylab = "UTM N", scales = list(draw = T)))
```



Now the kriging variances. We use a different colour scheme (here, the colour ramp produced by the `topo.colors` method) so that we don't visually confuse predictions and their variances.

```
> print(spplot(k.val, zcol = "var1.var", col.regions = topo.colors(64),
+   key.space = "right", main = "Kriging variance, Co (ppm^2)",
+   xlab = "UTM E", ylab = "UTM N", scales = list(draw = T)))
```



4.3 Confidence limits

The advantage of knowing both the **predictions** and their **variances** is that we can determine the **probability** that the true value at the prediction location is in a given range, or above or below a given threshold. A common application is in **risk assessment**.

Task 18 : Determine the 95% confidence limits for the first evaluation point. •

Recall from probability theory that the two-sided interval which has probability $(1 - \alpha)$ of containing the true value z is:

$$(\hat{z} - \zeta_{\alpha/2} \cdot \sigma) \leq z \leq (\hat{z} + \zeta_{\alpha/2} \cdot \sigma)$$

where \hat{z} is the predicted value, $\zeta_{\alpha/2}$ is the standard normal variate at confidence level $\alpha/2$, and σ is the kriging prediction standard deviation.

Note: The total probability of Type I error α , say 0.05, is halved, say to 0.025 for each side of the interval, because this is a two-sided interval.

Q10 : Which field in the kriging prediction object corresponds to the estimated value \hat{z} and which to the kriging prediction standard deviation σ ? *Jump to A10 •*

To evaluate this formula, we need to find quantiles of the Normal distribution. As you might expect, R includes a method for this; in fact R includes methods to evaluate cumulative distribution functions (CDF), the probability density function (PDF) and the quantiles, and to draw random samples, from a large number of distributions; see see Chapter 8 of Venables et al. [1] for details.

We use the `qnorm` (“quantile of the **normal**-distribution) method, specifying the quantile (here, the 97.5%, because the test is two-tailed):

```
> (z.val <- qnorm(0.975))  
  
[1] 1.96
```

This is then multiplied by the standard deviation of the prediction to give the width of the confidence interval on each side of the prediction. For example, at the first prediction point:

```
> sqrt(k.val$var1.var[1]); (ciw <- z.val * sqrt(k.val$var1.var[1]))  
  
[1] 1.878  
  
[1] 3.6808
```

Finally we can compute the interval:

```
> k.val$var1.pred[1]; k.val$var1.pred[1] - ciw; k.val$var1.pred[1] + ciw  
  
[1] 5.0495  
  
[1] 1.3686  
  
[1] 8.7303
```

Let’s express this in the language of hypothesis testing:

Q11 : What is the **maximum** cobalt concentration at the first evaluation point, with only a probability of 2.5% chance that this value is exceeded? *Jump to A11 •*

In Exercise 6 we will compare these to the known values at these points, so do not delete object `k.val`.

Clean up from this section:

```
> rm(z.val, ciw)
```

4.4 Prediction on a regular grid

Kriging is typically used to make a **raster map** by predicting at all nodes of a **regular grid**.

We have the 50 m grid of the irregularly-shaped study area provided with the Jura dataset in the `gstat` package (§3); we will use this grid in subsequent exercises. However here we create and use a rectangular grid for two reasons: (1) to demonstrate how to create one; (2) to see the effects of extrapolation outside the study area.

Task 19 : Make a regular grid covering the Jura study area. •

First, we have to decide on a grid spacing. This is always a compromise between speed/storage space and spatial resolution. The spacing also depends on the objectives of the interpolation. There are some objective criteria that can help us decide on a suitable spacing, which we now examine.

The first criterion is the minimum spacing between sample points; it doesn't make sense to predict any more densely, because we have no information about spatial dependence at closer spacings. We can see from the post-plots we've made of the calibration dataset that there are some points quite close to each other.

To find out the actual minimum spacing, we can use the variogram cloud. Recall from Exercise 2 §7.1 that this semivariance vs. separation. So we can compute the cloud for close spacings and then find the minimum separation with the `min` method applied to the `dist` field of the variogram cloud object:

```
> vc <- variogram(Co ~ 1, loc = jura.cal, cloud = T,
+               cutoff = 0.1)
> str(vc)

Classes 'variogramCloud' and 'data.frame':      262 obs. of  6 variables:
 $ np      : num  8.59e+10 6.44e+10 1.16e+11 6.44e+10 1.50e+11 ...
 $ dist    : num  0.0064 0.00608 0.01526 0.09161 0.08737 ...
 $ gamma   : num  0.1152 0.0128 0.1152 0.8712 0.6728 ...
 $ dir.hor : num  0 0 0 0 0 0 0 0 0 0 ...
 $ dir.ver : num  0 0 0 0 0 0 0 0 0 0 ...
 $ id      : Factor w/ 1 level "var1": 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "direct")= 'data.frame':      1 obs. of  2 variables:
 ..$ id      : Factor w/ 1 level "var1": 1
 ..$ is.direct: logi TRUE
 - attr(*, ".BigInt")= num 4.29e+09
 - attr(*, "what")= chr "semivariance"

> min(vc$dist)

[1] 0.005

> sum(vc$dist < 0.1)

[1] 262
```

Q12 : *How many point-pairs are separated by 100 m or less? What is the closest spacing?* Jump to A12 •

Another criterion is the number of points in the grid; this is the product of the number of rows and columns at a given grid spacing.

Task 20 : Compute the number of grid cells that would be needed to cover the bounding box of the full Jura dataset at 100 m, 50 m, 20 m, and 5 m spacing. •

We can use the `bbox` spatial method to find the corners of the enclosing rectangle, compute its area, and the number of cells. Note the use of the factor 1000^2 to convert square kilometers (units of the bounding box) to square meters (units of the grid).

```
> bbox(jura.all)

      min  max
X 0.491 4.92
Y 0.524 5.69

> diff(bbox(jura.all)["X", ])

      max
4.429

> diff(bbox(jura.all)["Y", ])

      max
5.166

> area <- diff(bbox(jura.all)["X", ]) * diff(bbox(jura.all)["Y",
+      ])
> for (i in c(100, 50, 20, 5)) print(paste("spacing",
+      i, "m requires ", ceiling(area * (1000^2)/(i^2)),
+      "points"))

[1] "spacing 100 m requires  2289 points"
[1] "spacing 50 m requires  9153 points"
[1] "spacing 20 m requires  57201 points"
[1] "spacing 5 m requires  915209 points"

> rm(area)
```

The expression `area*(1000^2)` converts the area in km^2 to area in m^2 ; this is then divided by the number of m^2 of each cell.

Note the use of the `for` command to repeat the `print` method, once for each spacing. The output was formatted with the `paste` method, which makes one string of characters out of its arguments.

Q13 : *What is the area of the bounding box? How many grid cells would*

be required to cover it at 100 m, 50 m, 20 m and 5 m spacing? [Jump to A13](#) •

To keep the processing time and storage space down while still having a relatively attractive map, we'll use a 50 m spacing.

For the trend surface of §2 we made a rectangular grid. It is easy enough to repeat this procedure for this area.

Task 21 : Determine the corners of a bounding box for a 50 m prediction grid, and the number of cells in each dimension. •

We first expand the bounding box by 100 m and round to the nearest 100 m in the correct sense (lower for the minimum, higher for the maximum). This makes use of the `floor` and `ceiling` methods, which work on integers. Note how we first multiply the coördinates (which are in km) by 10 to make an integral number of 100 m, then use the `floor` and `ceiling` methods, and then divide the result by 10 to recover the km units. The number of cells is the dimension in km multiplied by number of cells per km, i.e. 20.

```
> (min.x <- floor((bbox(jura.all)["X", "min"] - 0.1) *
+      10)/10)

[1] 0.3

> (max.x <- ceiling((bbox(jura.all)["X", "max"] + 0.1) *
+      10)/10)

[1] 5.1

> (min.y <- floor((bbox(jura.all)["Y", "min"] - 0.1) *
+      10)/10)

[1] 0.4

> (max.y <- ceiling((bbox(jura.all)["Y", "max"] + 0.1) *
+      10)/10)

[1] 5.8

> (cells.x <- (max.x - min.x) * 20)

[1] 96

> (cells.y <- (max.y - min.y) * 20)

[1] 108

> cells.x * cells.y

[1] 10368
```

Q14 : How many points are in this prediction grid? What is its bounding box? [Jump to A14](#) •

Instead of the `expand.grid` method we used in §2, we will use the `SpatialGrid` method from the `sp` spatial package to create an object of class `SpatialGrid`. This class takes advantage of the known regular spacing between points, as opposed to the `SpatialPoints` class which can include any set of points, no matter what their spacing.

The first argument to the `SpatialGrid` method is a grid, which is created by the `GridTopology` method. This has three arguments:

- `cellcentre.offset`: a vector with the smallest coordinate of the **cell centre** for each dimension; in this case `min.x` and `min.y` plus half a cell size;
- `cellsize`: a vector with the cell size in each dimension; in this case both are 0.05 km;
- `cells.dim`: a vector with (integer) number of cells in each dimension; we calculated these from the expanded bounding box as `cells.x` and `cells.y`.

```
> jura.raster <- SpatialGrid(GridTopology(c(min.x +
+   0.025, min.y + 0.025), c(0.05, 0.05), c(cells.x,
+   cells.y)))
> str(jura.raster)
```

```
Formal class 'SpatialGrid' [package "sp"] with 3 slots
..@ grid      :Formal class 'GridTopology' [package "sp"] with 3 slots
.. .. ..@ cellcentre.offset: num [1:2] 0.325 0.425
.. .. ..@ cellsize         : num [1:2] 0.05 0.05
.. .. ..@ cells.dim        : int [1:2] 96 107
..@ bbox      : num [1:2, 1:2] 0.3 0.4 5.1 5.75
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : NULL
.. .. ..$ : chr [1:2] "min" "max"
..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
.. .. ..@ projargs: chr NA
```

```
> rm(min.x, max.x, min.y, max.y, cells.x, cells.y)
```

Task 22 : Use the fitted variogram model to predict at all the grid points by Ordinary Kriging (OK). •

This is exactly the same as predicting at the 100 evaluation points, above, but here we specify the prediction grid instead as the `newdata`:

```
> k.grid <- krige(Co ~ 1, loc = jura.cal, newdata = jura.raster,
+   model = vmf)
```

```
[using ordinary kriging]
```

```
> summary(k.grid)
```

```
Object of class SpatialGridDataFrame
```

```
Coordinates:
```

```
   min max
```

```

[1,] 0.3 5.10
[2,] 0.4 5.75
Is projected: NA
proj4string : [NA]
Grid attributes:
  cellcentre.offset cellsize cells.dim
1          0.325      0.05         96
2          0.425      0.05        107
Data attributes:
  var1.pred      var1.var
Min.   : 2.91    Min.   : 1.85
1st Qu.: 8.48    1st Qu.: 4.26
Median : 9.73    Median : 5.63
Mean   : 9.54    Mean   : 7.86
3rd Qu.:10.98    3rd Qu.:12.24
Max.   :15.81    Max.   :15.01

```

Task 23 : Plot the predictions and their variances. •

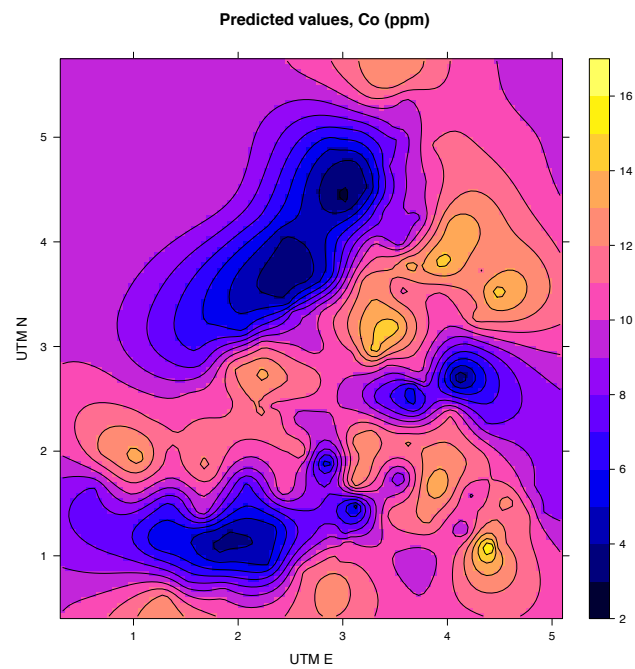
Again we use the `spplot` method to plot a spatial object, but this time with the optional `contour=T` argument to get a superimposed contour lines, and with the optional `pretty=T` argument to ensure that the contour lines are at “pretty” (i.e. integer) values.

First the predictions:

```

> print(spplot(k.grid, zcol="var1.pred", pretty=T,
+             contour=T, col.regions=bpy.colors(64),
+             main="Predicted values, Co (ppm)",
+             xlab="UTM E", ylab="UTM N",
+             scales=list(draw=T)))

```

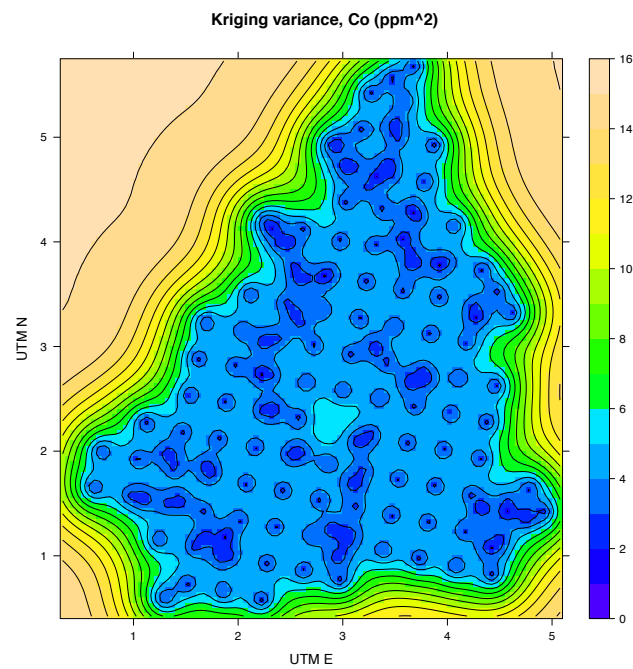
Q15 : *Describe the major spatial features of the Ordinary Kriging prediction.*

[Jump to A15](#)

•

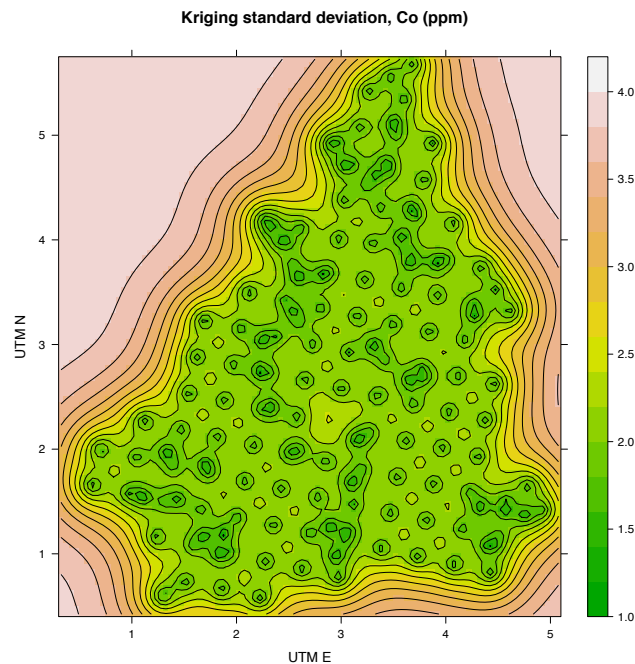
Now the kriging variances:

```
> print(spplot(k.grid, zcol="var1.var", pretty=T,
+             contour=T, col.regions=topo.colors(64),
+             main="Kriging variance, Co (ppm^2)",
+             xlab="UTM E", ylab="UTM N",
+             scales=list(draw=T)))
```



This can also be displayed as a kriging standard deviation, in the same units as the prediction. To do this, we first add the standard deviation as a field in the kriging object, then display it:

```
> k.grid$var1.sd <- sqrt(k.grid$var1.var)
> print(spplot(k.grid, zcol="var1.sd", pretty=T,
+             contour=T, col.regions=terrain.colors(64),
+             main="Kriging standard deviation, Co (ppm)",
+             xlab="UTM E", ylab="UTM N",
+             scales=list(draw=T)))
```



Q16 : Describe the major spatial features of the variances of the Ordinary Kriging prediction. *Jump to A16* •

The kriging variances are only an **internal** measure of quality; like the predictions they depend on the **model**. We will see how to evaluate the quality of these predictions with **external** measures (i.e. **evaluation**) in Exercise 6.

4.5 * Predicting at a sample point

This **optional** section illustrates a quirk of the OK system: using OK to predict at a sample point:

1. The **prediction** is **exactly the sample value**; this is because there is one point (the sample point) at zero separation, so it receives all the weight in the OK system of equations;
2. The **prediction variance** is **zero**, even if there is a nugget in the variogram model. If any part of the nugget is known error (e.g., lab. precision) this must be added back in.

Note: The mathematics behind this are explored in optional §7.1, below.

Task 24 : Create a spatial object with just the first calibration point. •

This point is the first row (case) of the `jura.cal` object; so we just select this row and assign it to a new object:

```
> (jura.test <- jura.cal[1, ])
```

```

      coordinates      Rock   Land   Cd    Cu    Pb    Co    Cr
1 (2.386, 3.077) Sequanian Meadow 1.74 25.72 77.36 9.32 38.32
      Ni    Zn
1 21.32 92.56

> jura.test$Co

[1] 9.32

```

Task 25 : Predict the Co concentration by OK. •

This is exactly as in §4.2, but at this point as `newdata`, using the fitted model, which has a nugget (as we first show):

```

> print(vmf[1, ])

      model psill range
1   Nug 1.3712     0

> (k.test <- krige(Co ~ 1, loc = jura.cal, newdata = jura.test,
+   model = vmf))

[using ordinary kriging]
      coordinates var1.pred   var1.var
1 (2.386, 3.077)      9.32 5.5348e-31

```

Q17 : *What is the prediction? How does it compare with the sample point?* Jump to A17 •

Q18 : *What is the prediction variance?* Jump to A18 •

Conclusion: a known point included in the sample set is back-predicted exactly with no error. This seems counter-intuitive if there is any nugget variance, but the theory of random fields requires it. If the nugget is (partially) due to measurement error, this can be accounted for after the prediction.

To show that the nugget has not disappeared, let's predict at a point that is just a very short distance away from the sample point.

Task 26 : Change each coördinate of the single point to be predicted by 0.001 km (i.e., 1 m), predict the Co concentration at this point by OK. •

It is not possible to change coördinates of a spatial object directly; instead:

1. Convert from a spatial object to a dataframe;
2. Adjust the fields representing coördinates;
3. Convert back to a spatial object by assigning coördinates.

```

> (jura.test <- as.data.frame(jura.test))

```

```

      X      Y      Rock  Land  Cd    Cu    Pb    Co    Cr    Ni
X 2.386 3.077 Sequanian Meadow 1.74 25.72 77.36 9.32 38.32 21.32
      Zn
X 92.56

> jura.test$X <- jura.test$X + 0.001
> jura.test$Y <- jura.test$Y + 0.001
> coordinates(jura.test) <- ~X + Y
> print(jura.test)

      coordinates      Rock  Land  Cd    Cu    Pb    Co    Cr
X (2.387, 3.078) Sequanian Meadow 1.74 25.72 77.36 9.32 38.32
      Ni      Zn
X 21.32 92.56

```

Now we can kriging and observe the results:

```

> (k.test <- krige(Co ~ 1, loc = jura.cal, newdata = jura.test,
+   model = vmf))

[using ordinary kriging]
      coordinates var1.pred var1.var
X (2.387, 3.078)   9.4493   2.3013

```

Q19 : *What is the prediction? How does it compare with the sample point (which was the same as the OK prediction at this point)?* [Jump to A19](#) •

Q20 : *What is the prediction variance? How does it compare to the nugget variance?* [Jump to A20](#) •

4.6 Answers

A9 : *There are two:*

1. `var1.pred`: the predicted value of the attribute; units are $\text{mg kg}^{-1} \text{ Co}$
2. `var1.var`: the variance of the predicted value; units are $\text{mg kg}^{-1} \text{ Co}^2$

[Return to Q9](#) •

A10 : *Field `var1.pred` corresponds to the estimated value \hat{z} ; the **square root** of field `var1.var` to the kriging prediction standard deviation σ .* [Return to Q10](#) •

A11 : 8.73 mg kg^{-1}

[Return to Q11](#) •

A12 : *262 point-pairs are separated by 100 m (0.1 km) or less; the closest spacing*

is 5 m.

[Return to Q12](#) •

A13 : The area of the bounding box is 22.88 km²; a grid with 100 m spacing would require 2 289 points; 50 m spacing would require 9 153 points; 20 m spacing would require 57 201 points; and 5 m spacing would require 915 209 points; i.e. almost a million. [Return to Q13](#) •

A14 : 10368 points, from $X = (0.3, 5.1)$, $Y = (0.4, 5.8)$ [Return to Q14](#) •

A15 : The predictions form a smooth surface, with no abrupt changes. There are clear “hot” and “cold” spots (high and low values). At the NW and NE corners of the grid there is a uniform prediction; this is the spatial mean. [Return to Q15](#) •

A16 : Variances are lowest near sample points, and in particular near clusters where several points were sampled nearby. Away from the area where samples were located the variance increases rapidly and reaches the variance of the dataset. [Return to Q16](#) •

A17 : The prediction is 9.32, exactly the same as this calibration point. [Return to Q17](#) •

A18 : The prediction variance is effectively 0. [Return to Q18](#) •

A19 : The prediction is 9.4493, considerably different from the calibration point 9.32 – even though this prediction point is only $\sqrt{2}$ m away from a known sample! [Return to Q19](#) •

A20 : The prediction variance is 2.3013; this is quite a bit higher than the nugget, 1.3712. [Return to Q20](#) •

5 Saving the workspace

Several objects created or modified here will be used later exercises:

1. All sample points `jura.all`
2. Calibration points `jura.cal`
3. Validation points `jura.val`
4. Prediction grid `jura.raster`
5. The fitted variogram model for Co, `vmf`
6. Predictions of Co at the evaluation points, `k.val`
7. Predictions of Co on the grid, `k.grid`

In order to avoid re-creating them each time, we **save** them as an **R data set**.

Task 27 : Save the above-list objects in an R data set disk file. •

One or more objects are saved with the **save** method, specifying a file name; we did this with the full Jura data set in Exercise 2, §3.2. To save more than one object, just list them:

```
> save(jura.all, jura.cal, jura.val, jura.raster, jura.grid,
+      vmf, k.val, k.grid, file = "JuraEx4.RData")
```

These can then be loaded with the **load** method in later exercises.

6 * Ordinary kriging with anisotropy

In Exercise 3 §4.6 we saw how to fit a variogram model incorporating geometric anisotropy (same model and sill, different ranges). In this section we see how to predict by OK with such a model, and compare it to OK with an isotropic model. As explained in Exercise 3 §4.6 the Jura data exhibits no anisotropy, so for this section we return to the Meuse dataset used in Exercise 2 §8 and Exercise 3 §4.6.

We begin by summarizing the variogram modelling of Exercise 3 §4.6.

Task 28 : Load the **meuse** sample dataset from the **sp** package and convert it from a dataframe to a spatial object by specifying its coördinates. •

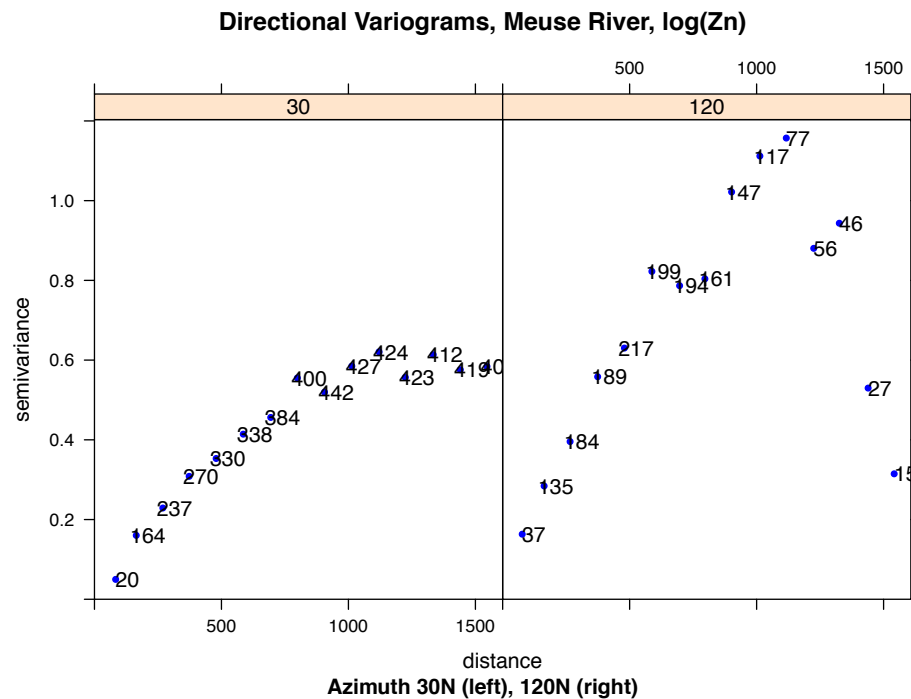
```
> require(sp)
> data(meuse)
> coordinates(meuse) <- ~x + y
```

6.1 Computing the anisotropic variogram

This repeats some of Exercise 3 §4.6.

Task 29 : Compute and display the directional variograms of the logarithm of Zn content at 30° N and 120° N, i.e. the suspected major and minor axes of the anisotropy ellipse. •

```
> require(gstat)
> v.a <- variogram(log(zinc)~1, meuse, alpha=c(30,120), cutoff=1600)
> print(plot(v.a,
+           main="Directional Variograms, Meuse River, log(Zn)",
+           sub="Azimuth 30N (left), 120N (right)",
+           pl=T, pch=20, col="blue"))
```



As explained in Exercise 3 §4.6, these directional variograms show strong anisotropy – the two variograms are quite different.

An important detail about this computation is the **horizontal** (angular) **tolerance** on both sides the major and minor directions. By default, as here when not specified, this optional `tol.hor` argument to the `variogram` function is 90° divided by the number of angles specified with the `alpha` argument. Here we specified two angles (30° N and 120° N), so each will have a horizontal tolerance of 45° on both sides of the directions; that is, every point-pair will be included in either the 30° N or 120° N directional variograms. This can lead to a “smearing” effect where the directionality is not so clear; on the other hand it allows for some angular deviations from single directions.

Challenge: Compute and display the two directional variograms with smaller angular tolerances, e.g., by specifying `tol.hor=30` or narrower. Compare these with the default `tol.hor=45`. What are the differences? You could also use the variograms with narrower angular tolerance for the following steps (variogram modelling) and compare with the results.

6.2 Geometric anisotropy

Recall that geometric anisotropy is when the major and minor directions (axes) are fit with the same model and sill, but with different ranges. In essence, the **anisotropy ratio** transforms the ellipse into a circle.

In the present example, the anisotropy does not appear to be geometric at longer ranges (past about 500 m). However, we will investigate how to model it as geometric anisotropy because (1) the small sample size makes the variogram for the minor axis not very reliable; (2) the dimension of the study area in the direction of the minor axis is small, so that modelling

longer ranges is not important for interpolation.

Task 30 : Fit a spherical variogram model, with geometric anisotropy, to the directional variograms and display the fit. •

From the two directional variograms we estimate that the structural sill of the major axis variogram is ≈ 0.55 , the range is ≈ 1100 m, and the nugget variance is ≈ 0.05 (for a total sill of ≈ 0.6). This total sill is reached at ≈ 550 m along the minor axis, so the anisotropy ratio is $1100/550 = 0.5$ in the minor direction, i.e., 120° .

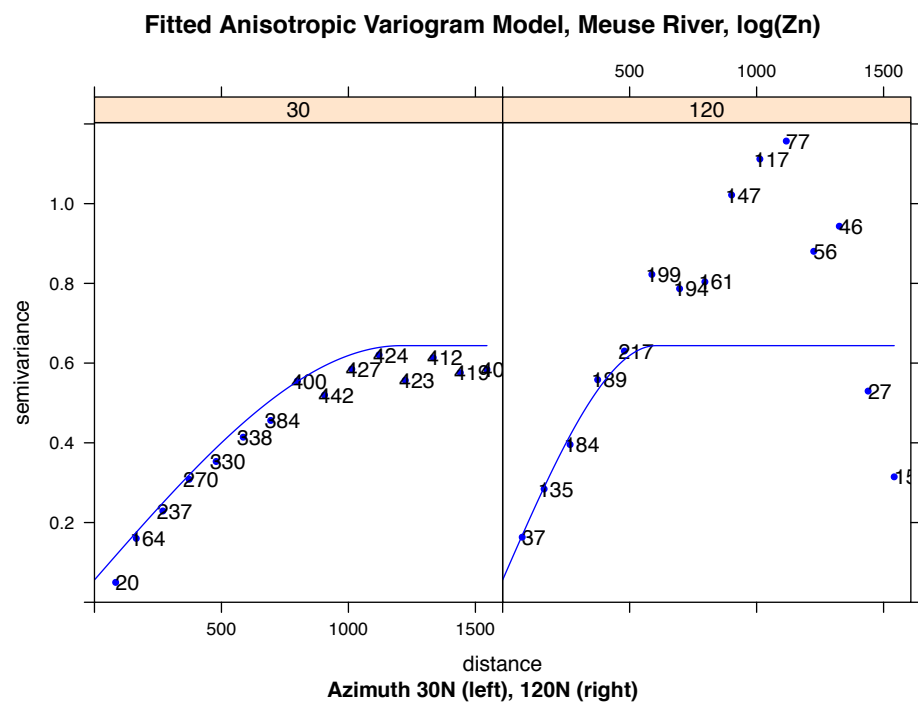
The tricky part here is the **anis** “anisotropy” argument to the **vgm** “specify a variogram model” function. In 2D (as in the present case), this is a list (composed with the **c** function) of:

1. the direction of the **major** axis, in degrees from N;
2. the “shrinkage” along the **minor** axis; that is, the minor axis’ proportion of the major axis range specified with the **range** argument.

So in this case we specify **range** = 1100, that is, the range along the major axis is 1100 m; and **anis** = **c**(30, 0.5), that is, the major axis is at 30°N , and the range along the orthogonal axis (here, 120°N) is half of the range along the major axis, i.e., $1100 * 0.5 = 550$ m.

The automatic fit with **fit.variogram** adjusts the major range, structural sill, and nugget, but not the angle or anisotropy ratio.

```
> (vmf.a <-  
+   fit.variogram(v.a,  
+               vgm(psill=0.55, model="Sph", range=1100,  
+               nugget=0.05, anis=c(30, 0.5))))  
  
      model      psill      range angl anis1  
1   Nug 0.0560948    0.000    0    1.0  
2   Sph 0.5877192 1208.673   30    0.5  
  
> print(plot(v.a,  
+   main="Fitted Anisotropic Variogram Model, Meuse River, log(Zn)",  
+   model=vmf.a,  
+   sub="Azimuth 30N (left), 120N (right)",  
+   pl=T, pch=20, col="blue"))
```



Q21 : How well does the model fit the empirical variogram? Identify any discrepancies. Jump to A21 •

Challenge: Recompute the empirical variogram with a shorter cutoff and re-fit the anisotropic model.

Task 31 : Compute the best omnidirectional spherical model for the same variable and display the fitted model on the empirical variogram. •

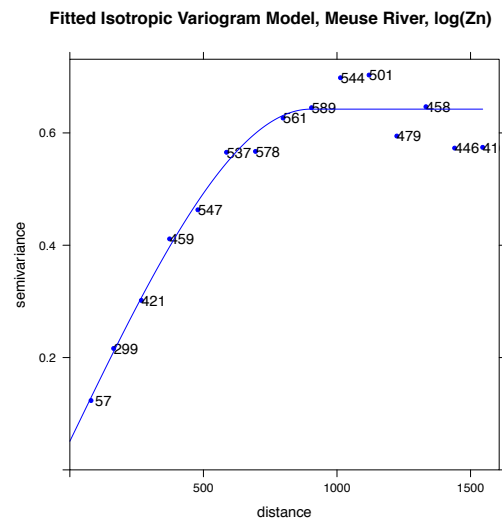
```
> v <- variogram(log(zinc)~1, meuse, cutoff=1600)
> (vmf <- fit.variogram(v, vgm(0.55, "Sph", 1100, 0.05)))

  model      psill    range
1  Nug 0.05097029  0.0000
2  Sph 0.59139781 901.8033

> attributes(vmf)$SSErr

[1] 9.453761e-06

> print(plot(v,
+   main="Fitted Isotropic Variogram Model, Meuse River, log(Zn)",
+   model=vmf, pl=T, pch=20, col="blue"))
```



This model fits very well; however note that the range is a compromise between the very different ranges in the two orthogonal directions.

Task 32 : Predict by Ordinary Kriging over the Meuse grid with both the anisotropic and isotropic models, and summarize the (non-spatial) differences for the predictions and their variances. •

A grid of the Meuse area at 40 m horizontal resolution is provided with the `sp` package as `meuse.grid`.

The exact same `krige` method of the `gstat` package is used for the anisotropic and isotropic cases; the only difference is in the variogram model.

```
> data(meuse.grid)
> coordinates(meuse.grid) <- ~x + y
> gridded(meuse.grid) <- TRUE
> k <- krige(log(zinc) ~ 1, loc = meuse, newdata = meuse.grid,
+   model = vmf)

[using ordinary kriging]

> k.a <- krige(log(zinc) ~ 1, loc = meuse, newdata = meuse.grid,
+   model = vmf.a)

[using ordinary kriging]

> summary(k.a$var1.pred - k$var1.pred)

      Min.   1st Qu.   Median     Mean   3rd Qu.    Max.
-0.330400 -0.035760  0.005721  0.017200  0.057860  0.412800

> summary(k.a$var1.var - k$var1.var)

      Min.   1st Qu.   Median     Mean   3rd Qu.    Max.
-0.045200  0.003089  0.011660  0.014150  0.021630  0.097860
```

Task 33 : Display the predictions side-by-side on the same scale. •

```

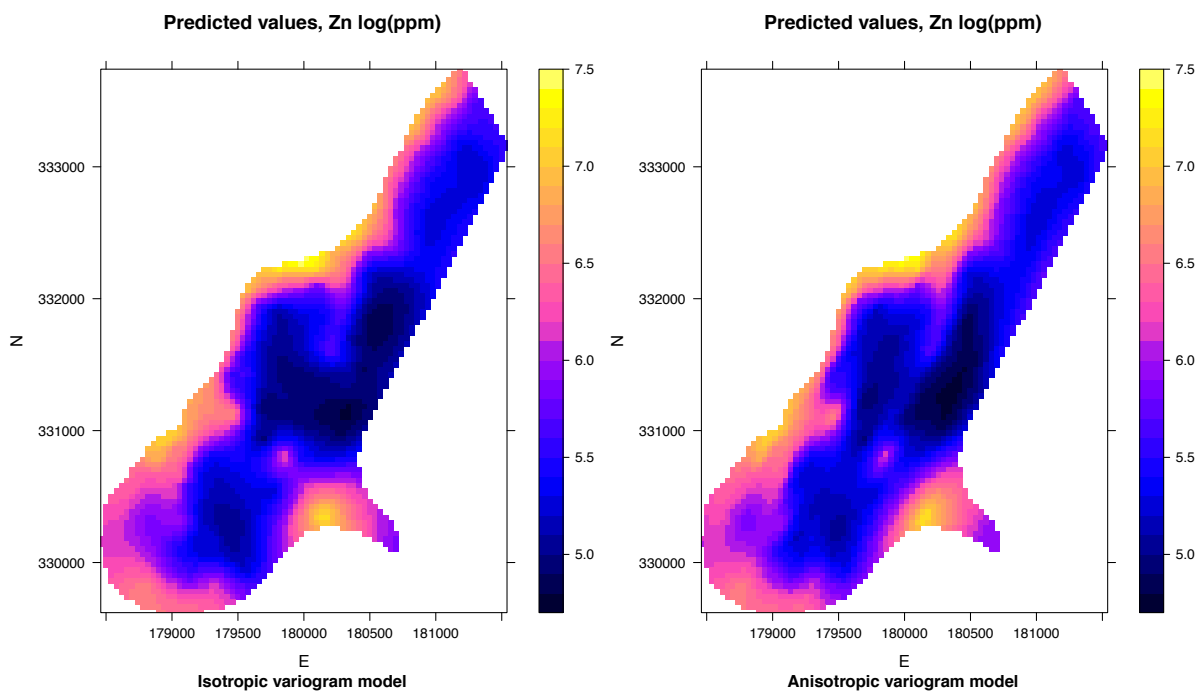
> p.at <- seq(round(min(k$var1.pred, k.a$var1.pred),1)-0.1,
+             round(max(k$var1.pred, k$var1.pred.a)+0.1, 1),
+             by=0.1)
> p1 <- spplot(k, zcol="var1.pred",
+             col.regions=bpy.colors(length(p.at)),
+             key.space="right", at=p.at,
+             main="Predicted values, Zn log(ppm)",
+             sub="Isotropic variogram model", xlab="E", ylab="N",
+             scales=list(draw=T))
> p2 <- spplot(k.a, zcol="var1.pred",
+             col.regions=bpy.colors(length(p.at)),
+             key.space="right", at=p.at,
+             main="Predicted values, Zn log(ppm)",
+             sub="Anisotropic variogram model", xlab="E", ylab="N",
+             scales=list(draw=T))
>

```

```

> print(p1, split=c(1,1,2,1), more=T)
> print(p2, split=c(2,1,2,1), more=F)

```



Q22 : Describe the spatial differences between the two predictions. [Jump to A22](#) •

We can also map the differences themselves.

Task 34 : Compute and display the differences between the two predictions.

•

It's easiest to compute the differences and add as a field to one of the prediction grid data frames. To visualize, it's best to use a different colour scheme (here, the colour ramp produced by the `heat.colors` method) so that we don't visually confuse predictions and differences.

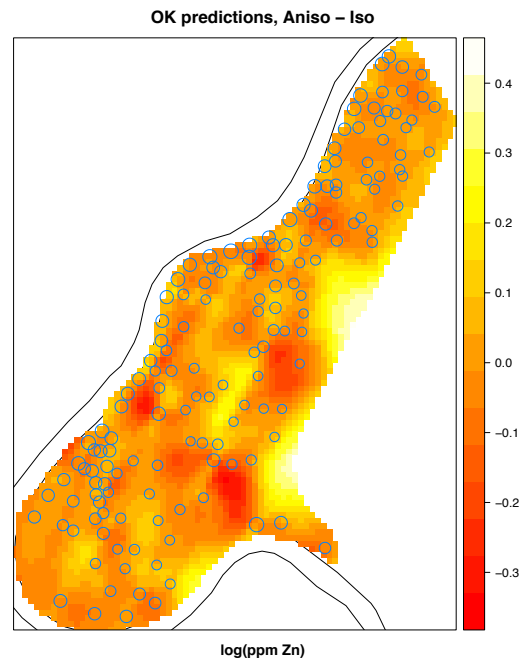
We also show the values at each point as a postplot, using the `sp.layout` optional argument to `splot`. This argument takes a list of specifications of plots to add to the main figure.

Finally, we add the river outline to help interpretation.

Note: The river outline is provided with the `sp` package as `meuse.riv`, a matrix of coordinates. It can be converted into an `SpatialPolygons` object with the `SpatialPolygons` method, which requires a list of `Polygons` objects; this is built with the `Polygons` method, which itself requires a list of `Polygon` objects, built with the `Polygon` method.

This is a lot of work just to get one polygon, but the logical sequence of single polygon, to set of polygons, to spatially-explicit polygons, is logical.

```
> data(meuse.riv)
> meuse.riv.poly <- SpatialPolygons(
+   list(Polygons(list(Polygon(meuse.riv)),
+                     "meuse.riv")))
> k.a$var1.diff <- k.a$var1.pred - k$var1.pred
> sp.pts <- list("sp.points", meuse, pch=1,
+               cex=2*log(meuse$zinc)/max(log(meuse$zinc)))
> sp.riv <- list("sp.polygons", meuse.riv.poly)
> splot(k.a, zcol="var1.diff", col.regions=heat.colors(64),
+       main="OK predictions, Aniso - Iso", sub="log(ppm Zn)",
+       sp.layout=list(sp.pts, sp.riv))
```



Q23 : Where are the largest prediction differences, both positive and negative? *Jump to A23*

•

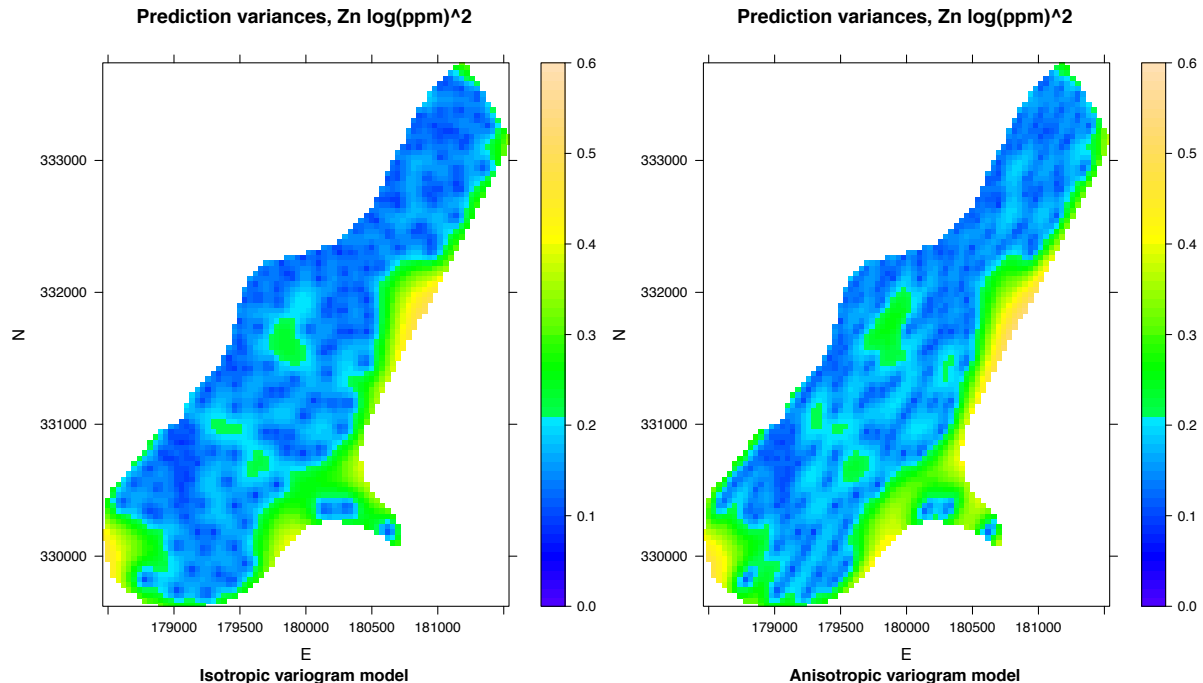
Task 35 : Display the prediction variances side-by-side on the same scale.

•

It's best to use a different colour scheme (here, the colour ramp produced by the `topo.colors` method) so that we don't visually confuse predictions and their variances.

```
> p.at <- seq(round(min(k$var1.var, k.a$var1.var),1)-0.1,
+             round(max(k$var1.var, k$var1.var.a)+0.1, 1),
+             by=0.01)
> p1 <- spplot(k, zcol="var1.var",
+             col.regions=topo.colors(length(p.at)),
+             key.space="right", at=p.at,
+             main="Prediction variances, Zn log(ppm)^2",
+             sub="Isotropic variogram model",
+             xlab="E", ylab="N", scales=list(draw=T))
> p2 <- spplot(k.a, zcol="var1.var",
+             col.regions=topo.colors(length(p.at)),
+             key.space="right", at=p.at,
+             main="Prediction variances, Zn log(ppm)^2",
+             sub="Anisotropic variogram model",
+             xlab="E", ylab="N", scales=list(draw=T))
> print(p1, split=c(1,1,2,1), more=T)
> print(p2, split=c(2,1,2,1), more=F)
```

```
> print(p1, split=c(1,1,2,1), more=T)
> print(p2, split=c(2,1,2,1), more=F)
```

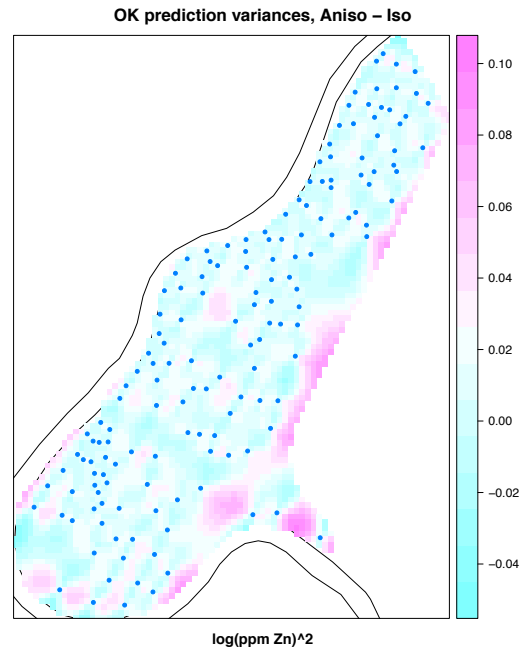


Q24 : Describe the spatial pattern of the differences between the two OK prediction variances. Jump to A24 •

Task 36 : Compute and display the differences between the two prediction variances. •

Again we compute the differences and add as a field to one of the prediction grids data frames, and visualize with yet another colour scheme. We overlay the points, but here only their location; recall that OK prediction variances does not depend on data values, only the point configuration.

```
> k.a$var1.diff.v <- k.a$var1.var - k$var1.var
> sp.pts <- list("sp.points", meuse, pch=20)
> spplot(k.a, zcol="var1.diff.v", col.regions=cm.colors(64),
+       main="OK prediction variances, Aniso - Iso",
+       sub="log(ppm Zn)^2",
+       sp.layout=list(sp.pts, sp.riv))
```



Q25 : Where are the largest prediction variance differences, both positive and negative? *Jump to A25 •*

Q26 : For this dataset is isotropic or anisotropic OK more appropriate? Why? *Jump to A26 •*

Task 37 : Clean up from this section, leaving the points, grid, and empirical variogram for the next section. •

```
> rm(v, vmf, vmf.a, k, k.a, p.at, sp.pts, sp.riv,
+     meuse.riv, meuse.riv.poly, p1, p2)
```

6.3 Zonal anisotropy

We saw that the anisotropy is not geometric beyond a minor range of about 600 m. Thus the anisotropy is **zonal**. Modelling this is more complicated than for the geometric case, because two geometric models (one per direction) must be combined.

In this case we have a geometric model that fits fairly well up to 550 m in the minor range. That is, an ellipse with eccentricity $\sqrt{1 - (0.5)^2} \approx 0.8660$ represents the anisotropy within a range of 600 m (minor) and 1200 m (major). This will give proper weighting for kriging.

However, points with greater separations in the minor axis (or, proportionally at intermediate angles) will receive too much weight with the geometric

model – the semivariance between these and the prediction point as modelled by the geometric model is too low. So we need to add another structure to the model, with a longer range and higher partial sill in the minor direction. This is possible by specifying the optional `add.to` argument to the `vgm` function, when describing the added structure.

To ensure that the added component does not interfere with the successful geometric model along the major axis, the range of the added component is set to a very large value, and the anisotropy ratio to a correspondingly very small value **along the opposite axis**; these should be set to cancel each other out except for the desired range.

For example, suppose the desired range is 1100 m along a given axis, say 120°. We would then specify `anis=c(30, .0001)`, i.e., the orthogonal axis and a range that multiplies the desired range by the inverse of the anisotropy ratio, here $1/0.0001 = 10000$. Then the range along the orthogonal axis (the one which we don't want to modify) becomes very long, so the existing model is hardly affected in that direction – the sill is not reached until a very long range, so the slope of the added variogram is very low near the origin, where the existing model is well-fit. On the given axis, however, the anisotropy ratio cancels out the very long range factor, restoring the desired range along that axis:

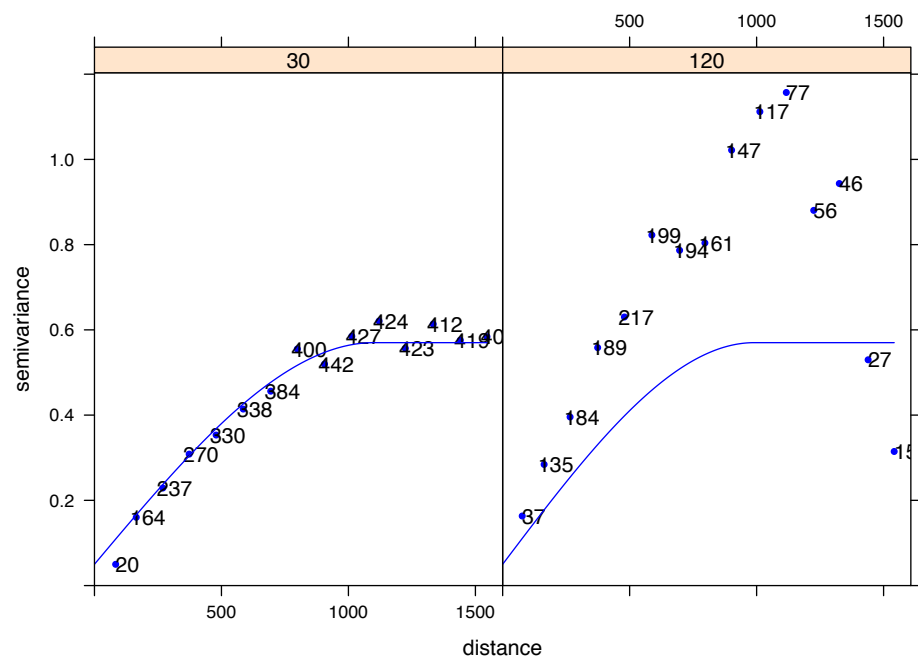
$$\begin{aligned} 10000 * 0.0001 &= 1, \text{therefore:} \\ 1100 * (10000 * 0.0001) &= 1100 - \text{along the opposite axis} \\ 1100 * 10000 &= 11000000 - \text{along the named axis} \end{aligned}$$

Task 38 : Specify an anisotropic variogram model with two components: geometric at short range, with an added component for the minor axis. •

A detail here is that the range of the geometric model along the minor axis should be lengthened (i.e., larger anisotropy ratio) to compensate for the added range of the second component; otherwise

First the adjusted geometric model:

```
> vm.major <- vgm(0.52, "Sph", 1100, 0.05, anis = c(30, 0.9))
> plot(v.a, vm.major, pl = T, pch = 20, col = "blue")
```

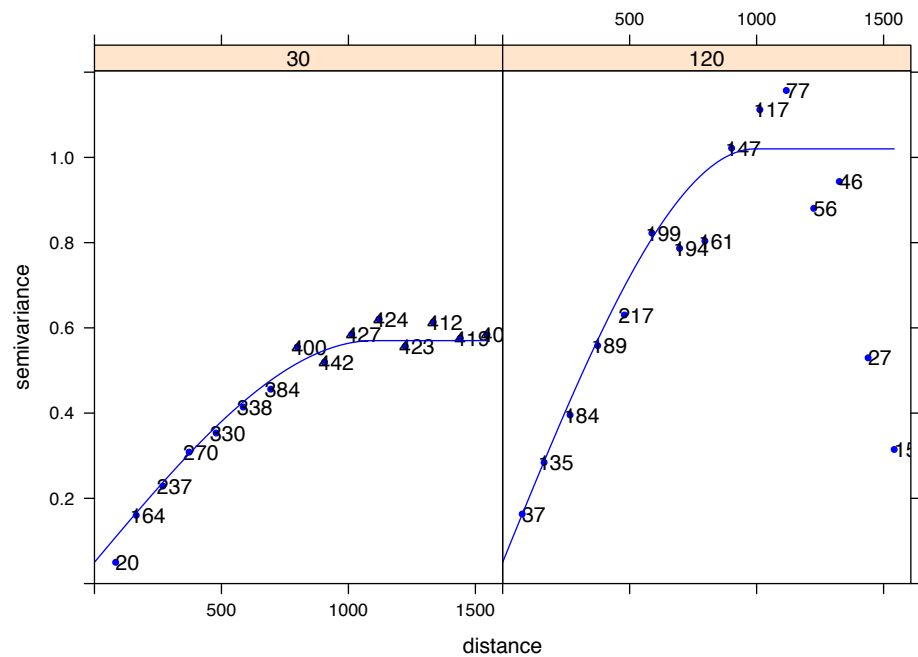


And then add the second component:

```
> (vm.zonal <- vgm(0.45, "Sph", 1000 * 10000, anis = c(30, 1e-04),
+   add.to = vm.major))

model psill   range ang1 anis1
1  Nug  0.05 0.0e+00   0 1e+00
2  Sph  0.52 1.1e+03   30 9e-01
3  Sph  0.45 1.0e+07   30 1e-04

> plot(v.a, vm.zonal, pl = T, pch = 20, col = "blue")
```



Note: Automatic fitting with `fit.variogram` doesn't usually work because of the large difference between ranges, leading to numerical instability and a computationally-singular matrix during the attempted minimization.

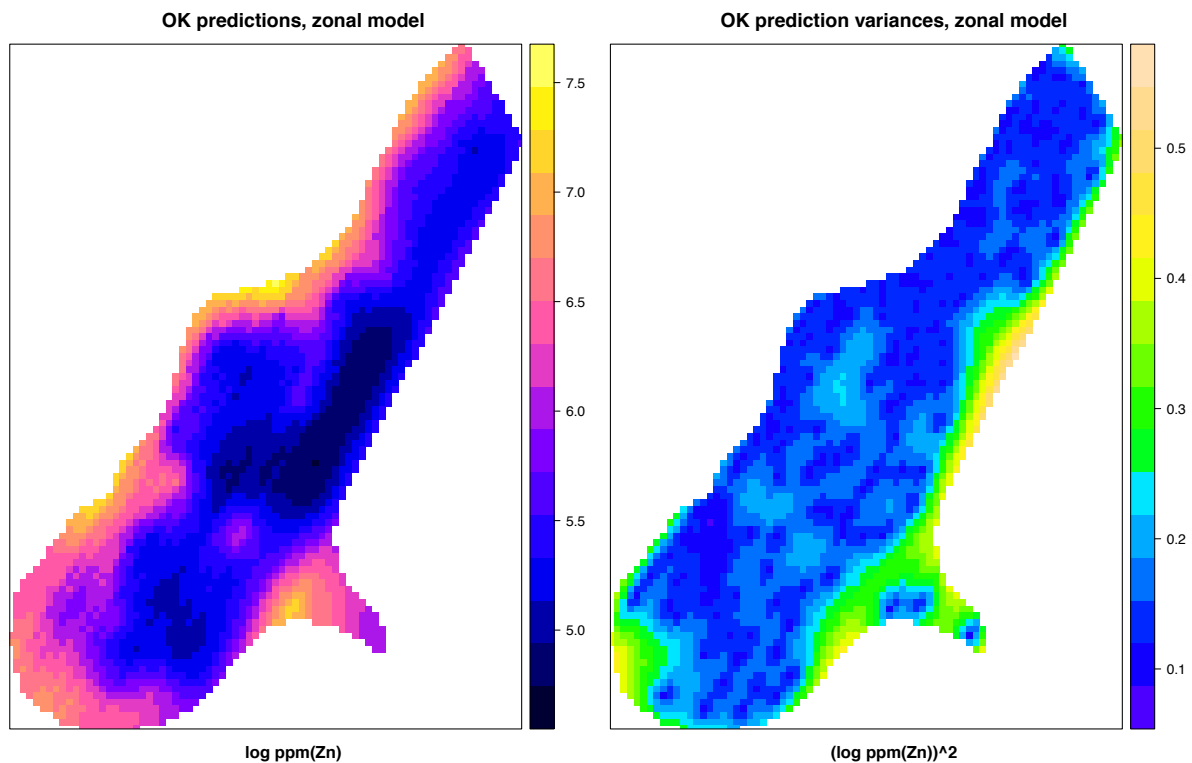
Task 39 : Predict by Ordinary Kriging over the Meuse grid; display the predictions and their variances. •

```
> k.z <- krige(log(zinc) ~ 1, loc=meuse,
+             newdata=meuse.grid, model=vm.zonal)

[using ordinary kriging]

> p1 <- spplot(k.z, zcol="var1.pred",
+             col.regions=bpy.colors(64),
+             main="OK predictions, zonal model",
+             sub="log ppm(Zn)")
> p2 <- spplot(k.z, zcol="var1.var",
+             col.regions=topo.colors(64),
+             main="OK prediction variances, zonal model",
+             sub="(log ppm(Zn))^2")
```

```
> print(p1, split = c(1, 1, 2, 1), more = T)
> print(p2, split = c(2, 1, 2, 1), more = F)
```



Task 40 : Clean up from this section. •

```
> rm(k.z, vm.zonal, vm.major, v.a, meuse, meuse.grid, p1, p2)
```

Q27 : What are the spatial differences between this prediction (and its variances) and that using the geometric model of the previous §6.2? *Jump to A27* •

Challenge: Compare the results of the zonal and geometric models using the same techniques as the previous §6.2.

Challenge: After completing §3 “Cross-validation” of Exercise 6, compare the three predictions (isotropic OK, geometric anisotropic OK, zonal anisotropic OK) by cross-validation. Display the residuals as bubble plots and compare their numeric summaries. Which approach gives the most reliable predictions? Explain why.

6.4 Answers

A21 : The model fits reasonably well first 500 m of the 120° variogram, but at longer ranges the fit is very bad, seriously underfitting the sill. The fit to the 30° variogram is a bit high, because of the influence of the 120° variogram's longer-range point-pairs on the sill. [Return to Q21 •](#)

A22 : The anisotropic predictions clearly show the effect of the longer major axis at 30° N. The higher predictions along the NW side are narrower (aligned with that axis); the lower predictions in the centre of the area are elongated along the NNE-SSW axis. This effect is also seen around the “hot spot” in the SE. [Return to Q22 •](#)

A23 : The largest positive differences (anisotropic predicts higher) are on the E edge, especially above the “hot spot” in the SE and in the centre-E. These are influenced by higher values along the NNE-SSW axis near them.

The largest positive differences (anisotropic predicts lower) are at several patches just inside (SE) of the W edge (main river) and two large patches in the centre. These are areas where low values are to the NNE and/or SSE. [Return to Q23 •](#)

A24 : The variances are also aligned NNE-SSW for the anisotropic OK predictions, whereas they are isotropic for isotropic OK. [Return to Q24 •](#)

A25 : The largest positive differences (anisotropic prediction variance higher) are along the E edge and to the NW and SE of the “hot spot” at the first bend in the river. This is because there are few points in the NNE-SSW direction from these locations.

The largest negative differences (anisotropic prediction variance higher) are at many locations; note that these differences are not as large in absolute value as the positive differences. These are locations with points to the orthogonal direction (WNW-SSE) that in the anisotropic case have less weight. [Return to Q25 •](#)

A26 : The directional variograms show strong anisotropy, due to the main river axis and the presumed origin of the metals from river flooding. Although the kriging prediction variances are on average higher for anisotropic OK, this seems realistic. So, anisotropic OK is preferred here. [Return to Q26 •](#)

A27 : The pattern of predictions with the zonal model are more strongly aligned with the major axis, because the influence of minor axis points further than ≈ 600 m is less. The pattern of prediction variances is along more linear; this is especially apparent along the E side of the grid, where there are very narrow bands of similar variances aligned at 30° N. [Return to Q27 •](#)

7 * Insight into the OK system

In this **optional** section we look at how **gstat** solves the kriging system.

Because the size of the kriging matrix depends on the number of sample points, we first reduce the size of the problem.

Task 41 : Select the first six sample points of the prediction sample, and their Co concentration. •

To select one row (case), we use the row selection operator `[]`, with the sequence operator `:` to select the first six. We also specify the data field:

```
> (jura.pred.6 <- jura.pred[1:6, "Co"])
```

```
      coordinates      Co
1 (2.386, 3.077)  9.320
2 (2.544, 1.972) 10.000
3 (2.807, 3.347) 10.600
4 (4.308, 1.933) 11.920
5 (4.383, 1.081) 16.320
6 (3.244, 4.519)  3.508
```

Q28 : What is the bounding box of these six points? How does that compare with the bounding box for the whole prediction sample? *Jump to A28* •

```
> bbox(jura.pred.6)
```

```
      min  max
Xloc 2.386 4.383
Yloc 1.081 4.519
```

```
> bbox(jura.pred)
```

```
      min  max
Xloc 0.626 4.92
Yloc 0.580 5.69
```

Task 42 : Define a one-point `SpatialPoints` object at (3.0,2.5), and plot its location along with the six sample points. •

This location is chosen to illustrate kriging weights as influenced by point configuration.

```
> jura.pt <- SpatialPoints(data.frame(Xloc = 3, Yloc = 2.5))
> str(jura.pt)
```

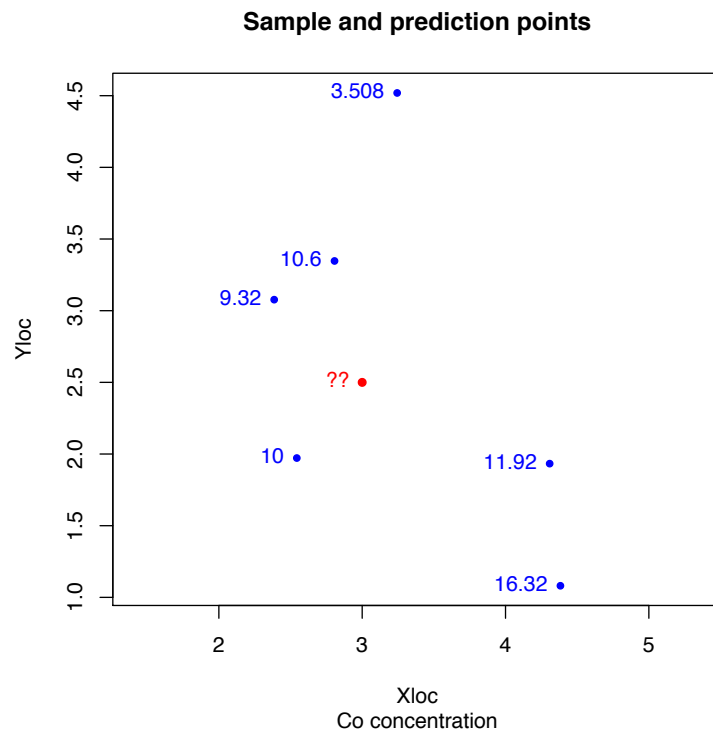
```
Formal class 'SpatialPoints' [package "sp"] with 3 slots
 ..@ coords      : num [1, 1:2] 3 2.5
 .. ..- attr(*, "dimnames")=List of 2
 .. .. ..$ : NULL
 .. .. ..$ : chr [1:2] "Xloc" "Yloc"
 ..@ bbox        : num [1:2, 1:2] 3 2.5 3 2.5
 .. ..- attr(*, "dimnames")=List of 2
 .. .. ..$ : chr [1:2] "Xloc" "Yloc"
 .. .. ..$ : chr [1:2] "min" "max"
```

```

..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
.. ..@ proj4args: chr NA

> plot(coordinates(jura.pred.6), col = "blue", pch = 20,
+       asp = 1, main = "Sample and prediction points",
+       sub = "Co concentration")
> text(coordinates(jura.pred.6)[, "Xloc"], coordinates(jura.pred.6)[,
+       "Yloc"], col = "blue", pos = 2, jura.pred.6$Co)
> points(coordinates(jura.pt), col = "red", cex = 1.2,
+       pch = 20)
> text(coordinates(jura.pt), col = "red", pos = 2,
+       "??")

```



Task 43 : Predict at this point with OK, showing the kriging system, by using the optional `debug.level` argument to the `krige` method. •

The `debug.level` argument is passed by the `krige` function to the more general `predict.gstat` method; `?predict.gstat` shows various useful values; `debug.level=32` prints the covariance matrices, design matrices, solutions, and kriging weights.

```

> k.pt <- krige(Co ~ 1, locations = jura.pred.6, newdata = jura.pt,
+       model = vmf, debug.level = 32)

```

```

[using ordinary kriging]
we're at location X: 3 Y: 2.5 Z: 0
zero block size
we're at point X: 3 Y: 2.5 Z: 0

```

```

# X:
Matrix: 6 by 1
row 0:      1
row 1:      1
row 2:      1
row 3:      1
row 4:      1
row 5:      1
[using generalized covariances: max_val - semivariance()]
# Covariances (x_i, x_j) matrix C (lower triangle only):
Matrix: 6 by 6
row 0:      14.3033801      0      0      0      0
           0      0
row 1:      0.501396836      14.3033801      0      0
           0      0
row 2:      5.52709612      0.0163626925      14.3033801      0
           0      0
row 3:      0      0      0      0      14.3033801
           0      0
row 4:      0      0      0      0      1.91089294
           14.3033801      0
row 5:      0      0      0      0.161934353      0
           0      14.3033801

# X'C-1 X:
Matrix: 1 by 1
row 0:      0.35937143

# beta:
Vector: dim: 1
          10.1549349
# Cov(beta), (X'C-1 X)-1:
Matrix: 1 by 1
row 0:      2.78263634

# Corr(beta):
Matrix: 1 by 1
row 0:      1

# X0 (X values at prediction location x0):
Matrix: 1 by 1
row 0:      1

# BLUE(mu), E(y(x0)) = X0'beta:
Vector: dim: 1
          10.1549349
# Covariances (x_i, x_0), C0:
Matrix: 6 by 1
row 0:      2.00719186
row 1:      3.28488332
row 2:      1.81217503
row 3:      0.00826190553
row 4:      0
row 5:      0

```



```

# C-1 C0:
Matrix: 6 by 1
row 0: 0.0982056253
row 1: 0.226114052
row 2: 0.0884997462
row 3: 0.000588115932
row 4: -7.85706995e-05
row 5: -0.00100194143

# [a] Cov_ij(B,B) or Cov_ij(0,0):
Matrix: 1 by 1
row 0: 14.3033801

# [c] (x0-X'C-1 c0)'(X'C-1 X)-1(x0-X'C-1 c0):
Matrix: 1 by 1
row 0: 0.961009957

# [b] c0'C-1 c0:
Matrix: 1 by 1
row 0: 1.1002577

# Best Linear Unbiased Predictor:
Vector: dim: 1
10.0845083
# MSPE ([a]-[b]+[c]):
Matrix: 1 by 1
row 0: 14.1641324

# kriging weights:
Matrix: 6 by 1
row 0: 0.176715804
row 1: 0.337595687
row 2: 0.17107886
row 3: 0.101442478
row 4: 0.100775792
row 5: 0.11239138

```

Working through this output, we see:

1. The **prediction point** and **block size** (here, zero);
2. The **design matrix** X , here just a column of 1's to predict the spatial mean;
3. The **covariance matrix** C between sample points; this depends of course on the **variogram model**;
4. The **quadratic form** $X^T C^{-1} X$ used in many of the subsequent calculations;
5. The **spatial mean** β , estimated by GLS using the covariance matrix;
6. The **variance** of the spatial mean;
7. The **correlation** of the the spatial mean with itself, of course this is 1 for a single trend parameter (the spatial mean) but is not necessarily

so for a UK system with more trend parameters;

8. The **BLUE** of the mean at the prediction point, here just the spatial mean;
9. The **covariance vector** C_o between the prediction point and each sample point;
10. The product $C^{-1}C_o$;
11. The **within-block** or **at-point** covariance, here just the total variogram sill (estimating the overall variance);
12. A matrix product used in the BLUP;
13. A matrix product used in the BLUP;
14. The **BLUP** at the prediction point; this is the **kriging prediction**;
15. The **kriging prediction variance** at the prediction point;
16. The **kriging weights**, i.e. the weights given to each sample point when their values are summed into the BLUP.

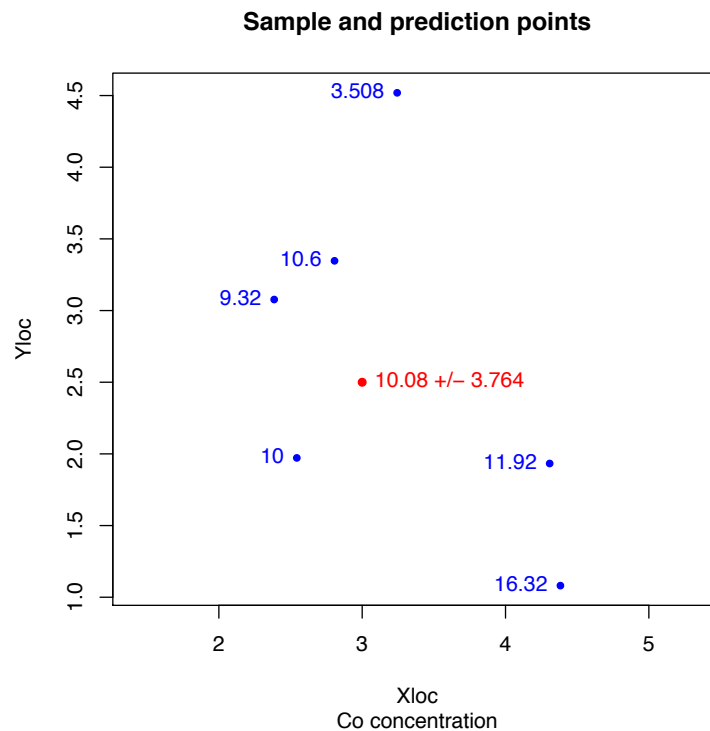
The final result is in the kriging object:

```
> print(k.pt)

coordinates var1.pred var1.var
1 (3, 2.5) 10.085 14.164
```

Task 44 : Plot the sample points and their Co concentration, with the prediction point and its predicted concentration and standard error of the prediction. •

```
> plot(coordinates(jura.pred.6), col = "blue", pch = 20,
+       asp = 1, main = "Sample and prediction points",
+       sub = "Co concentration")
> text(coordinates(jura.pred.6)[, "Xloc"], coordinates(jura.pred.6)[,
+       "Yloc"], col = "blue", pos = 2, jura.pred.6$Co)
> points(coordinates(jura.pt), col = "red", cex = 1.2,
+        pch = 20)
> text(coordinates(jura.pt), col = "red", pos = 4,
+        paste(round(k.pt$var1.pred, 2), "+/-", round(sqrt(k.pt$var1.var),
+        3)))
```



Task 45 : Display this plot but with the kriging weights, rather than the predictions. •

To find the weights, we capture the output into a workspace variable with `capture.output`, and then search through it for the weights.

```
> tmp.ok <- capture.output(krige(Co ~ 1, locations = jura.pred.6,
+   newdata = jura.pt, model = vmf, debug.level = 32))

> (ix <- which(tmp.ok == "# kriging weights:"))

[1] 89

> (n <- as.numeric(strsplit(tmp.ok[ix + 1], " ")[[1]][2]))

[1] 6

> (ix <- which(tmp.ok == "# kriging weights:"))

[1] 89

> ok.wt <- NULL
> for (i in 1:n) ok.wt[i] <- as.numeric(strsplit(tmp.ok[ix +
+   1 + i], ":")[1][2])
> print(ok.wt)

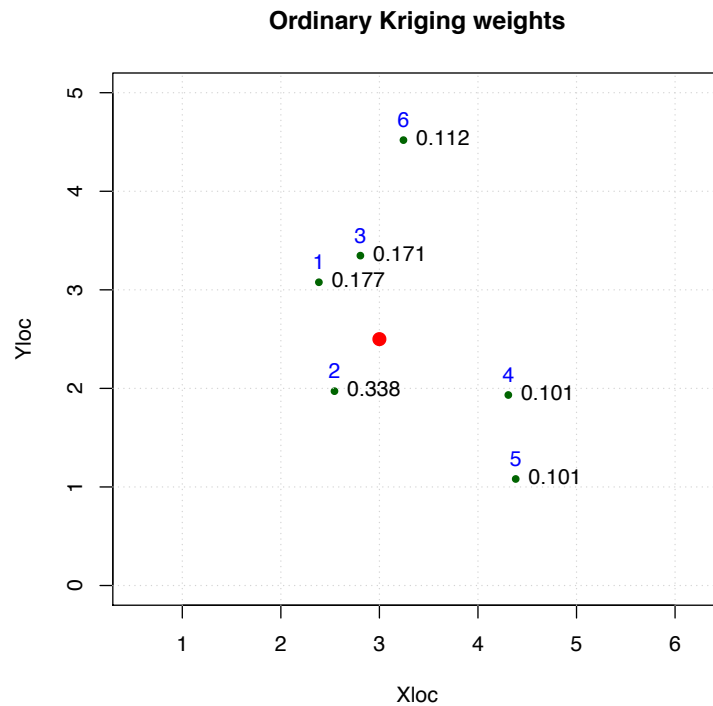
[1] 0.17672 0.33760 0.17108 0.10144 0.10078 0.11239

> plot(coordinates(jura.pred.6), col="darkgreen",
+   pch=20, asp=1, ylim=c(0,5),
```

```

+      main="Ordinary Kriging weights")
> grid()
> text(coordinates(jura.pred.6)[,"Xloc"],
+       coordinates(jura.pred.6)[,"Yloc"],
+       col="black", pos=4, round(ok.wt, 3))
> text(coordinates(jura.pred.6)[,"Xloc"],
+       coordinates(jura.pred.6)[,"Yloc"],
+       col="blue", pos=3, 1:6)
> points(coordinates(jura.pt), col="red",
+        cex=2, pch=20)

```



Q29 : Which points get the most kriging weight?

Jump to A29 •

We will return to this example in Universal Kriging.

7.1 The OK system at a sample point

What happens to the system if we try to predict at a known point? As explained in §4.5, the known point (at zero separation from the point to be predicted) gets all the weight, is predicted exactly, and the MSPE (kriging prediction variance) is zero. Here we demonstrate how the OK system looks in this case.

Task 46 : Create a spatial object with a single prediction point: the first point of the calibration set, and predict at this point from the six sample points, again showing the diagnostics. •

The point to be predicted does not need any data, just a location (although, there is no harm in keeping the data); so we convert the object to the `SpatialPoints` class with the generic `as` method:

```
> (jura.pt <- as(jura.pred[1, ], "SpatialPoints"))

SpatialPoints:
  Xloc  Yloc
1 2.386 3.077
Coordinate Reference System (CRS) arguments: NA

> (k.pt <- krige(Co ~ 1, locations = jura.pred.6, newdata = jura.pt,
+               model = vmf, debug.level = 32))

[using ordinary kriging]
we're at location X: 2.386 Y: 3.077 Z: 0
zero block size
we're at point X: 2.386 Y: 3.077 Z: 0

# X:
Matrix: 6 by 1
row 0:      1
row 1:      1
row 2:      1
row 3:      1
row 4:      1
row 5:      1
[using generalized covariances: max_val - semivariance()]
# Covariances (x_i, x_j) matrix C (lower triangle only):
Matrix: 6 by 6
row 0:  14.3033801      0      0      0
      0      0
row 1:  0.501396836  14.3033801      0      0
      0      0
row 2:  5.52709612  0.0163626925  14.3033801      0
      0      0
row 3:      0      0      0      0  14.3033801
      0      0
row 4:      0      0      0      0  1.91089294
      14.3033801      0
row 5:      0      0      0  0.161934353      0
      0  14.3033801

# X'C-1 X:
Matrix: 1 by 1
row 0:  0.35937143

# beta:
Vector: dim: 1
      10.1549349
# Cov(beta), (X'C-1 X)-1:
Matrix: 1 by 1
row 0:  2.78263634

# Corr(beta):
Matrix: 1 by 1
```

```

row 0:          1

# X0 (X values at prediction location x0):
Matrix: 1 by 1
row 0:          1

# BLUE(mu), E(y(x0)) = X0'beta:
Vector: dim: 1
      10.1549349
# Covariances (x_i, x_0), C0:
Matrix: 6 by 1
row 0:      14.3033801
row 1:      0.501396836
row 2:      5.52709612
row 3:          0
row 4:          0
row 5:          0

# C-1 C0:
Matrix: 6 by 1
row 0:          1
row 1:          0
row 2:          0
row 3:          0
row 4:          0
row 5:          0

# [a] Cov_ij(B,B) or Cov_ij(0,0):
Matrix: 1 by 1
row 0:      14.3033801

# [c] (x0-X'C-1 c0)'(X'C-1 X)-1(x0-X'C-1 c0):
Matrix: 1 by 1
row 0:          0

# [b] c0'C-1 c0:
Matrix: 1 by 1
row 0:      14.3033801

# Best Linear Unbiased Predictor:
Vector: dim: 1
      9.32
# MSPE ([a]-[b]+[c]):
Matrix: 1 by 1
row 0:          0

# kriging weights:
Matrix: 6 by 1
row 0:          1
row 1:          0
row 2:          0
row 3:          0
row 4:          0
row 5:          0

```

```

      coordinates var1.pred var1.var
1 (2.386, 3.077)      9.32      0

```

Q30 : What is the OK prediction at the known point? How does it compare with the original value? [Jump to A30](#) •

Q31 : What is the OK prediction variance at this point? [Jump to A31](#) •

Task 47 : Clean up from this section. •

```
> rm(jura.pred.6, jura.pt, k.pt)
```

7.2 Answers

A28 : The bounding box is 2.386 to 4.383 (X) and 1.081 to 4.519 (Y); this is considerably smaller than the bounding box for the whole sample. [Return to Q28](#) •

A29 : Point 2 gets over 1/3 of the total weight, since (1) it is fairly close to the prediction point; (2) it is the only point in the SW sector; (3) points 1 and 3 are also close to the prediction point but are clustered and hence “share” the weight for the NW sector. Interestingly, points 4 and 5 have the same weight although point 4 is closer to the prediction point. This is likely because point 4 is also closer to point 2. [Return to Q29](#) •

A30 : The prediction is 9.32, exactly the same as this calibration point. [Return to Q30](#) •

A31 : The prediction variance is 0. [Return to Q31](#) •

8 Self-test

This section is a small self-test of how well you mastered this exercise. You should be able to complete the tasks and answer the questions with the knowledge you have gained from the exercise. Please **submit your answers (including graphical output) to the instructor** for grading and sample answers.

For this self-test we continue with the Jura dataset, which should already be loaded from the exercise.

We begin with a closer look at a portion of the study area.

Task 1 : Make a prediction grid with 10 m resolution of the square km bounded by [3 ...4] East and [3 ...4] North¹. Use the corner points as centres of the grid (i.e. grid points fall on the even km lines). •

Task 2 : Predict the Cobalt concentration over this grid with Ordinary Kriging, using the variogram model previously determined and the Jura sample set. Plot the predictions and their variances. •

Q1 : *What are the minimum, mean and maximum predictions? How do these compare with the values for the rectangle covering the entire study area (see above)? Explain any differences.* •

Q2 : *What are the minimum, mean and maximum kriging variances? How do these compare with the values for the rectangle covering the entire study area (see above)? Explain any differences.* •

Task 3 : Remove temporary objects from the workspace. •

Now we pose some theoretical questions, to test your understanding of trend surfaces, variograms, and kriging. The following questions require careful thought. They are about the general situation, i.e., in any area, not only about our test area.

Q3 : *For predicting by a **trend surface**:*

(1) *In general, would there be a different **trend surface equation** using (a) all the calibration sample points or (b) just the calibration samples from this square to fit the trend surface?*

(2) *If the aim is to predict in this 1 km² area, under what assumptions would you prefer to use all the calibration sample points, and under what assumptions would you prefer to use just the calibration samples from this square to fit the trend surface to be used for that prediction?* •

Q4 : *For predicting by a **Ordinary Kriging**:*

(1) *In general, would there be a different **model of spatial dependence** (i.e. variogram model) using (a) all the calibration sample points or (b) just the calibration samples from this in this 1 km² area to fit the model of spatial dependence?*

(2) *What is a practical reason to use all the calibration sample points, rather than just those in the square, to model the variogram?*

¹ i.e. lower-left corner $(E,N) = (3,3)$, upper-right corner $(E,N) = (4,4)$

(3) What assumption must be made in order to use all the calibration sample points, rather than just the calibration samples from this square, to fit the model of spatial dependence to be applied to predict in the square? •

Q5 : For predicting by a **Ordinary Kriging**; once we have a model of spatial dependence (i.e. variogram model):

(1) In general, would there be a different **prediction map** using the same variogram model to kriging from (a) all the calibration sample points or (b) just the calibration samples from this in this 1 km² area for prediction by OK?

(2) What assumption must be made in order to use all the calibration sample points, rather than just the calibration samples from this square, to predict in the 1 km² area by OK?

(3) What are the advantages of using all the calibration sample points, not just those in the square? •

9 Quitting R

We will use both the Meuse and Jura datasets again, so let's leave them in the workspace and save them in `.RData`.

Task 4 : Leave R, saving the workspace. •

`q()`

The `q` “quit” method without any argument saves all objects in the special file (normally hidden in Windows Explorer) `.RData`.

References

- [1] W N Venables, D M Smith, and R Development Core Team. *An Introduction to R; Notes on R: A Programming Environment for Data Analysis and Graphics*. R Foundation for Statistical Computing, Vienna, version 2.6.1 (2007-11-26) edition, 2007. URL <http://www.R-project.org>. 19

Index of R Concepts

: operator, 46
[] operator, 16, 46

add.to argument (vgm function), 41
alpha argument (variogram function), 32
anis argument (vgm function), 33
as, 53
as.numeric, 10
as.ordered, 12
auto.key=T graphics argument, 10

bbox (sp package), 21
by, 11, 13

c, 33
capture.output, 51
ceiling, 22
conf.level argument (t.test function), 14
coordinates (sp package), 8

data, 7
data.frame, 14
data.frame class, 12
debug.level function argument, 47

expand.grid, 23

fit.variogram (gstat package), 33, 43
fitted, 3
floor, 22
for operator, 21

gridded (sp package), 8
GridTopology (sp package), 23
groups graphics argument, 10
gstat package, 3, 7, 16, 20, 35, 45

heat.colors, 37

jura dataset, 7

key.space graphics argument, 9
krige (gstat package), 3–5, 16, 17, 35, 47

lattice package, 5, 9
levelplot (lattice package), 5, 9
lm, 3, 4, 10
load, 31

meuse dataset, 31

meuse.grid dataset, 35
meuse.riv dataset, 37
min, 20

newdata argument (predict.lm function), 4, 12
nlevels, 9

paste, 21
pch graphics argument, 9
Polygon (sp class), 37
Polygon (sp package), 37
Polygons (sp class), 37
Polygons (sp package), 37
predict, 11
predict.gstat (gstat package), 3, 47
predict.lm, 4, 12
print, 21

q, 57
qnorm, 19

range argument (vgm function), 33
require, 5
row.names, 14

sapply, 13, 14
save, 31
sd, 13
sp package, 3, 23, 31, 35, 37
sp.layout argument (spplot function), 37
SpatialGrid (sp package), 23
SpatialGridDataFrame class, 9
SpatialPixelsDataFrame (sp class), 8
SpatialPixelsDataFrame class, 12
SpatialPoints (sp class), 4, 53
SpatialPoints (sp package), 46
SpatialPointsDataFrame (sp class), 8
SpatialPolygons (sp class), 37
SpatialPolygons (sp package), 37
split, 13
spplot (sp package), 9, 24, 37
summary.lm, 10

t.test, 14
tol.hor argument (variogram function), 32
topo.colors, 18, 38

variogram (gstat package), 32

`vgm` (gstat package), 33, 41

`xyplot` (lattice package), 9