Applied geostatistics
Exercise 4: Predicting from point samples (Part 1)

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

June 26, 2010

Contents

1 Introduction **1**

2 Trend surfaces **1**

3 * Design-based prediction **3**
3.1 Answers 11

4 Ordinary kriging **12**
4.1 Variogram modelling 12
4.2 Prediction at known points 13
4.3 Confidence limits 15
4.4 Prediction on a regular grid 17
4.5 Answers 23

5 Saving the workspace **24**

6 * Insight into the OK system **25**
6.1 Answers 30

7 Self-test **30**

8 Quitting R **31**

References **32**

Version 1.92. Copyright © 2006-2008 ITC All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited. To adapt or translate please contact the author (http://www.itc.nl/personal/rossiter).

Index of R concepts 33

不怕慢，只怕站 (Bù pà màn, zhǐ pà zhàn)
"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. Use a trend surface to predict attriburtes over a region;
2. Predict attributes over a region using strata;
3. Construct a prediction grid over a region;
4. Predict attributes over a region using Ordinary Kriging.

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.

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

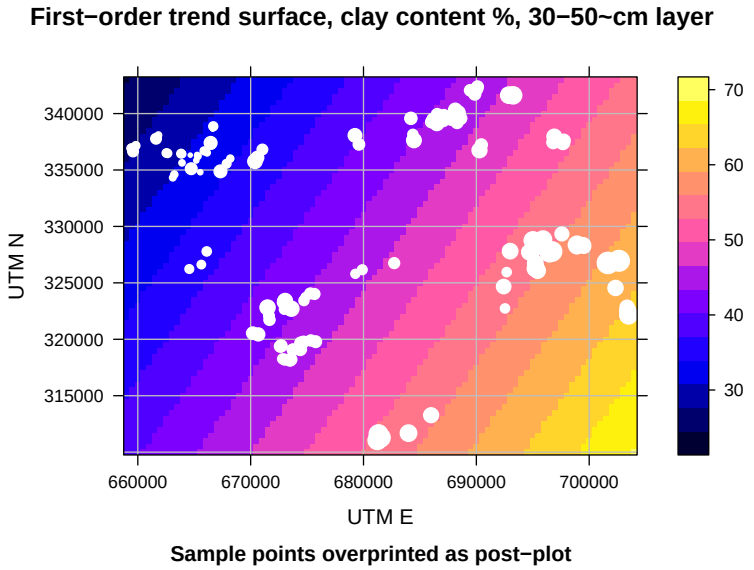
1. computing an interpolation grid;
2. computing a trend surface; and

3. 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  4.80 4.40 4.20 4.54 4.40 ...

> 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)
> str(gr)

'data.frame':        6097 obs. of  2 variables:
 $ UTM_E: num  659000 659500 660000 660500 661000 ...
 $ UTM_N: num  310000 310000 310000 310000 310000 310000 310000 310000 310000 310000 ...

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

[ordinary or weighted least squares prediction]
```

Now we use that trend surface to make a raster map of a study area:

```
> levelplot(var1.pred ~ UTM_E + UTM_N, as.data.frame(ts1),
+     col.regions=bpy.colors(64), asp="iso",
+    main="First-order trend surface, clay content %, 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=20, 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)
```

Task 4 : Indentify 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 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 10.00 10.60 11.92 16.32 ...
 $ Cr     : num  38.3 40.2 47.0 43.5 38.5 ...
 $ Cu     : num  25.72 24.76  8.88 22.70 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 90.0 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, specfying which fields represent the coordinates:

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

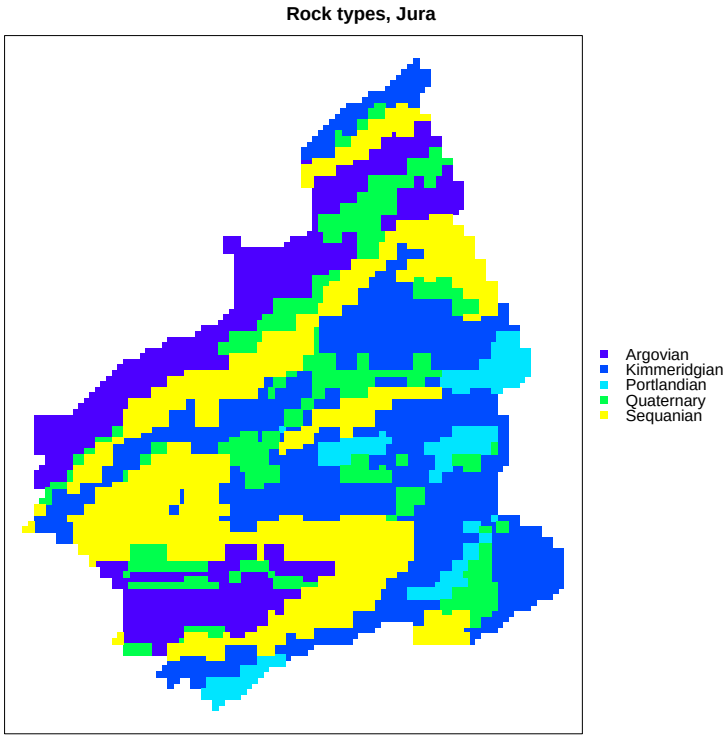
We will use **rock type** as a stratifying attribute for metal content. This is logical if the heavy metals in soils in this area are mainly from parent rock (and not from human activity).

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

One way to do this is with the `spplot` method. We specify a square filled printing character with the `pch` optional argument to simulate a raster display, and 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))`.

```
> spplot(jura.grid, zcol = "Rock", col.regions = topo.colors(nlevels(jura.grid$Rock)),
+     key.space = "right", pch = 15, main = "Rock types, Jura")
```



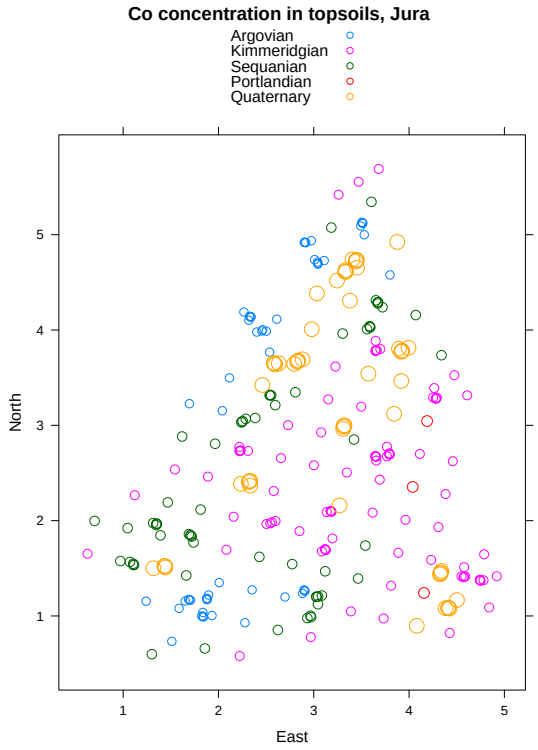
Note: The `spplot` method is a front-end for various methods in the `lattice` graphics package, such as `xyplot` and `levelplot`.

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

We use the `xyplot` method directly, 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
-9.4969 -1.5646  0.0861  1.8776  8.1241

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:

```
> head(predict(m.co.rt, jura.grid))

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

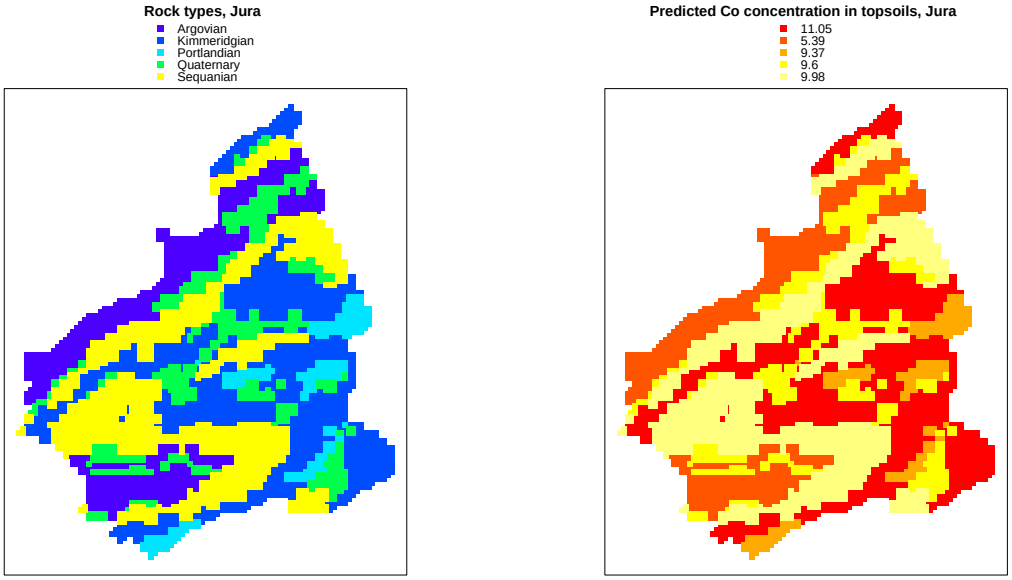
The difficulty here is that the ordered list of classes is not in sorted order of the predictions. For example, the second-listed class (`Kimmeridgian`) has the highest predicted concentration. So, we use the `as.ordered` method on the predicted values to sort the values in the proper order for a colour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

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

```
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':        5957 obs. of  2 variables:
  .. ..$ Rock: Factor w/ 5 levels "Argovian","Kimmeridgian",..: 3 3 3 2 2 2 3 3 1 1 ...
  .. ..$ Co  : Ord.factor w/ 5 levels "5.39"<"9.37"<..: 4 4 4 5 5 5 4 4 1 1 ...
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1:5957, 1:2] 0.3 0.35 0.35 0.4 0.4 0.4 0.4 0.4 0.4 0.4 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "Xloc" "Yloc"
  ..@ bbox       : num [1:2, 1:2] 0.3 0.1 5.1 5.9
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "Xloc" "Yloc"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

To emphasize that this is just a reclassification, we show the map of rock types and the reclassified map side-by-side, using the `split` and `more` arguments to the `print.trellis` method:

```
> plot.l <- spplot(jura.grid, zcol = "Rock", col.regions = topo.colors(nlevels(jura.grid$Rock)),
+     key.space = "top", pch = 15, main = "Rock types, Jura")
> plot.r <- spplot(co.reclass, zcol = "Co", col.regions = heat.colors(nlevels(jura.grid$Rock)),
+     key.space = "top", pch = 15, main = "Predicted Co concentration in topsoils, Jura")
> print(plot.l, split = c(1, 1, 2, 1), more = T)
> print(plot.r, split = c(2, 1, 2, 1), more = F)
> rm(plot.l, plot.r)
```



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.

Q7 : *What is the standard error of the prediction for each class?* *Jump to A7* •

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

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

jura.pred$Rock: Argovian
[1] 2.0673
------------------------------------------------
jura.pred$Rock: Kimmeridgian
[1] 2.8064
------------------------------------------------
jura.pred$Rock: Sequanian
[1] 2.1987
------------------------------------------------
jura.pred$Rock: Portlandian
[1] 2.4509
------------------------------------------------
jura.pred$Rock: Quaternary
[1] 4.3067
```

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:

```
> sapply(co.split, t.test)["conf.int", ]

$Argovian
[1] 4.8241 5.9637
attr(,"conf.level")
[1] 0.95

$Kimmeridgian
[1] 10.444 11.654
attr(,"conf.level")
[1] 0.95

$Sequanian
[1]  9.4215 10.5290
attr(,"conf.level")
[1] 0.95

$Portlandian
[1]  3.2849 15.4617
attr(,"conf.level")
[1] 0.95
```

```
$Quaternary
[1]  8.4316 10.7601
attr(,"conf.level")
[1] 0.95

> sapply(co.split, t.test, conf.level = 0.99)["conf.int",
+     ]

$Argovian
[1] 4.6346 6.1531
attr(,"conf.level")
[1] 0.99

$Kimmeridgian
[1] 10.247 11.851
attr(,"conf.level")
[1] 0.99

$Sequanian
[1]  9.239 10.711
attr(,"conf.level")
[1] 0.99

$Portlandian
[1] -4.6706 23.4173
attr(,"conf.level")
[1] 0.99

$Quaternary
[1]  8.0454 11.1464
attr(,"conf.level")
[1] 0.99
```

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. Note that the original data must be approximately normally distributed within each class for this to be valid.

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

Task 11 : 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 are also in the interpolation grid.* *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 class is:*

Argovian	*Kimmeridgian*	*Sequanian*	*Portlandian*	*Quaternary*
5.3939	*11.0489*	*9.9752*	*9.3733*	*9.5959*

Return to Q6 •

A7 : *The standard error of the prediction is:*

Argovian	*Kimmeridgian*	*Sequanian*	*Portlandian*	*Quaternary*
2.0673	*2.8064*	*2.1987*	*2.4509*	*4.3067*

Return to Q7 •

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

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

Task 13 : Divide the Jura dataset into a **calibration** set `jura.cal`, made up of the first 259 observations from the full set, and a **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 14 : 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 known points

Kriging can be used to predict at specific points; it does not have to be used to produce a regular grid. We illustrate this with the 100 observations which we held out as a **validation set**.

Task 15 : Use the fitted variogram model to predict at the 100 validation 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 validation 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
```

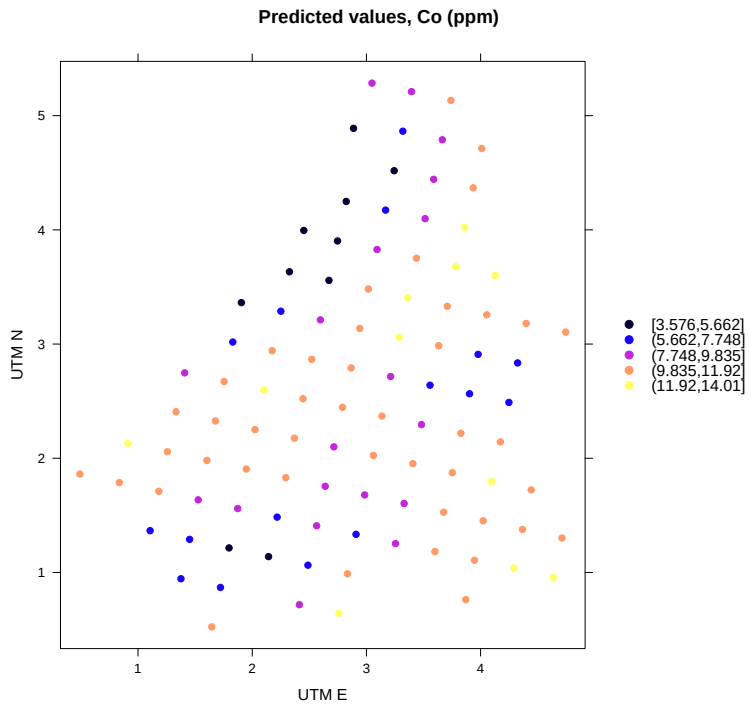
```
X 0.491 4.745
Y 0.524 5.285
Is projected: NA
proj4string : [NA]
Number of points: 100
Data attributes:
   var1.pred        var1.var
 Min.   : 3.58   Min.   :1.98
 1st Qu.: 7.70   1st Qu.:4.35
 Median :10.08   Median :4.84
 Mean   : 9.45   Mean   :4.71
 3rd Qu.:11.27   3rd Qu.:4.93
 Max.   :14.01   Max.   :7.92
```

Q8 : *What are the two data fields computed by* `krige`*? What are their units of measure?* *Jump to A8* •

Task 16 : Plot the predictions and their variances. •

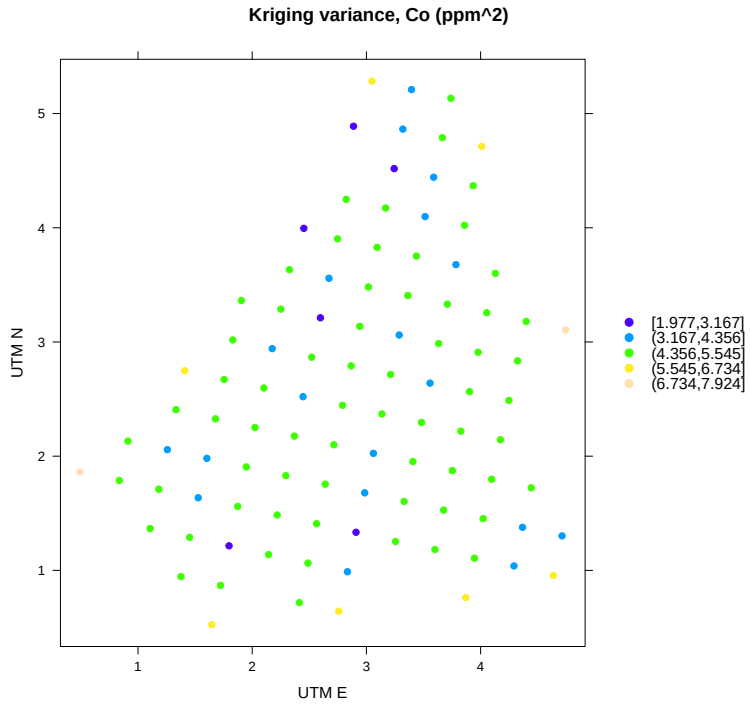
First the predictions:

```
> spplot(k.val, zcol = "var1.pred", key.space = "right",
+     col.regions = bpy.colors(64), 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.

```
> 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 assessement**.

Task 17 : Determine the 95% confidence limits for the first validation point. •

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

$$(\hat{z} - t_{\alpha/2,n-1} \cdot s_{\hat{z}}) \;\leq\; z \;\leq (\hat{z} + t_{\alpha/2,n-1} \cdot s_{\hat{z}})$$

where $\hat{z}$ is the estimated value, $t_{\alpha/2,n-1}$ is Student's t with $n-1$ degrees of freedom at confidence level $\alpha/2$ and $s_{\hat{z}}$ is the sample standard deviation; n is the sample size.

Note that 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. The t-distribution must be used because we are estimating both the mean and

variance from the same sample; for reasonably-large sample sizes the normal distribution itself can be used.

Q9 : *Which field in the kriging prediction object corresponds to the estimated value $\hat{z}$ and which to the sample standard deviation $s_{\hat{z}}$?* *Jump to A9* •

To evaluate this formula, we need to find quantiles of the t-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 `qt` ("**q**uantile of the **t**-distribution) method, specifying the quantile (here, the 97.5%, because the test is two-tailed) and the degrees of freedom, which is one less than the number of samples on which the estimate is based (**not** the number of points to be predicted).

```
> (n <- length(jura.cal$Co))

[1] 259

> (t.val <- qt(0.975, n - 1))

[1] 1.9692

> qnorm(0.975)

[1] 1.9600
```

Because of the large sample size, the t-value is effectively the quantile of the normal distribution, which we displayed with the `qnorm` ("**q**uantiles of the **normal** distribution") method. 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:

```
> sqrt(k.val$var1.var[1])

[1] 1.878

> (ciw <- t.val * sqrt(k.val$var1.var[1]))

[1] 3.6982
```

Finally we can compute the interval:

```
> k.val$var1.pred[1]

[1] 5.0495

> k.val$var1.pred[1] - ciw

[1] 1.3513

> k.val$var1.pred[1] + ciw

[1] 8.7476
```

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

Q10 : *What is the **maximum** cobalt concentration at the first validation point, with only a probability of 2.5% chance that this value is exceeded?* *Jump to A10* •

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(n, t.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); 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 18 : 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':        261 obs. of  6 variables:
 $ np     : num  1310749  983075 1769509  983085 2293805 ...
 $ dist   : num  0.00640 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 65536
```

```
> min(vc$dist)
```

```
[1] 0.005
```

```
> rm(vc)
```

Q11 : *How many point-pairs are separated by 100 m or less? What is the closest spacing?* *Jump to A11* •

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 19 : 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.

Q12 : *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 A12* •

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 20 : 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
```

Q13 : *How many points are in this prediction grid? What is its bounding box?* *Jump to A13* •

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

Formal class 'SpatialGrid' [package "sp"] with 5 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
  ..@ grid.index : int(0)
  ..@ coords     : num [1:2, 1:2] 0.325 5.075 0.425 5.725
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  ..@ bbox       : num [1:2, 1:2] 0.3 0.4 5.1 5.75
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  .. .. ..$ : 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)
```

Unfortunately, at the time these notes were prepared (January 2007), the `krige` method of the `gstat` package was not able to predict on an object of class `SpatialGrid`, so that the `jura.grid` object must be converted to a related spatial class, namely `SpatialPixels`, in order to continue.

We convert with the `SpatialPoints` spatial method, and add some efficiency to the kriging calculation by specifying that the prediction points are on a regular grid, with the `gridded` method; this converts the `SpatialPoints` object to class `SpatialPixels`.

```
> jura.grid <- SpatialPoints(jura.grid)
> gridded(jura.grid) <- T
> str(jura.grid)

Formal class 'SpatialPixels' [package "sp"] with 5 slots
  ..@ grid       :Formal class 'GridTopology' [package "sp"] with 3 slots
  .. .. ..@ cellcentre.offset: Named num [1:2] 0.325 0.425
  .. .. .. ..- attr(*, "names")= chr [1:2] "s1" "s2"
  .. .. ..@ cellsize         : Named num [1:2] 0.05 0.05
  .. .. .. ..- attr(*, "names")= chr [1:2] "s1" "s2"
  .. .. ..@ cells.dim        : Named int [1:2] 96 107
  .. .. .. ..- attr(*, "names")= chr [1:2] "s1" "s2"
  ..@ grid.index : int [1:10272] 1 2 3 4 5 6 7 8 9 10 ...
  ..@ coords     : num [1:10272, 1:2] 0.325 0.375 0.425 0.475 0.525 0.575 0.625 0.675 0.725
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "s1" "s2"
  ..@ bbox       : num [1:2, 1:2] 0.3 0.4 5.1 5.75
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "s1" "s2"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

Now we are in a position to predict on this grid.

Task 21 : 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 validation points, above, but we specify the prediction grid instead as the `newdata`:

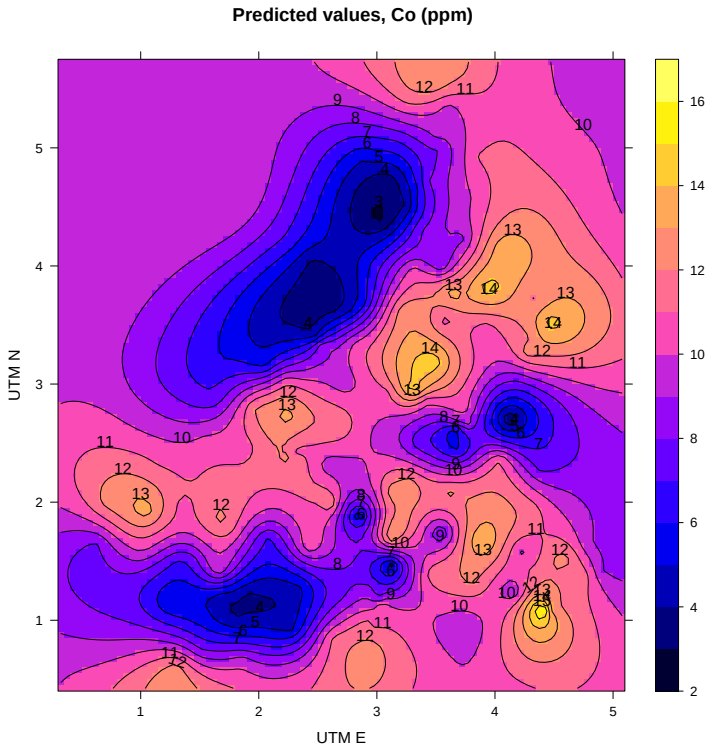
```
> k.grid <- krige(Co ~ 1, loc = jura.cal, newdata = jura.grid,
+     model = vmf)

[using ordinary kriging]

> summary(k.grid)

Object of class SpatialPixelsDataFrame
Coordinates:
   min  max
s1 0.3 5.10
s2 0.4 5.75
Is projected: NA
proj4string : [NA]
Number of points: 10272
Grid attributes:
   cellcentre.offset cellsize cells.dim
s1             0.325     0.05        96
s2             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 22 : 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)))
```

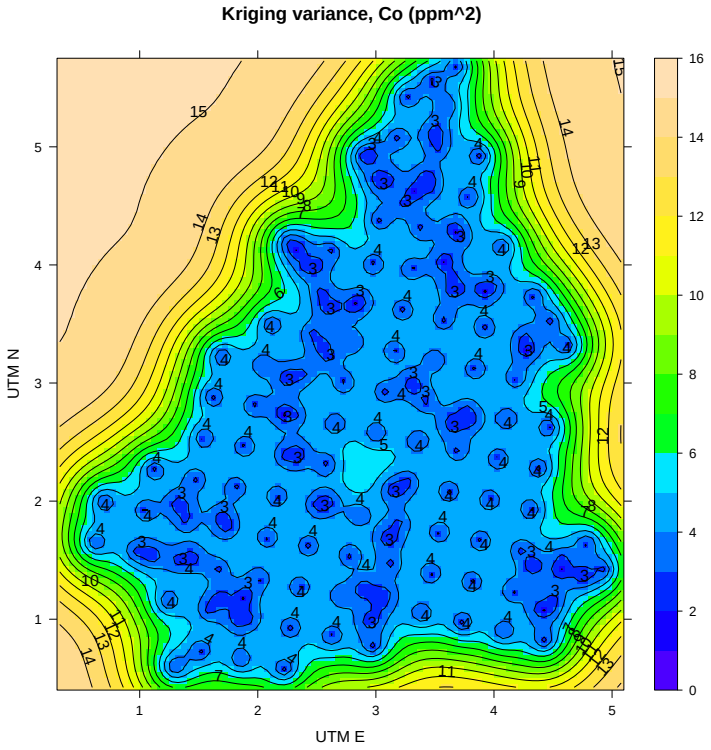


Q14 : *Describe the major spatial features of the Ordinary Kriging prediction.* *Jump to A14*

•

Now the kriging variances:

```
> 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))
```



Q15 : *Describe the major spatial features of the variances of the Ordinary Kriging prediction.* *Jump to A15* •

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. **validation**) in Exercise 6.

4.5 Answers

A8 : *There are two:*

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

Return to Q8 •

A9 : *Field* `var1.pred` *corresponds to the estimated value* $\hat{z}$*; the* ***square root*** *of field* `var1.var` *to the sample standard deviation* $s_{\hat{z}}$. *Return to Q9* •

A10 : 8.75 *mg kg*$^{-1}$ *Return to Q10* •

A11 : *261 point-pairs; closest spacing is 0.005 km = 5 m.* *Return to Q11* •

A12 : *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 Q12* •

A13 : 10368 *points, from* $X = (0.3, 5.1), Y = (0.4, 5.8)$ *Return to Q13* •

A14 : *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 Q14* •

A15 : *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 Q15* •

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.grid`
5. The fitted variogram model for Co, `vmf`
6. Predictions of Co at the validation 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 23 : 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.grid, vmf,
+     k.val, k.grid, file = "JuraEx4.RData")
```

These can then be loaded with the `load` method.

6 * 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 24 : Select the first six sample points of the prediction sample, and their Co concentration. •

```
> (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
```

Q16 : *What is the bounding box of these six points? How does that compare with the bounding box for the whole prediction sample?* *Jump to A16* •

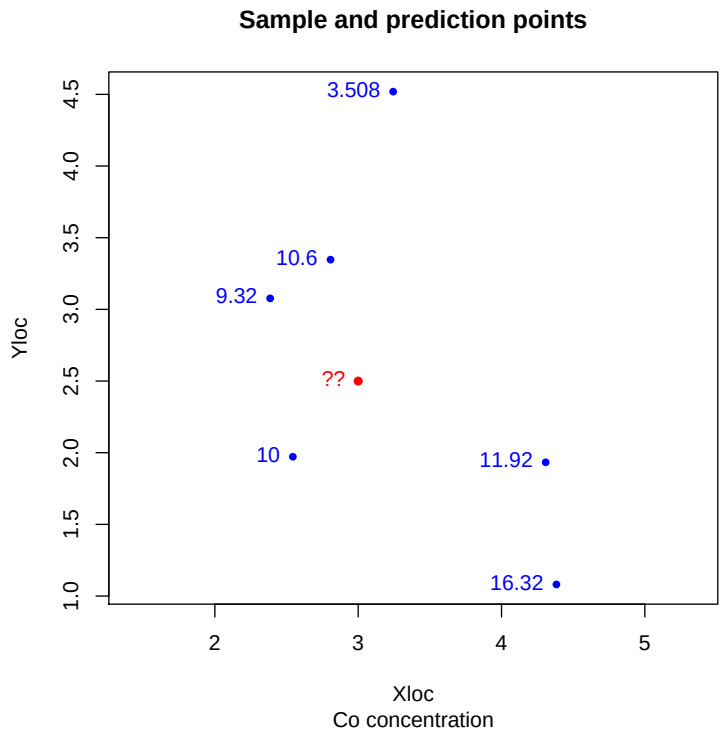
```
> bbox(jura.pred.6)

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

This location is choosen 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
  .. .. ..@ projargs: 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 26 : 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 `krige` 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
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     14.3033801
             0              0
row 4:              0              0              0     1.91089294
    14.3033801              0
row 5:              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; note that in
6. The **variance** of the spatial mean;
7. The **correlation** of the the spatial mean with itself, of course this is 1;
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 $C^{-1} C_0$;
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;

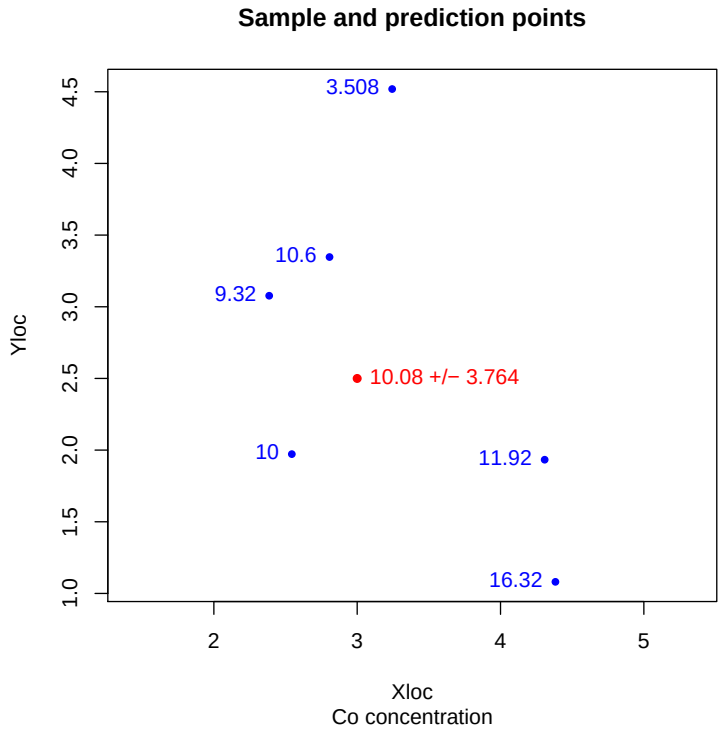
14. The **BLUP** at the prediction point; this is the **kriging prediction**;
15. The **kriging prediction variance** at the prediction point;
16. The **kriging weights**, i.e. the weights given to each sample point when their values are summed into the BLUP.

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 27 : 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)))
```



We will return to this example in Universal Kriging.

Task 28 : Clean up from this section. •

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

6.1 Answers

A16 : *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 Q16*
•

7 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] North[1]. Use the corner points as centres of the grid (i.e. grid points fall on the even km lines). •

The following questions require careful thought:

Q1 : *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 just the calibration samples from this square to fit the trend surface to be used for that prediction? •

Q2 : *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?*

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

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

(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? •

Q3 : *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 (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? •

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. •

Q4 : *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.* •

Q5 : *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. •

8 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. 16

Index of R Concepts

[] operator, 10

as.numeric, 5
as.ordered, 6
auto.key=T graphics argument, 5

bbox (package:sp), 15
by, 6, 7

ceiling, 16
coordinates (package:sp), 4

data, 3
debug.level method argument, 22

expand.grid, 17

fitted, 1
floor, 16
for operator, 16

gridded (package:sp), 18
GridTopology (package:sp), 17
groups graphics argument, 5
gstat package, 1, 3, 11, 14, 18, 21

jura dataset, 3

key.space graphics argument, 4
krige (package:gstat), 1, 11, 12, 18, 22

lattice package, 5
levelplot (package:lattice), 5
lm, 1, 5
load, 21

min, 14
more method argument, 7

nlevels, 4

paste, 16
pch graphics argument, 4
predict, 6
predict.gstat (package:gstat), 1, 22
print, 16
print.trellis (package:lattice), 7

q, 27
qnorm, 13
qt, 13

sapply, 7, 8
save, 21
sd, 7
sp package, 1, 17
SpatialGrid (package:sp), 17
SpatialPoints (package:sp), 18, 22
split, 7
split method argument, 7
spplot (package:sp), 4, 5, 19
summary.lm, 5

t.test, 8
topo.colors, 12

xyplot (package:lattice), 5