Geostatistics Tutorial

Ellie White

December 9th, 2016

Disclaimer: This tutorial was written by a HYD273 student. It will most definitely contain mistakes. Please let me know them at: white.elaheh@gmail.com
Credit: Hijmans, R. (2016) Interpolation [Source code] http://www.rspatial.org for cleaned up example data and starter code.

Contents

Libraries

library(sp) –for geostatistical analysis
library(rgdal) –for spatial data transformations
library(raster) –for spatial data manipulation and analysis
library(gstat) –for geostatistical analysis
library(dismo) –for k fold cross-validation
library(spMC) –for transition probability markov chains

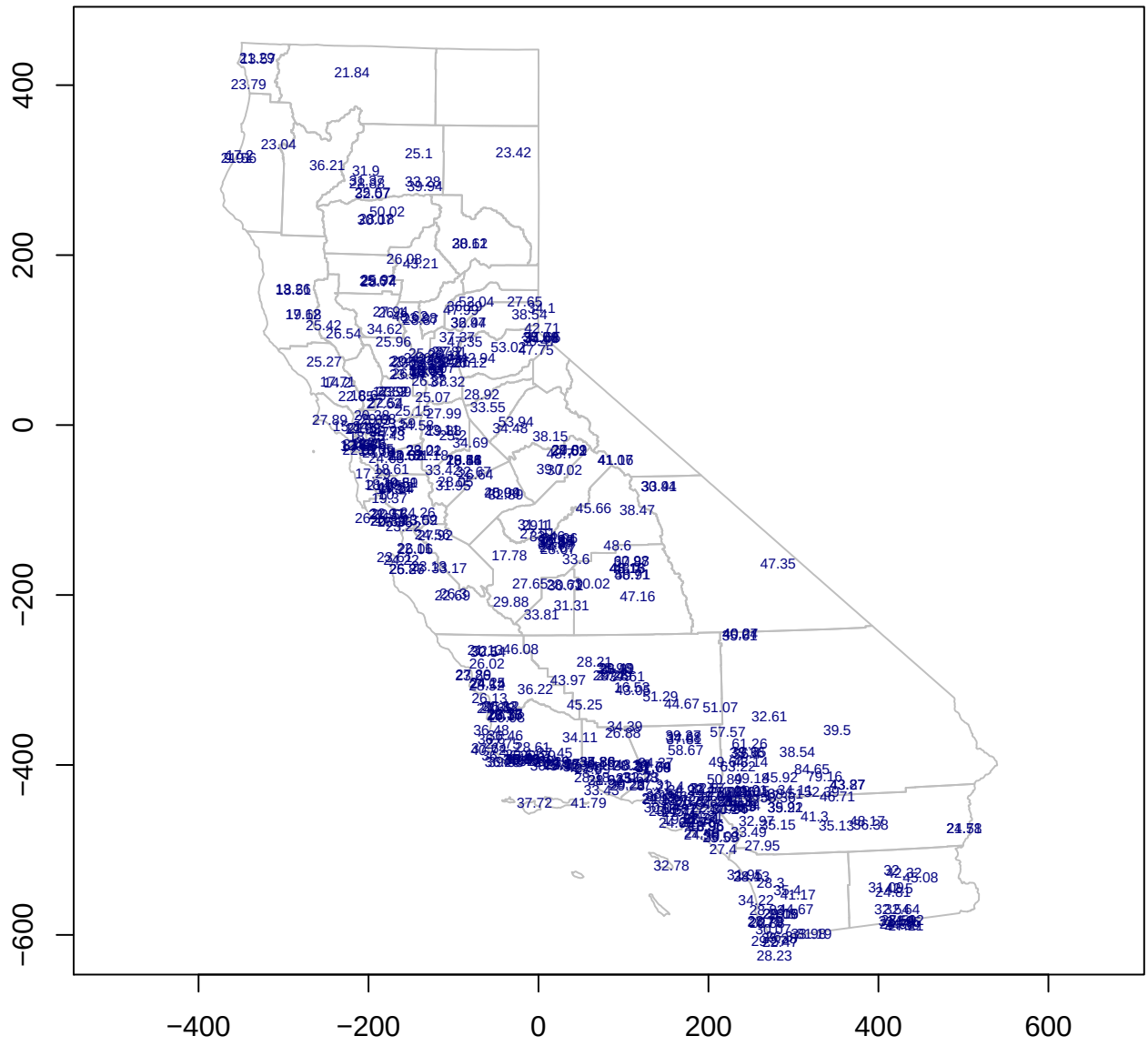
note: install each of these libraries by typing the following in your Console: install.packages("name of library")

1.0 Spatial Data

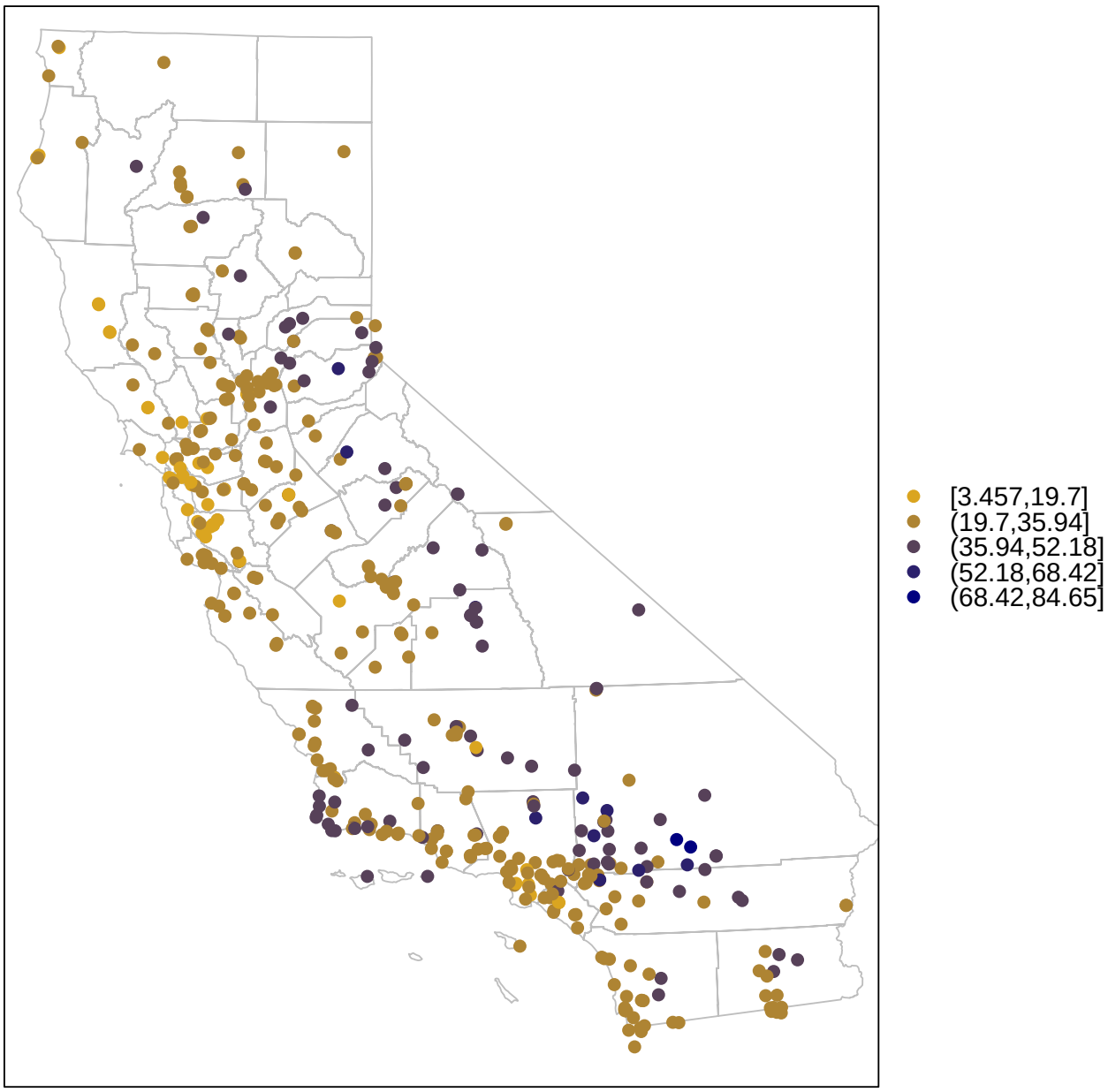
What: Ozone Concentrations
Units: ppb (parts per billion)
Time: averages for 1980-2009
Time resolution: 30 yr average
Data type: .csv
Data Source: https://www.arb.ca.gov/aqd/aqdcd/aqdcddld.htm, or download the clean version from: http://rspatial.org/analysis/data/airqual.csv **Provided by:** Cal EPA Air Resources Board

```
# download the clean version from: http://rspatial.org/analysis/data/airqual.csv, read in the data from
# directory
aq_data <- read.csv("airqual.csv")
head(aq_data)
##   LOCATION                   SITE_NAME   SHORT_NAME LATITUDE LONGITUDE
## 1     2001 Citrus Heights-Sunrise Blvd Citrus_Hghts  38.6988 -121.2711
## 2     2008            El Capitan Beach El_Capitan_B  34.4622 -120.0258
## 3     2011          Eureka-Fort Avenue Eureka-FtAve  40.8019 -124.1630
## 4     2012         Fresno-Cal State #2 Fresno-ClSt2  36.8136 -119.7402
## 5     2013      Fresno-Drummond Street Fresno-Drmnd  36.7055 -119.7413
## 6     2016     Carmel Valley-Ford Road Carm_Val-Frd  36.4819 -121.7333
##   CH4MAX1H CH4DLYAV  COMAX8N  COMXN8N  COMAX8O COMAX1HR  CODLYAVG
## 1 1.921959 1.699093 1.435813 1.424615 1.502238 2.448005 0.8281095
## 2       NA       NA       NA       NA       NA       NA        NA
## 3       NA       NA       NA       NA       NA       NA        NA
## 4       NA       NA 1.798955 1.781762 1.867940 2.732195 1.2297619
## 5       NA       NA 1.045182 1.038413 1.109875 1.570416 0.6951949
## 6       NA       NA       NA       NA       NA       NA        NA
##     H2SDLYAV NMHCMX1H  NMHCDAVG   NO2MAX1H    NO2DLYAV   NOMAX1HR
## 1         NA 0.737931 0.4078079 0.04155532 0.020119890 0.07982707
## 2 0.00068997       NA        NA 0.02178709 0.008927009 0.01551045
## 3         NA       NA        NA         NA          NA         NA
## 4         NA       NA        NA 0.03684358 0.018937586 0.05141533
## 5         NA       NA        NA 0.03827725 0.020317872 0.06137485
## 6         NA       NA        NA         NA          NA         NA
##      NODLYAV   NOXMAX1H   NOXDLYAV   OZMAX1HR  OZHMX1HR    OZMAX8O
## 1 0.02124207 0.11369346 0.04125031 0.05146295 11.842727 0.04045972
## 2 0.00307046 0.03516772 0.01202962 0.05001702 12.249517 0.04325032
## 3         NA         NA         NA 0.02632076  8.292453 0.02252830
## 4 0.01290800 0.08252951 0.03215824 0.06849787 11.937843 0.05609775
## 5 0.01572324 0.09271135 0.03604109 0.05656658 13.446605 0.04689867
## 6         NA         NA         NA 0.04190405 11.893539 0.03662570
##    OZH1MX8O    OZDLYAV    OZMX8ST  OZHMX8ST    SO2MAX1H    SO2MX24N
## 1  9.893424 0.02493750 0.04092387  9.893635 0.003246073 0.000802711
## 2  9.456927 0.03137524 0.04372468  9.466335 0.001017097 0.000279020
## 3 10.424528 0.01720210 0.02283962 10.396226 0.002083333 0.000188095
## 4 10.204918 0.03664366 0.05655859 10.201579          NA          NA
## 5 10.256402 0.02807397 0.04739147 10.249258          NA          NA
## 6  9.598142 0.02686707 0.03708901  9.597224          NA          NA
##      SO2MX24O   SO2DLYAV    SULFURMX THCMAX1H THCDLYAV    TRSMAX1H
## 1 0.001077212 0.00075267          NA 2.605642 2.113489          NA
## 2 0.000362556 0.00027886          NA 2.079051 1.831332 0.000640526
## 3 0.000306316 0.00021500 0.003173077 2.372549 2.023318          NA
## 4          NA         NA          NA       NA       NA          NA
```

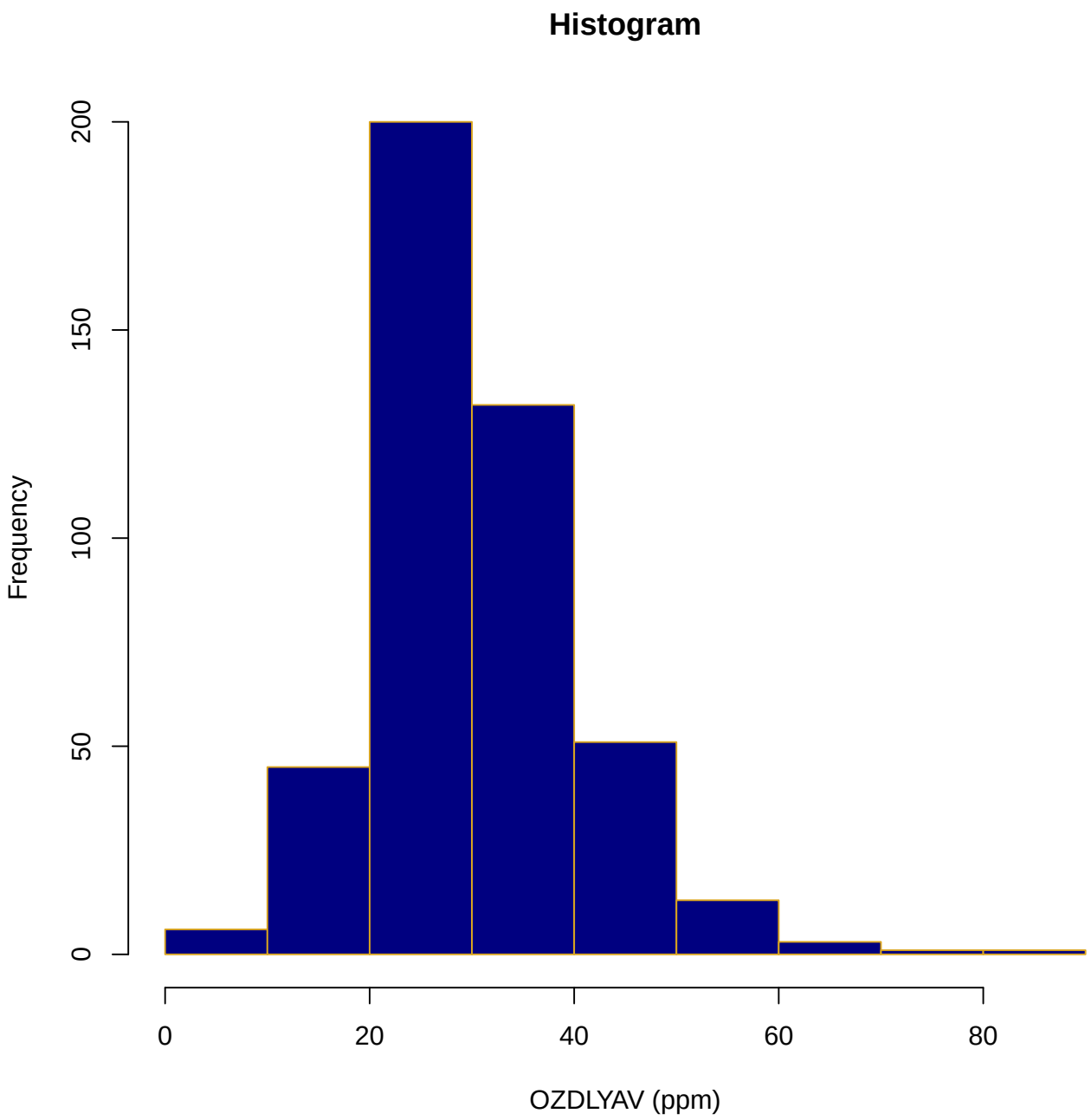
```
## 5          NA         NA          NA       NA       NA          NA
## 6          NA         NA          NA       NA       NA          NA
##    DAYOFWK
## 1 4.001256
## 2 4.002197
## 3 3.990654
## 4 3.997995
## 5 4.001840
## 6 4.000612

# change the OZDLYAV column to ppm (parts per million) so it is easier to read. This is the column of d
# working with
aq_data$OZDLYAV <- aq_data$OZDLYAV * 1000
aqd <- aq_data

# change data frame to a spatial points data frame
library(sp)
coordinates(aqd) <- ~LONGITUDE + LATITUDE
proj4string(aqd) <- CRS("+proj=longlat +datum=NAD83")

# transform data into a suitable projection, for California: Albers and Teale Albers are good
TA <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000 +datum=NAD83 +units

library(rgdal)
## rgdal: version: 1.0-4, (SVN revision 548)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 1.11.2, released 2015/02/10
##  Path to GDAL shared files: /Library/Frameworks/R.framework/Versions/3.2/Resources/library/rgdal/gdal
##  Loaded PROJ.4 runtime: Rel. 4.9.1, 04 March 2015, [PJ_VERSION: 491]
##  Path to PROJ.4 shared files: /Library/Frameworks/R.framework/Versions/3.2/Resources/library/rgdal/pr
##  Linking to sp version: 1.1-1
aq_ta <- spTransform(aqd, TA)

# for visualization purposes download California counties from link in:
# http://rspatial.org/analysis/rst/4-interpolation.html
library(raster)
ca <- shapefile("Counties/counties_2000.shp")
ca_ta <- spTransform(ca, TA)

# visualization of the data
plot(ca_ta, border = "gray", axes = TRUE, main = "Ozone Concentration (ppm)")
text(aq_ta, labels = round(aq_ta$OZDLYAV, 2), cex = 0.5, col = "navy")
```



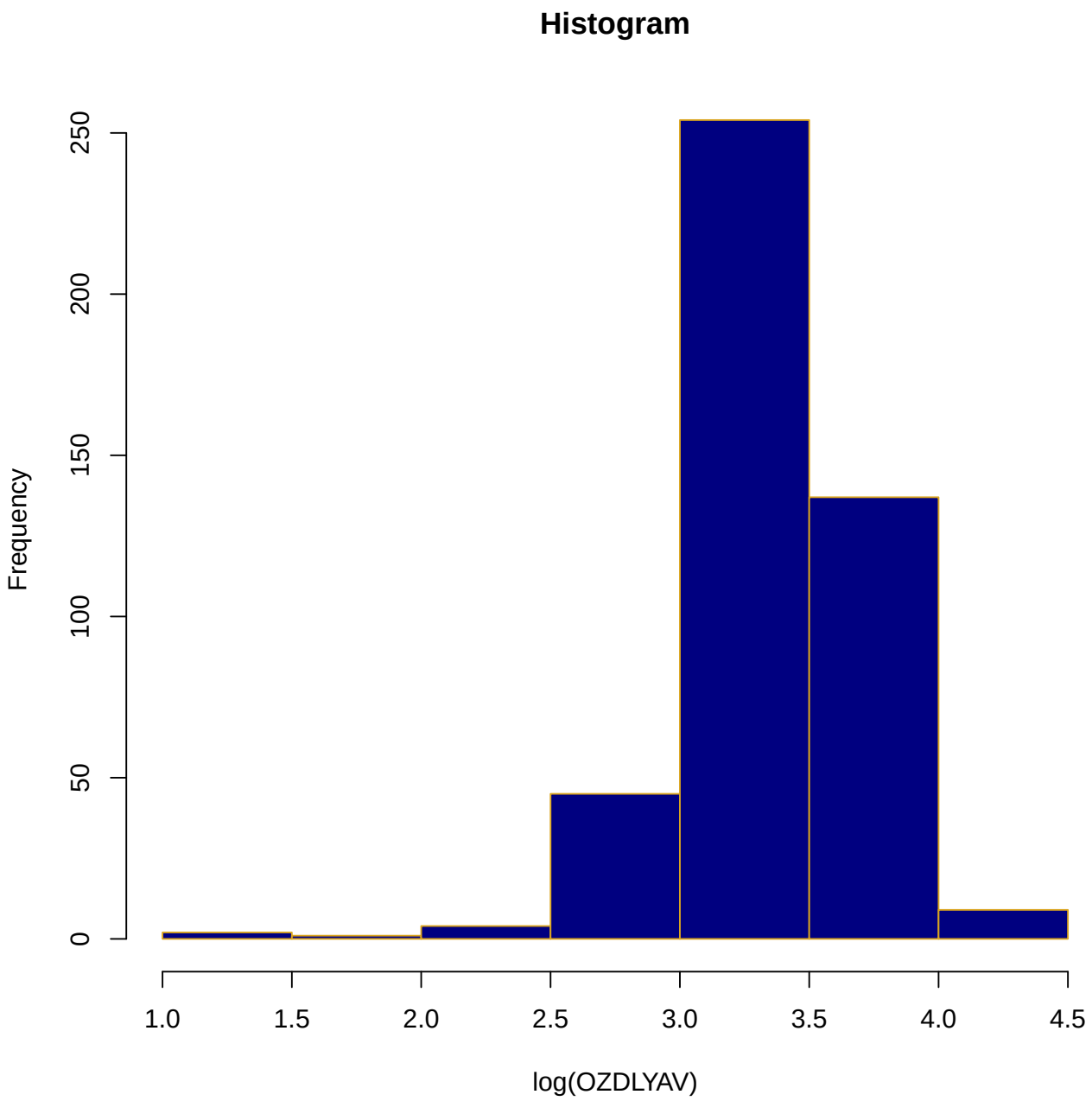

```
ucd_colors <- colorRampPalette(c("goldenrod", "navy"))
spplot(aq_ta, "OZDLYAV", col.regions = ucd_colors(6), key.space = "right", main = "Ozone Concentration
    ca_ta, col = "grey"))
```




```
hist(aq_data$OZDLYAV, main = "Histogram", col = "navy", border = "goldenrod", xlab = "OZDLYAV (ppm)")
```




```
hist(log(aq_data$OZDLYAV), main = "Histogram", col = "navy", border = "goldenrod", xlab = "log(OZDLYAV)
```



2.0 Blank Interpolation Zone

This section creates a spatial grid over california (projected to Teale Albers) to interpolate or make predictions to, i.e. a blank slate to be later given values with each model developed.

```
r <- raster(ca_ta)
res(r) <- 10  # 10 km if your Coordinate Reference System's units are in km, the Teal Albers system is

# coerce r into a spatial grid, not very different from a raster
g <- as(r, "SpatialGrid")
```

3.0 Evaluation Metrics

This section defines RMSE, R Squared, and NSE as functions to be used when evaluating and comparing models.

```
RMSE <- function(observed, predicted) {
    sqrt(mean((predicted - observed)^2, na.rm = TRUE))
}

R2 <- function(observed, predicted) {
    sum((predicted - mean(observed, na.rm = TRUE))^2, na.rm = TRUE)/sum((observed - mean(observed, na.r
}

# use this if you have hydrologic data, I have left it out of this analysis
NSE <- function(observed, predicted) {
    1 - (sum((predicted - observed)^2, na.rm = TRUE)/sum((mean(observed) - observed)^2, na.rm = TRUE))
}
```

4.0 The Average Model

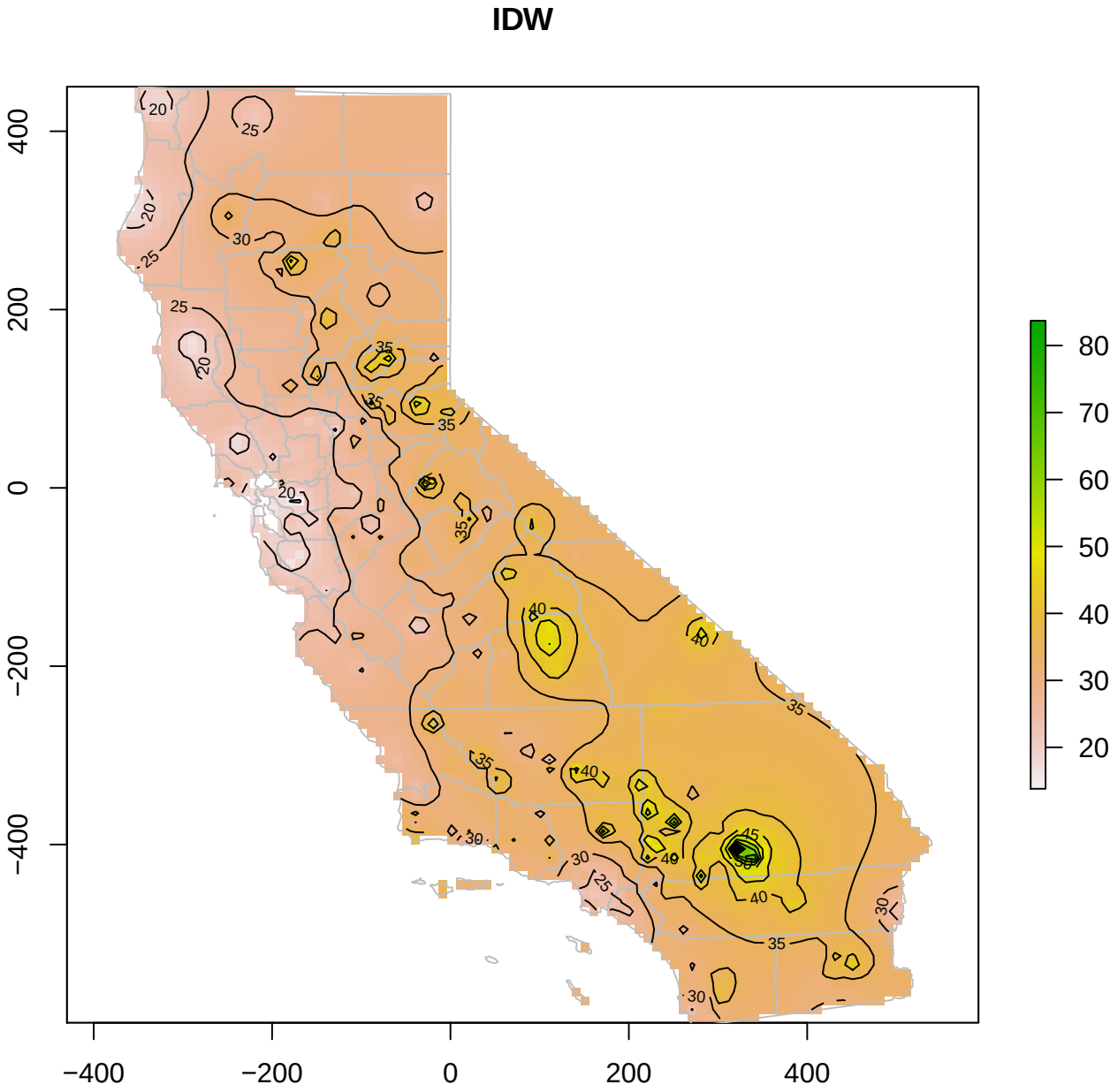
This model is the arithmatic average of all the observations.

```
am <- mean(aqd$OZDLYAV)

# make a raster that has this value everywhere, we may need this for comparison to other models later.
am_raster <- r
values(am_raster) <- am

# evaluation of this model
am_rmse <- RMSE(am, aq_ta$OZDLYAV)
am_rmse
## [1] 10.20169
```

5.0 Inverse Distance Weighted Model

This model is a weighted combination of all observations. The weights will be purely based on distance.

```
library(gstat)
# gstat objects hold all the information necessary for univariate or multivariate geostatistical predic
gs <- gstat(formula = OZDLYAV ~ 1, locations = aq_ta)

idw <- interpolate(r, gs)
## [inverse distance weighted interpolation]
idwr <- mask(idw, ca_ta)

# plot
plot(idwr, main = "IDW")
plot(ca_ta, border = "gray", add = TRUE)
contour(idwr, add = TRUE, nlevels = 10)
```

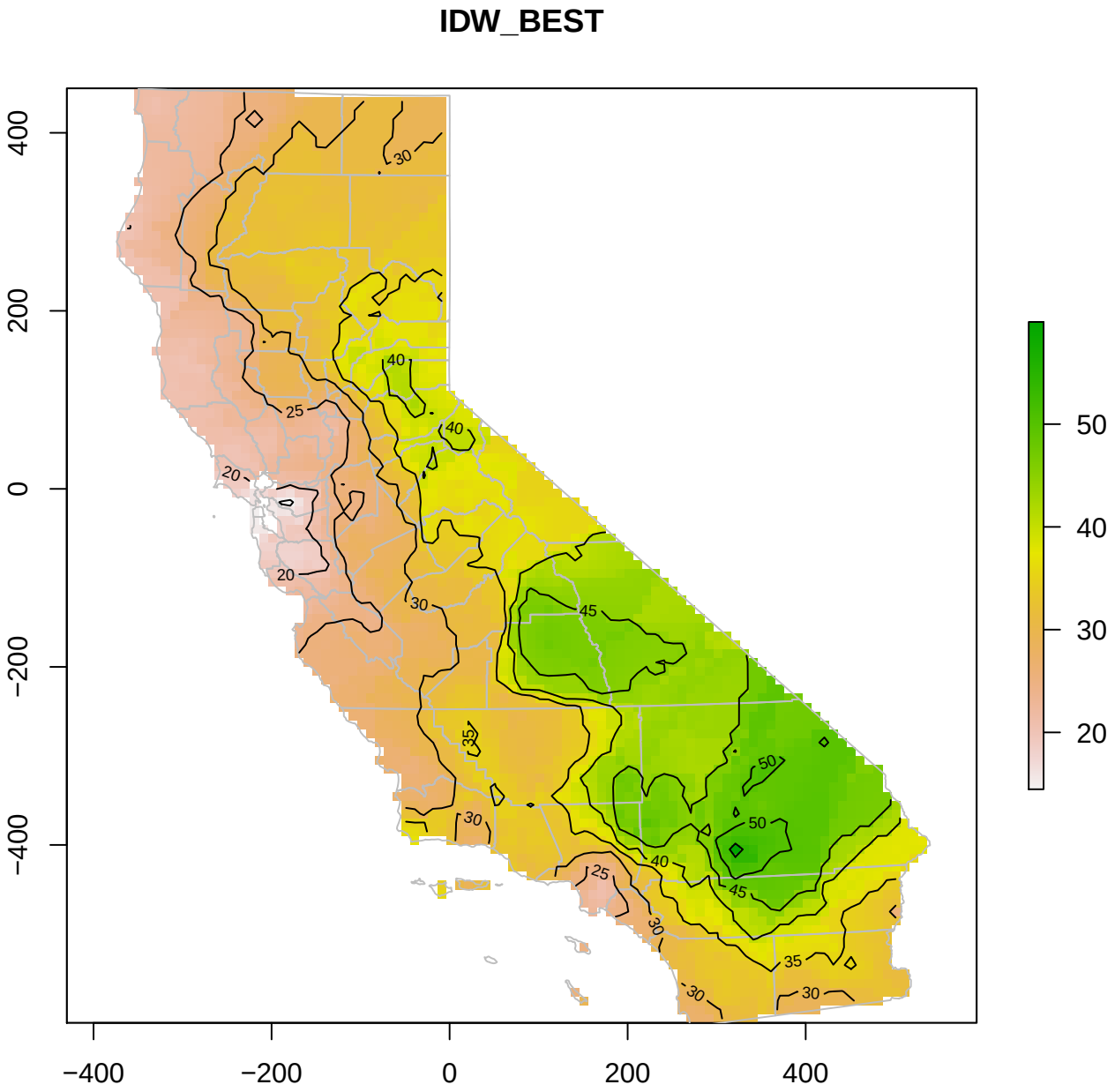


5.1 Optimal Inverse Distance Weighted Model

In the IDW model there are two parameters that can be optimized: 1) the number of nearest observations that should be used for a prediction, and 2) the distance decay rate.

```
# optimization objective: minimize the RMSE
f1 <- function(x, test, train) {
    nmx <- x[1]
    idp <- x[2]
    if (nmx < 1)
        return(Inf)
    if (idp < 0.001)
        return(Inf)
    m <- gstat(formula = OZDLYAV ~ 1, locations = train, nmax = nmx, set = list(idp = idp))
```

```
    p <- predict(m, newdata = test, debug.level = 0)$var1.pred
    RMSE(test$OZDLYAV, p)
}

# 5-fold cross validation
set.seed(20161209)
i <- sample(nrow(aq_ta), 0.2 * nrow(aq_ta))
tst <- aq_ta[i, ]
trn <- aq_ta[-i, ]

# optimization model
opt <- optim(c(8, 0.5), f1, test = tst, train = trn)
opt
## $par
## [1] 11.3759182  0.4271461
##
## $value
## [1] 6.676153
##
## $counts
## function gradient
##       51       NA
##
## $convergence
## [1] 0
##
## $message
## NULL

# optimal IDW model opt$par[1]: the number of nearest observations that should be used for a prediction
# distance decay rate. the power on the distance in the demoninator
idw_opt <- gstat(formula = OZDLYAV ~ 1, locations = aq_ta, nmax = opt$par[1], set = list(idp = opt$par[1
idw_b <- interpolate(r, idw_opt)
## [inverse distance weighted interpolation]
idw_best <- mask(idw_b, ca_ta)

# plot
plot(idw_best, main = "IDW_BEST")
plot(ca_ta, border = "gray", add = TRUE)
contour(idw_best, add = TRUE, nlevels = 10)
```

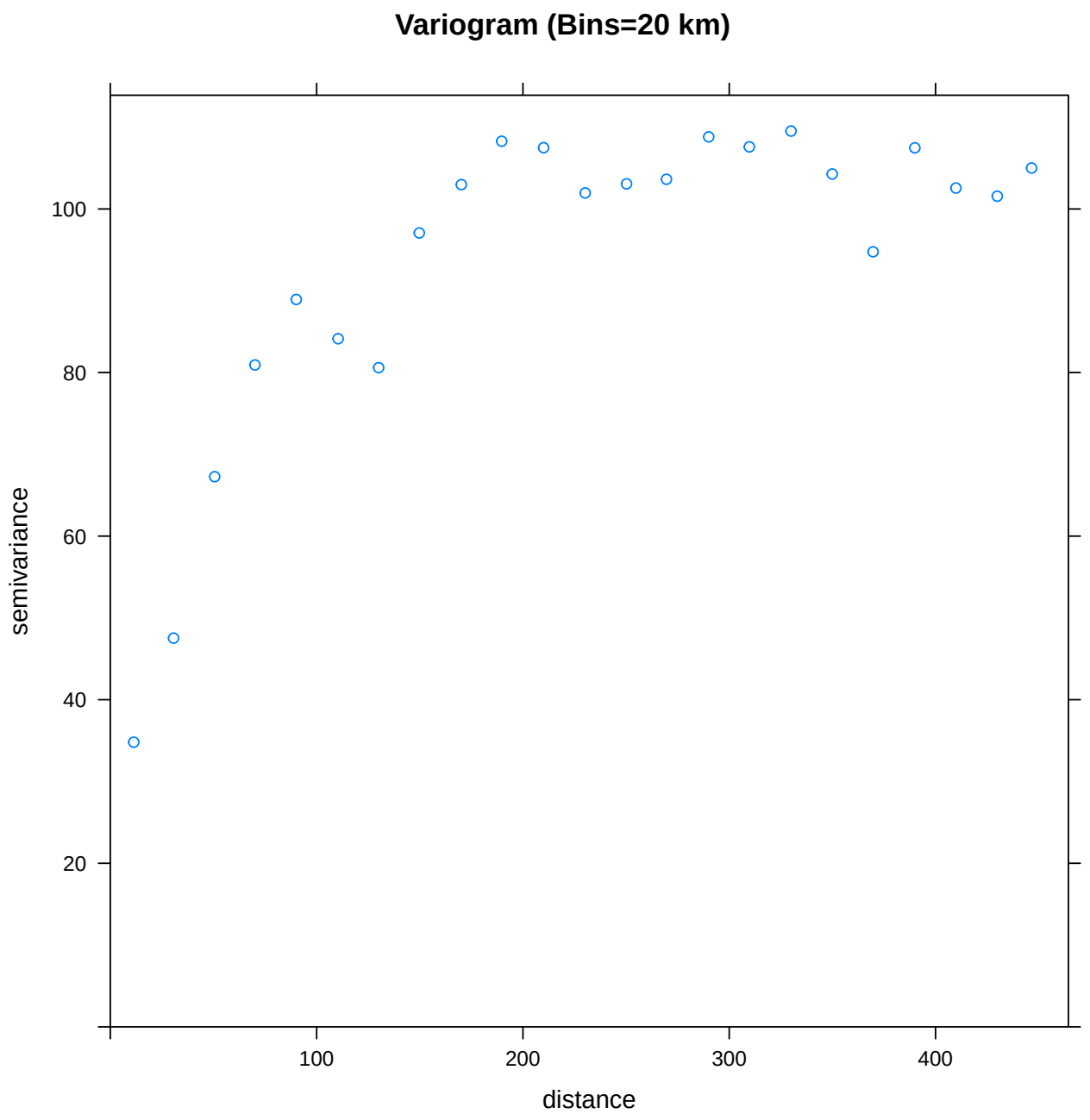


6.0 Variogram

A variogram is a description of the spatial continuity of the data. The experimental variogram is a function fitted to the variogram. This function measures the variability between pairs of points at various distances and is used in Kriging models.

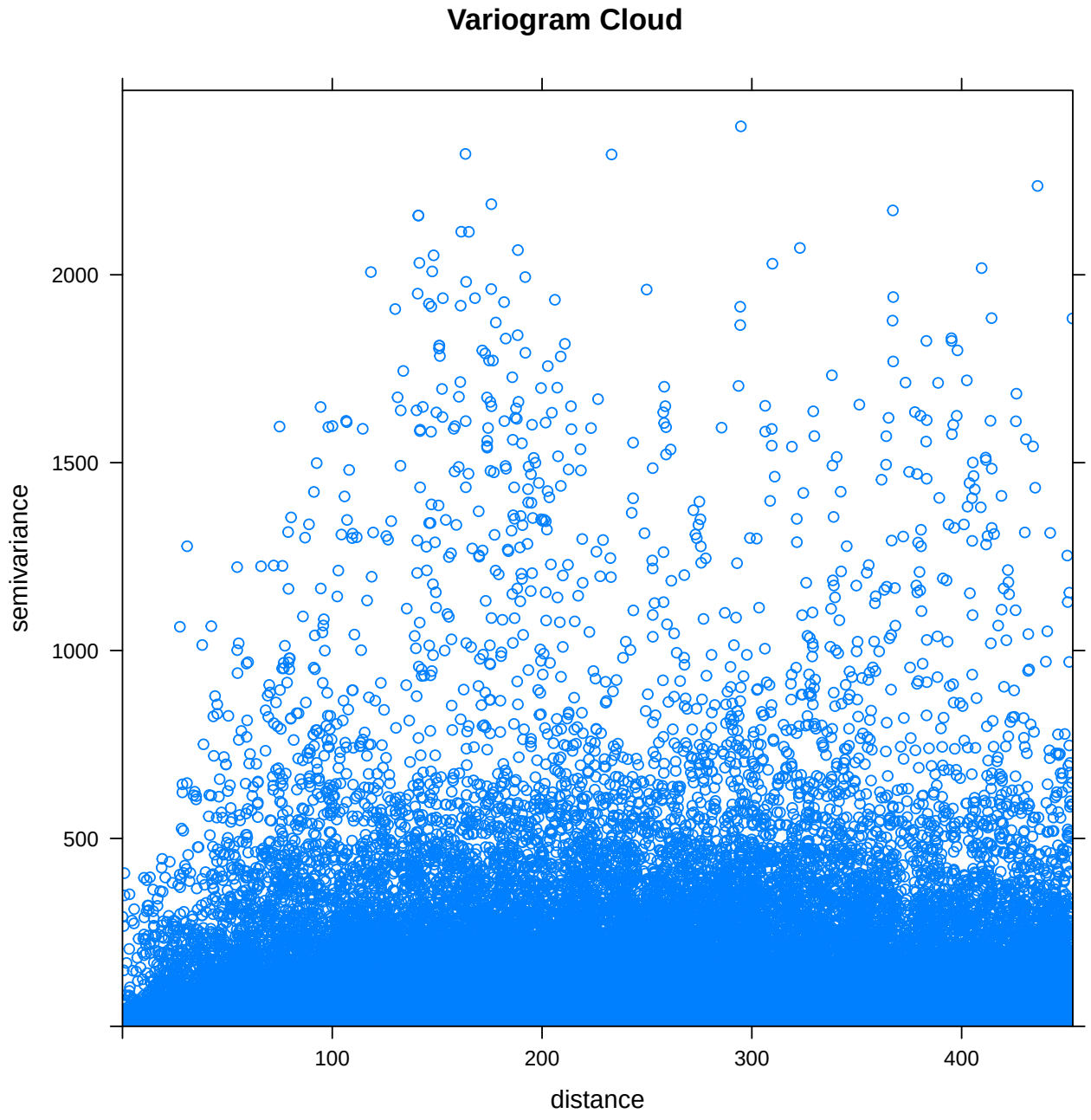
```
gs <- gstat(formula = OZDLYAV ~ 1, locations = aq_ta)
v <- variogram(gs, width = 20)  # width is the width of the bins for grouping data
head(v)
##     np      dist    gamma dir.hor dir.ver   id
## 1 1010  11.35040 34.80579       0       0 var1
## 2 1806  30.63737 47.52591       0       0 var1
## 3 2355  50.58656 67.26548       0       0 var1
```

```
## 4 2619  70.10411 80.92707       0       0 var1
## 5 2967  90.13917 88.93653       0       0 var1
## 6 3437 110.42302 84.13589       0       0 var1
plot(v, main = "Variogram (Bins=20 km)")
```

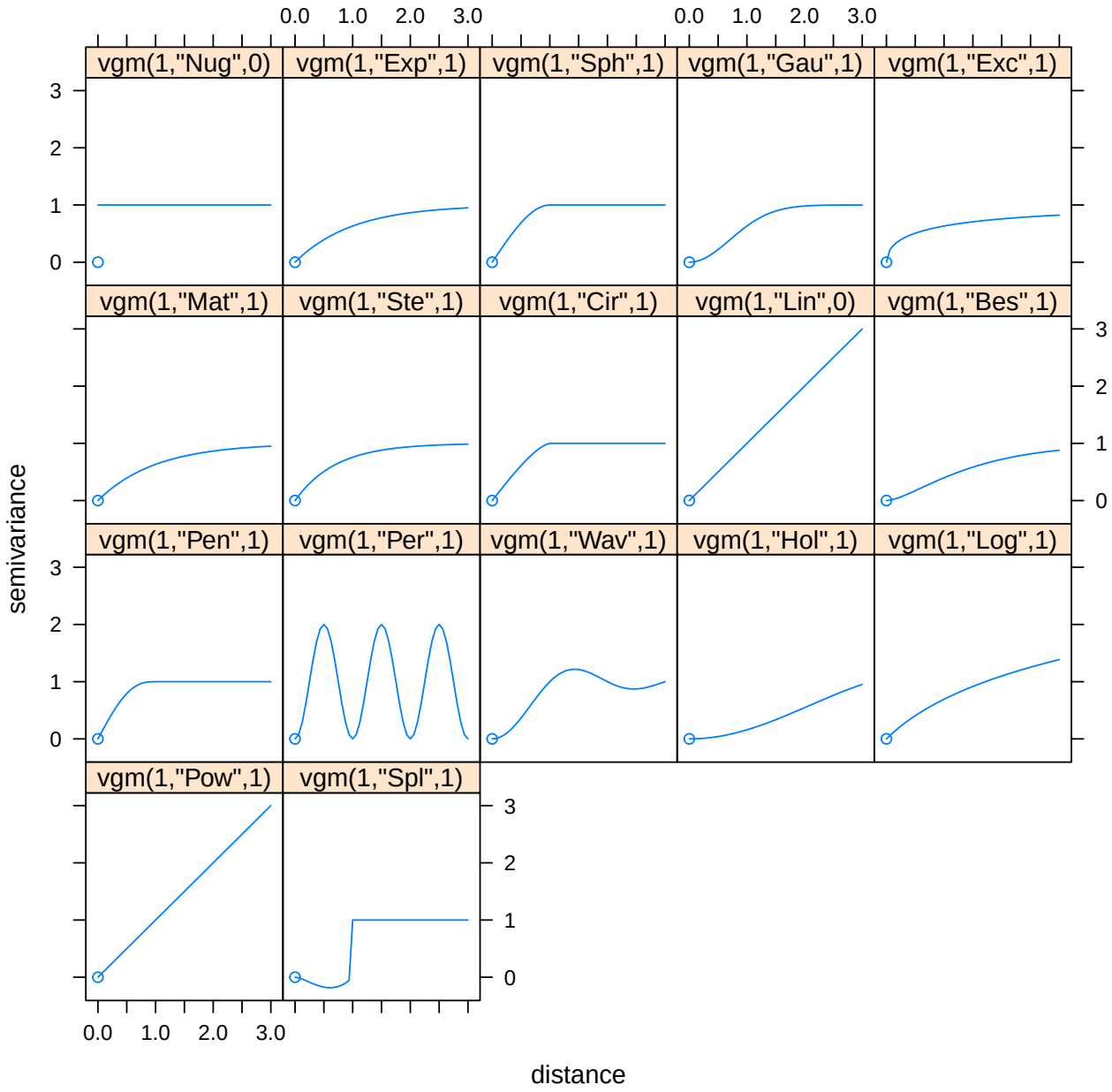



```
# set cloud=TRUE to see all the pairs of observations
vc <- variogram(gs, width = 20, cloud = TRUE)
head(vc)
##        dist     gamma dir.hor dir.ver   id left right
## 1 340.47167 29.918191       0       0 var1    3     1
## 2 249.10948 68.517044       0       0 var1    4     1
## 3 262.45958 13.878114       0       0 var1    4     2
```

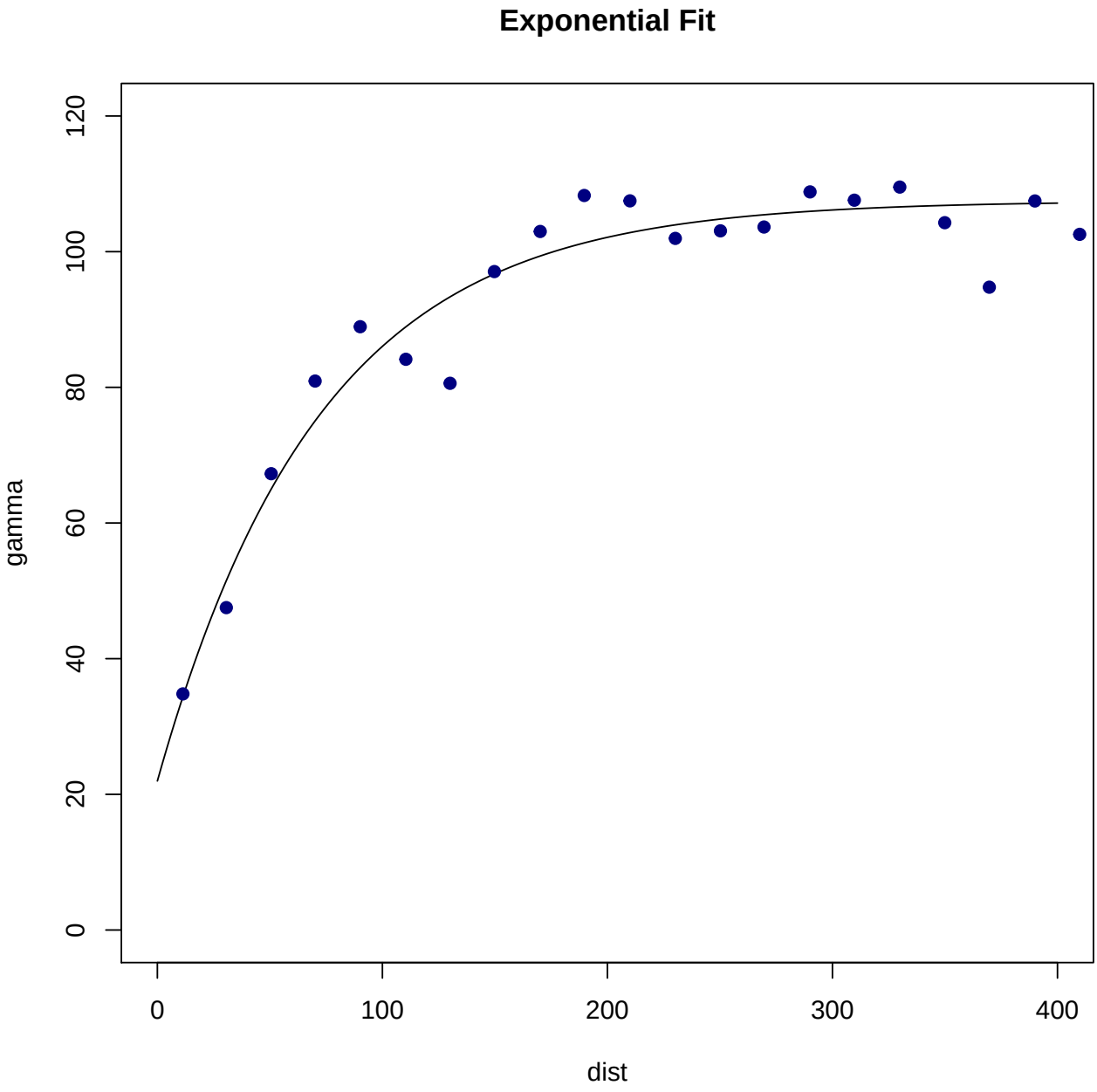
```
## 4 259.29774  4.918725       0       0 var1     5     1
## 5 250.49749  5.449182       0       0 var1     5     2
## 6  12.01527 36.719750       0       0 var1     5     4
plot(vc, main = "Variogram Cloud")
```



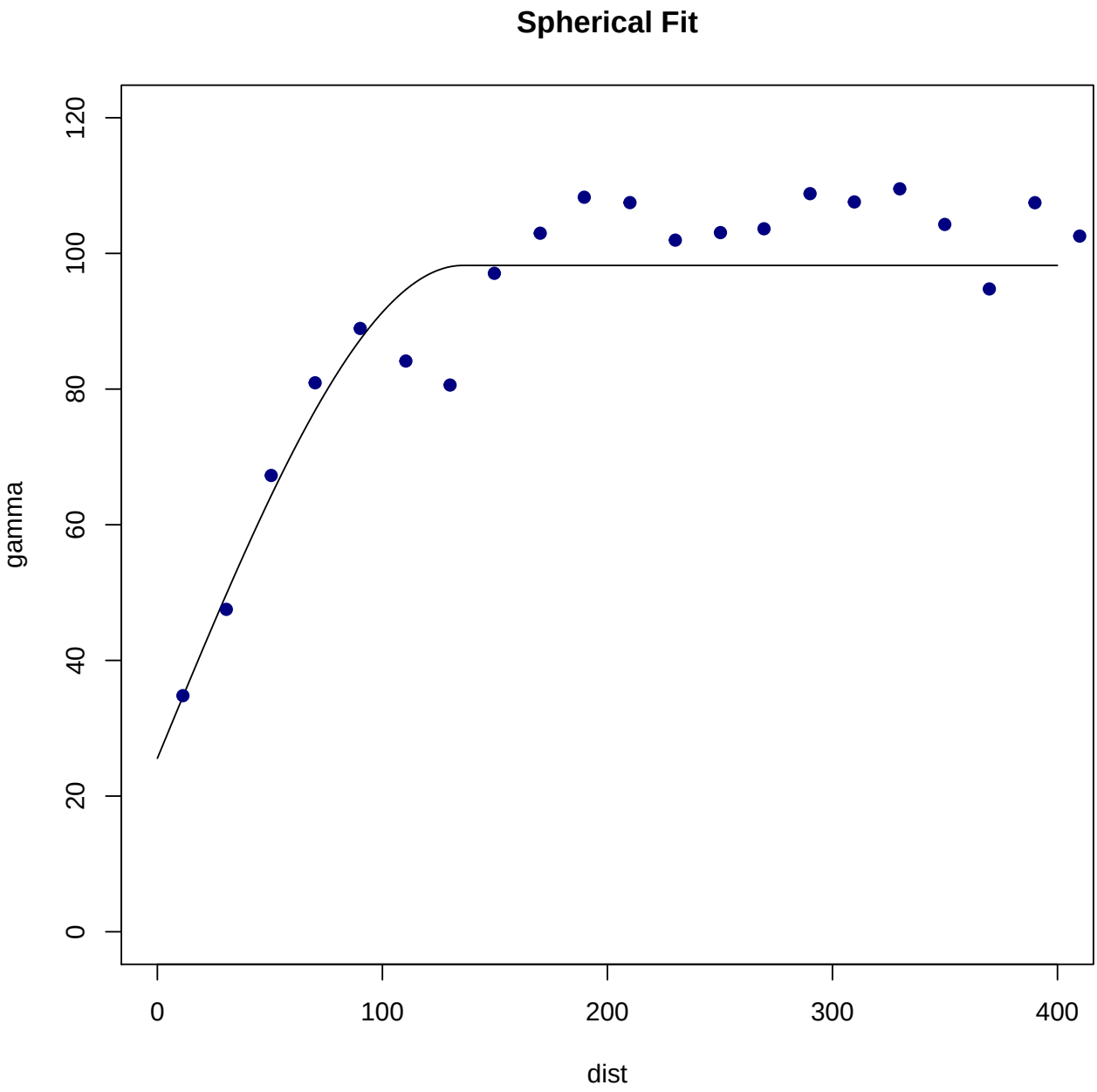

```
# now to fit a model to the variogram. Here are all the types of fits we can try
show.vgms()
```




```
# fit a variogram model, best way is to eye ball the sill, range, and nugget here is an exponential fit
fve <- fit.variogram(v, vgm(85, "Exp", 75, 20))
fve
##   model    psill    range
## 1   Nug 21.96600  0.00000
## 2   Exp 85.52957 72.31404
plot(variogramLine(fve, 400), type = "l", ylim = c(0, 120), main = "Exponential Fit")
points(v[, 2:3], pch = 20, col = "navy", cex = 1.5)
```



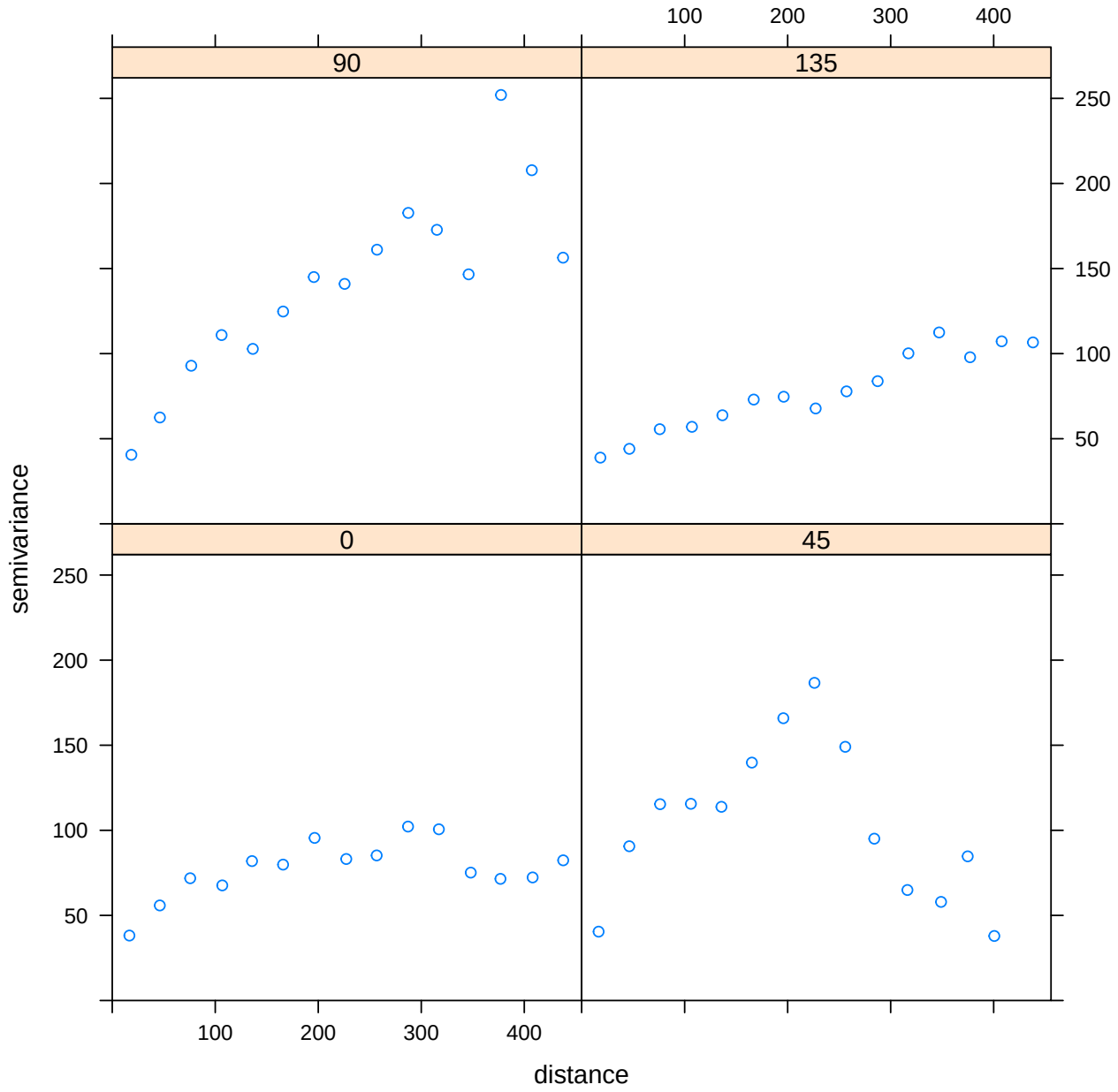

```
# here is a spherical fit
fvs <- fit.variogram(v, vgm(85, "Sph", 75, 20))
fvs
##   model    psill    range
## 1   Nug 25.57019   0.0000
## 2   Sph 72.65881 135.7744
plot(variogramLine(fvs, 400), type = "l", ylim = c(0, 120), main = "Spherical Fit")
points(v[, 2:3], pch = 20, col = "navy", cex = 1.5)
```



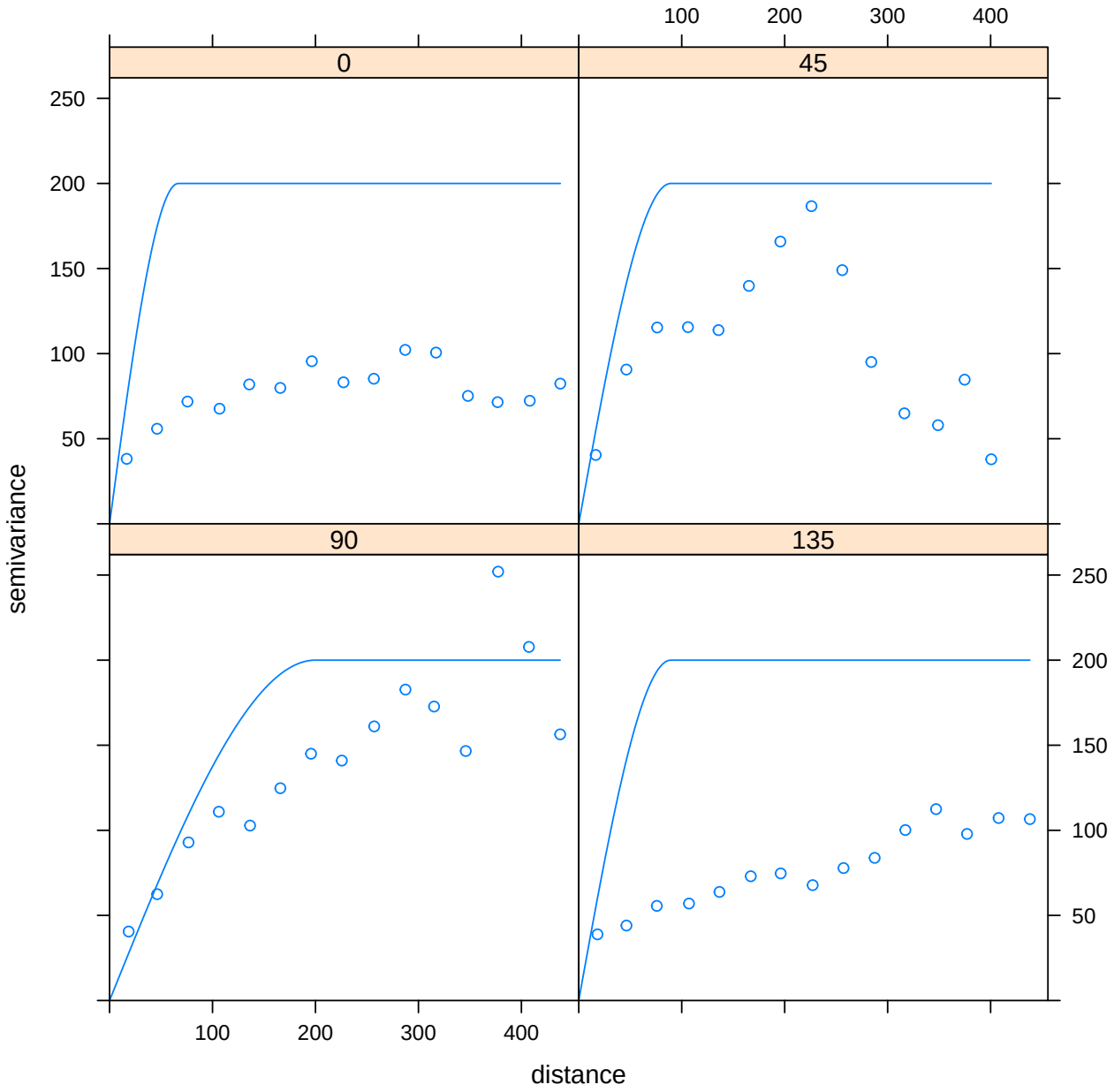
6.1 Anisotropy

Anisotropy is when the range (geometric anisotropy) or the sill (zonal anisotropy) of the semivariogram varies in different directions. Both types of anisotropy can be present.

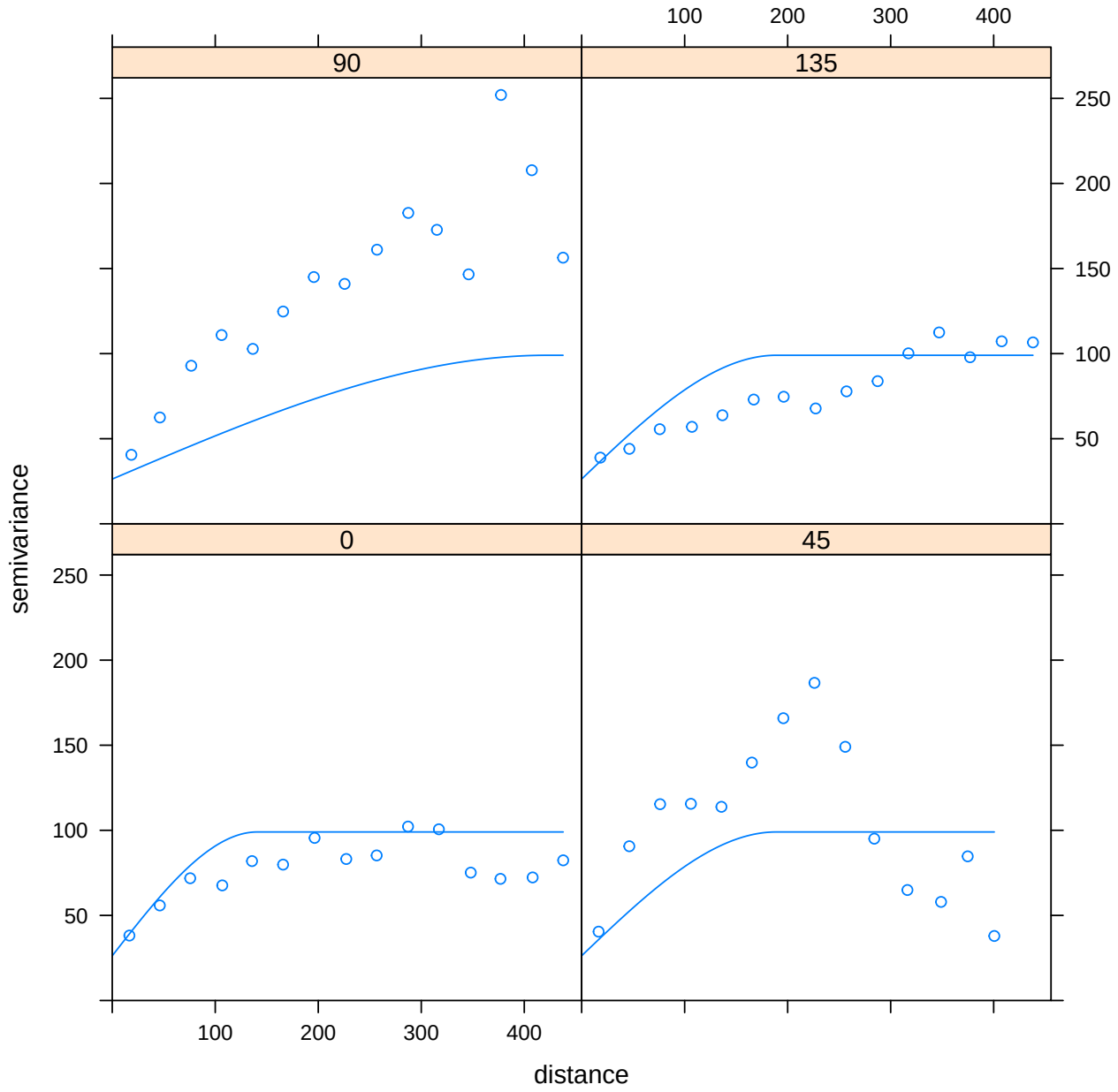
```
# use alpha to set the direction
vgm_aniso <- variogram(gs, alpha = c(0, 45, 90, 135))
plot(vgm_aniso)
```



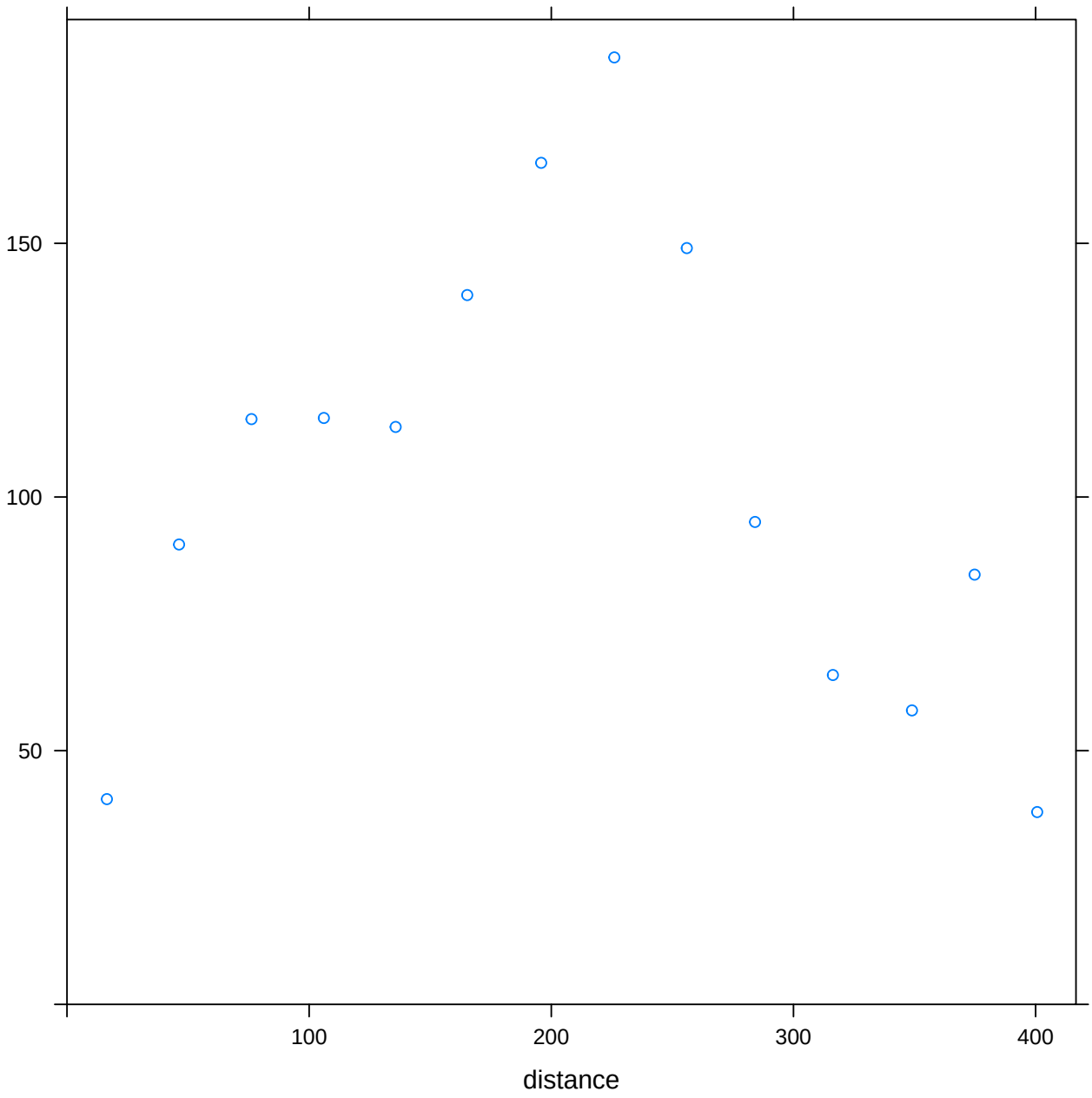

```
# eye ball a model fit, eyeball the sill, range, and nugget
aniso_fit <- vgm(200, "Sph", 200, 0, anis = c(90, 1/3))
plot(vgm_aniso, aniso_fit, as.table = TRUE)
```



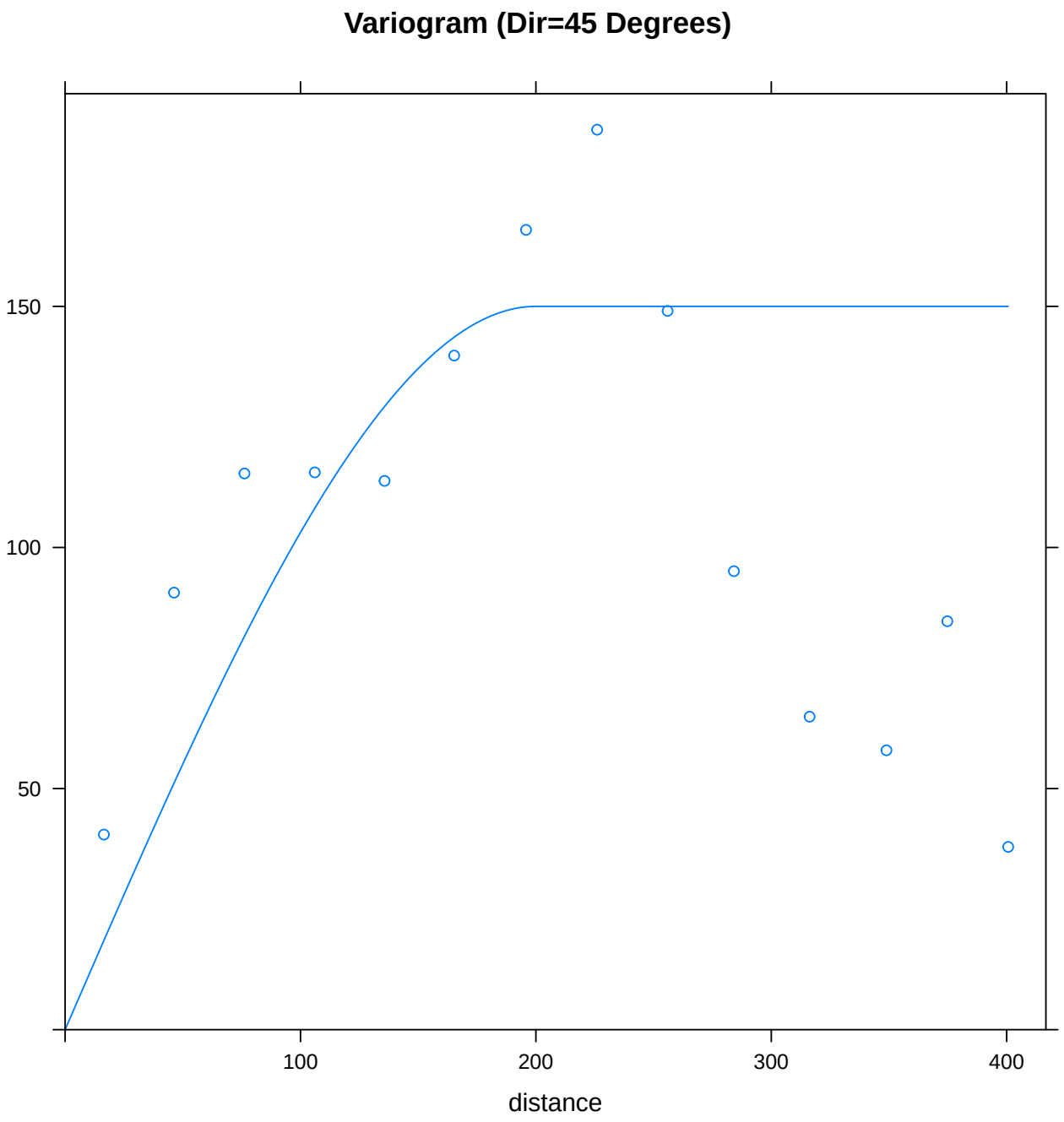

```
# that wasn't very good. let R try to fit a model through regression
raniso_fit <- fit.variogram(vgm_aniso, model = aniso_fit)
plot(vgm_aniso, raniso_fit)
```




```
# can also subset and eyeball a model fit in the 45 Degree direction
vgm_45 <- subset(vgm_aniso, vgm_aniso$dir.hor == 45)
plot(vgm_45)
```




```
vgm_45_sph <- vgm(150, "Sph", 200, 0)
plot(vgm_45, vgm_45_sph, main = "Variogram (Dir=45 Degrees)")  # notice this is another way of plotting
```



7.0 Kriging Models

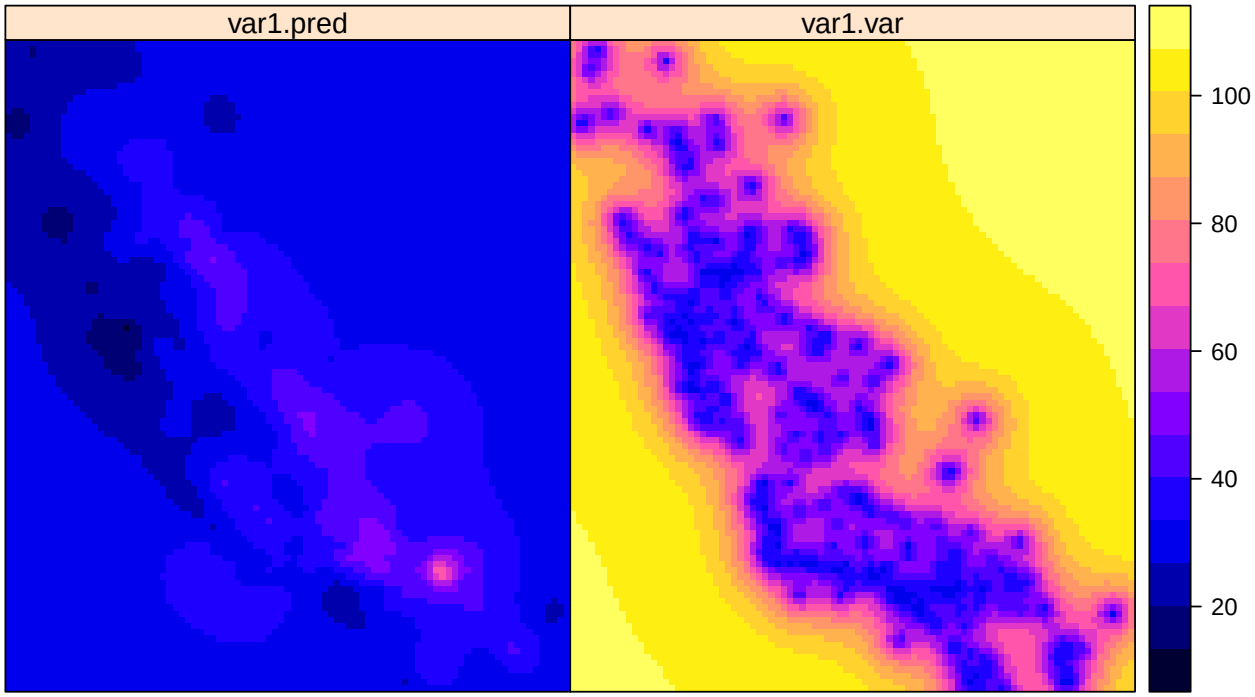
Originating in geostatistics, kriging or Gaussian process regression is a method of interpolation for which the interpolated values are modeled by a Gaussian process governed by prior covariances. We will look at a few different types of kriging below.

7.1 Simple Kriging –known mean

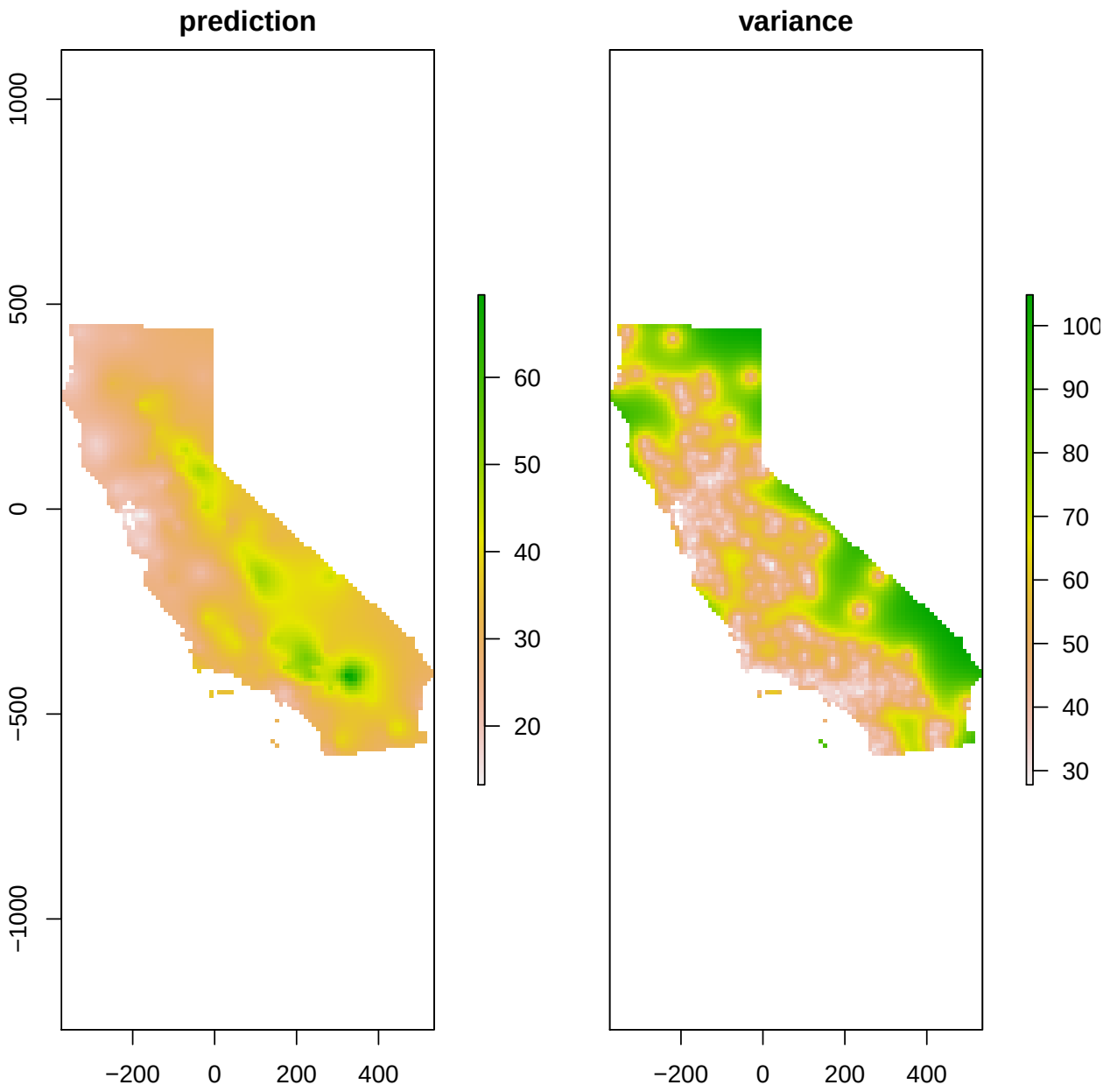
For simple kriging (and simulation based on simple kriging) you need to define a vector with the trend coefficients (including intercept); if no independent variables are defined the model only contains an intercept

and beta should be the simple kriging mean.

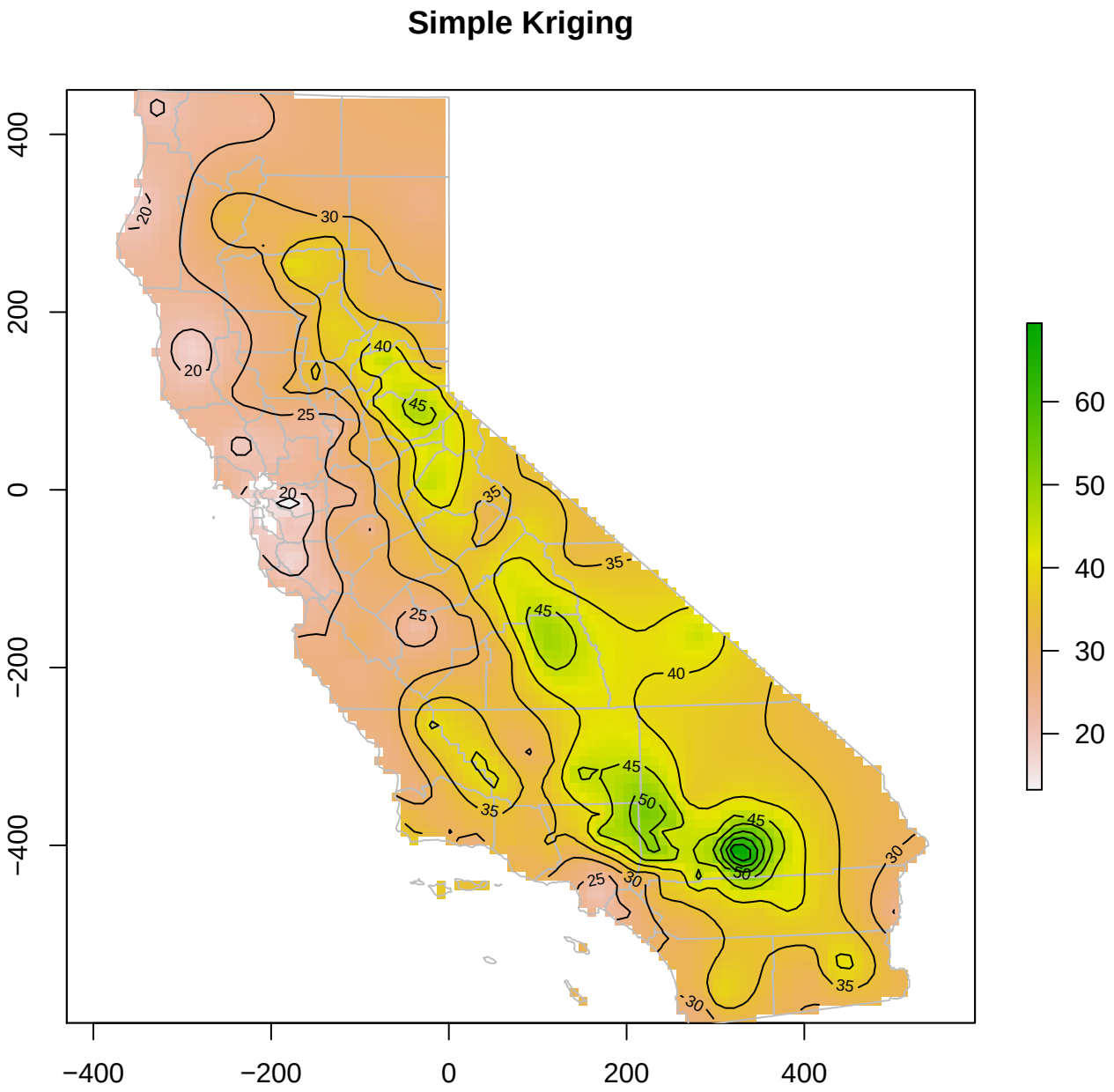
```
skp <- krige(formula = OZDLYAV ~ 1, locations = aq_ta, g, model = fve, beta = mean(aq_data$OZDLYAV))
## [using simple kriging]
spplot(skp)
```




```
# plot nicely
sk <- brick(skp)
sk <- mask(sk, ca_ta)
names(sk) <- c("prediction", "variance")
plot(sk)
title(main = "Simple Kriging", line = 6)
```



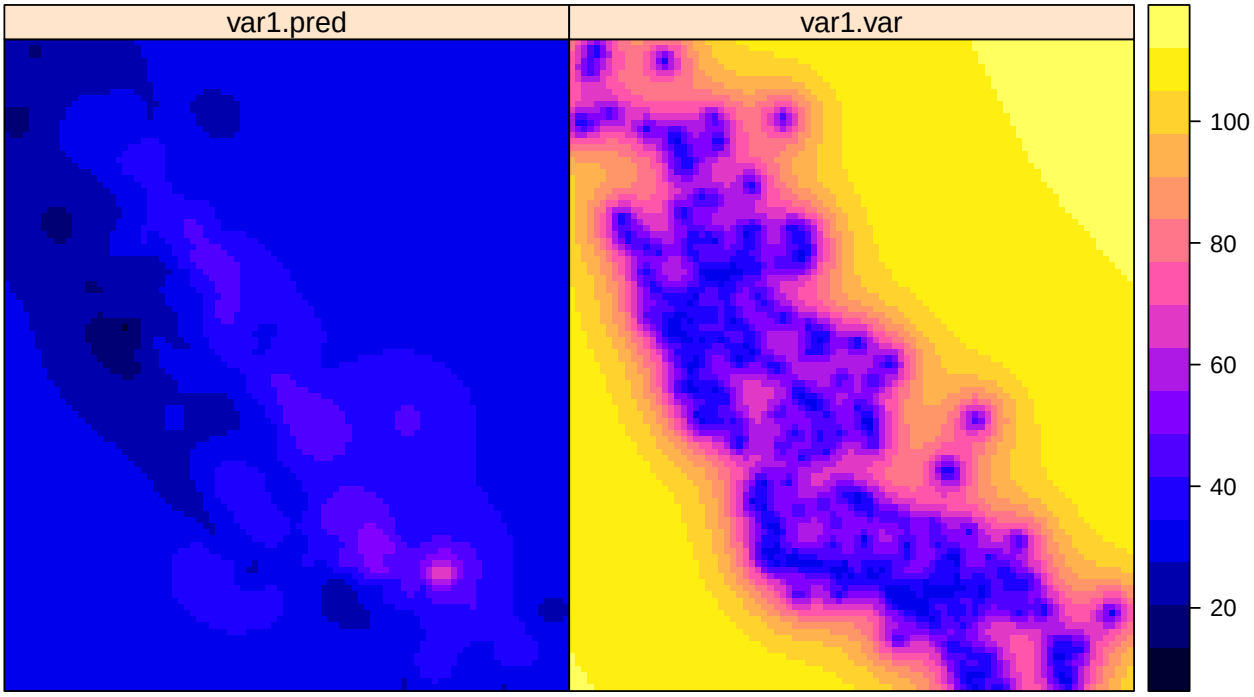

```
plot(sk$prediction, main = "Simple Kriging")
plot(ca_ta, border = "gray", add = TRUE)
contour(sk$prediction, add = TRUE, nlevels = 10)
```



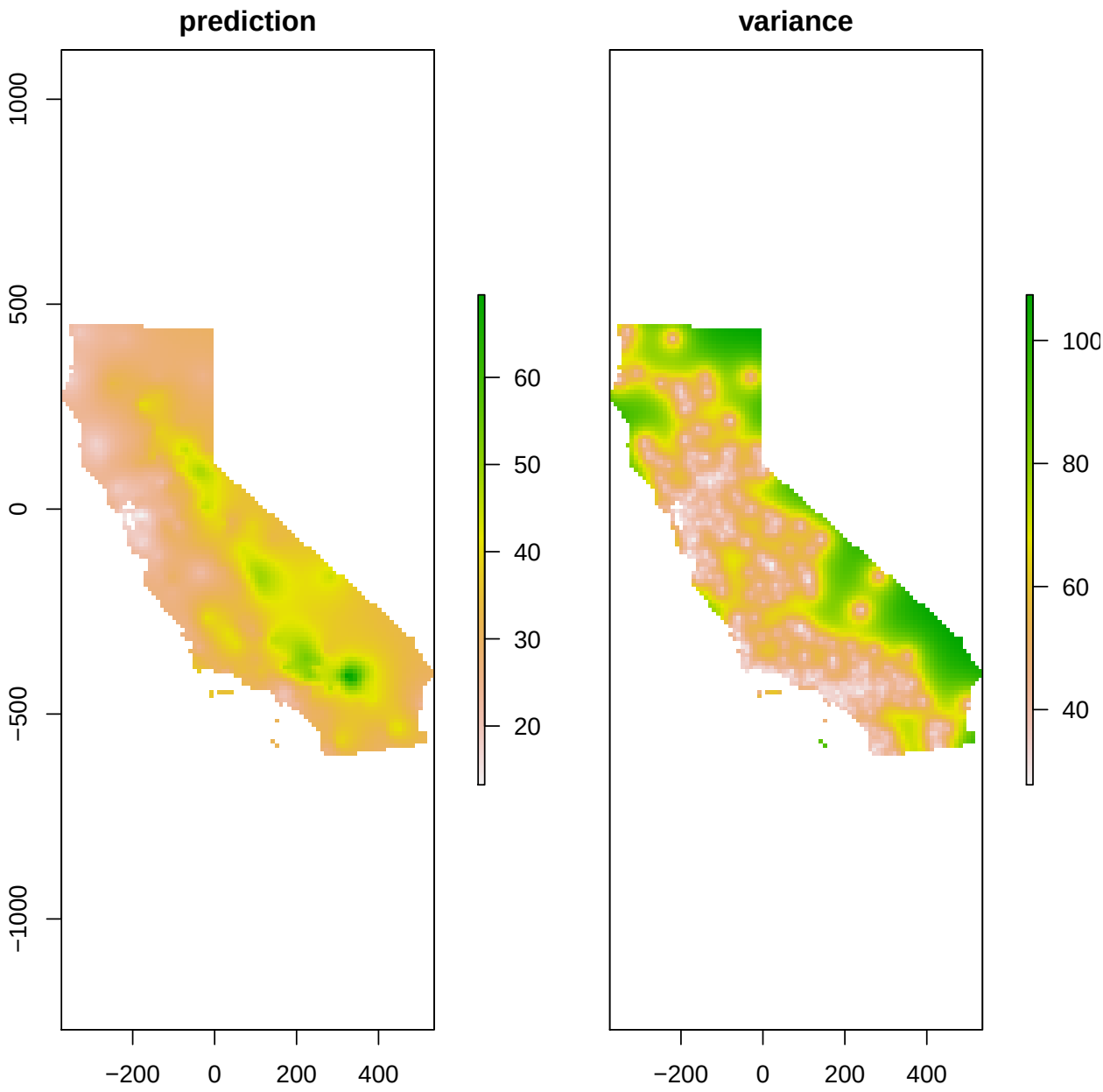
7.2 Ordinary Kriging –no known mean

Leave beta out of simple kriging to get an ordinary kriging estimate.

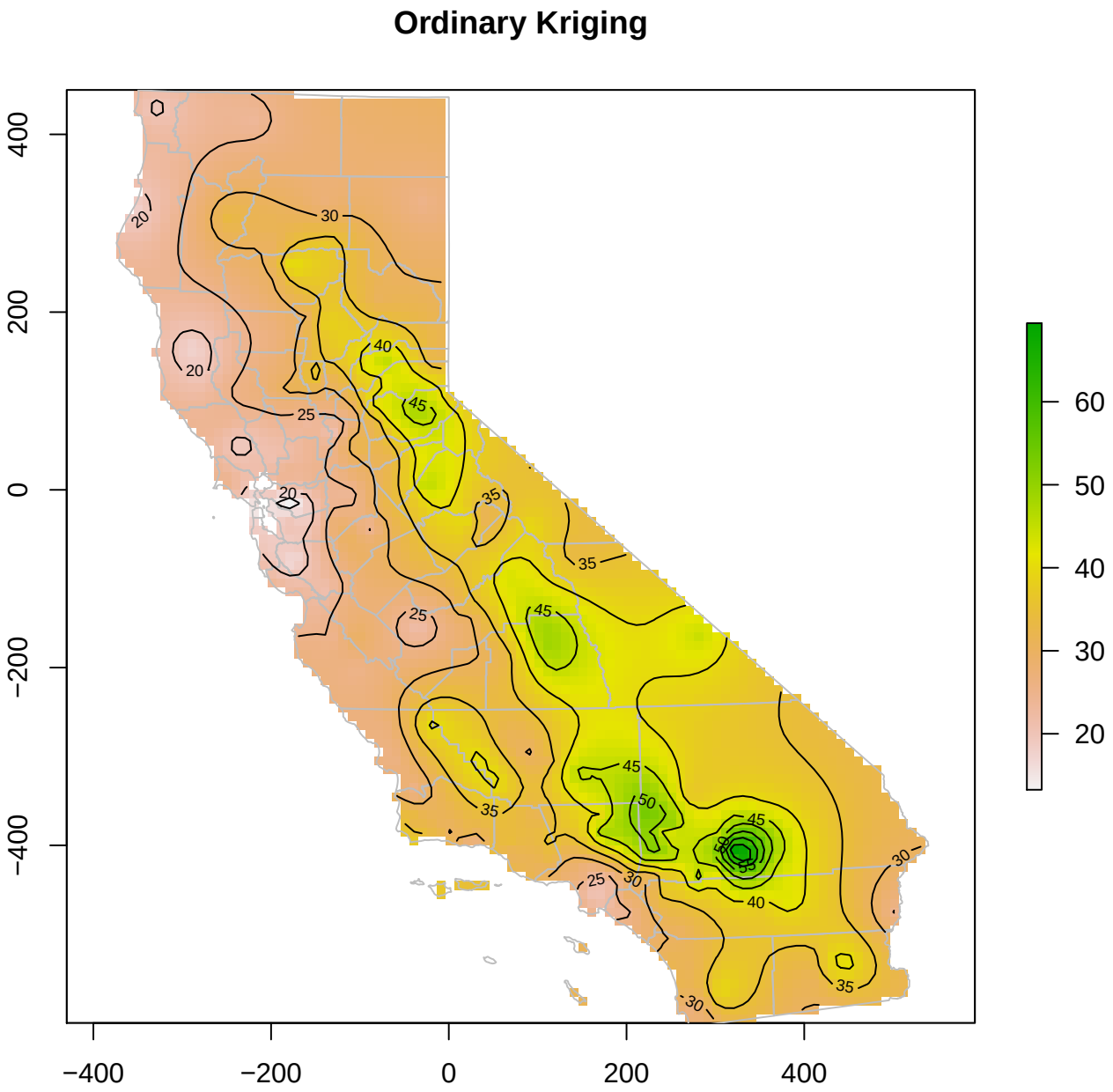
```
okp <- krige(formula = OZDLYAV ~ 1, locations = aq_ta, g, model = fve)
## [using ordinary kriging]
spplot(okp)
```




```
# plot nicely
ok <- brick(okp)
ok <- mask(ok, ca_ta)
names(ok) <- c("prediction", "variance")
plot(ok)
title(main = "Ordinary Kriging", line = 6)
```




```
plot(ok$prediction, main = "Ordinary Kriging")
plot(ca_ta, border = "gray", add = TRUE)
contour(ok$prediction, add = TRUE, nlevels = 10)
```

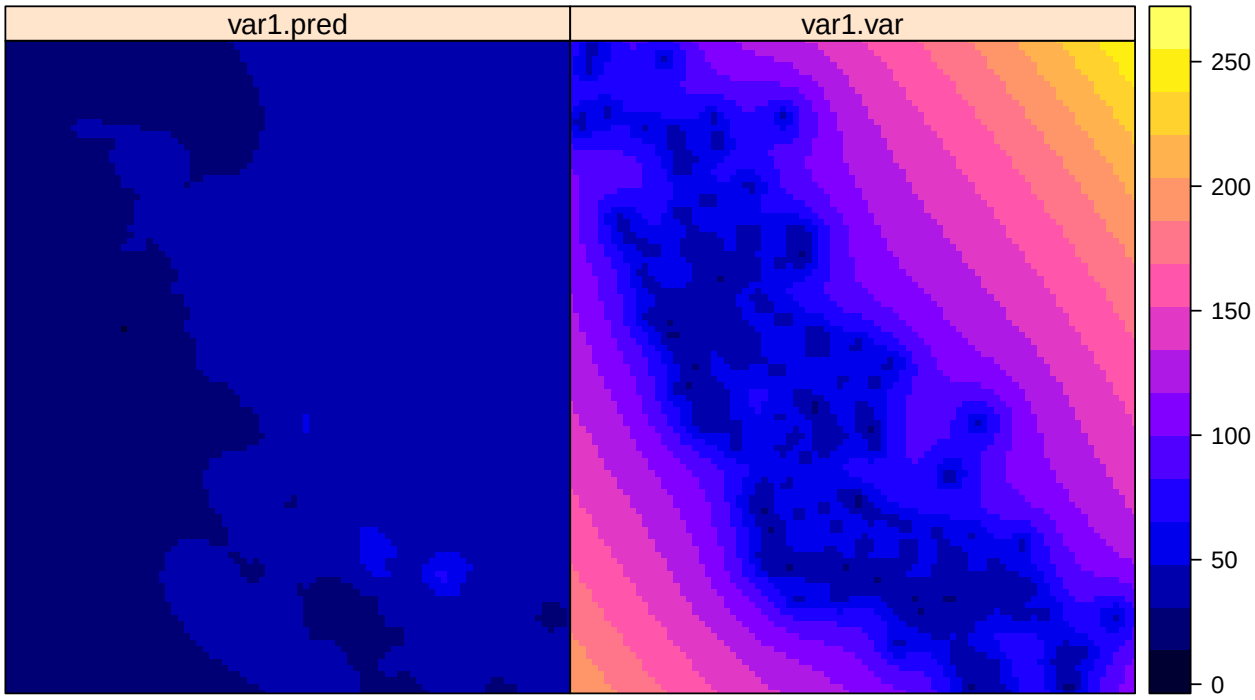


7.3 Universal Kriging –kriging with external drift

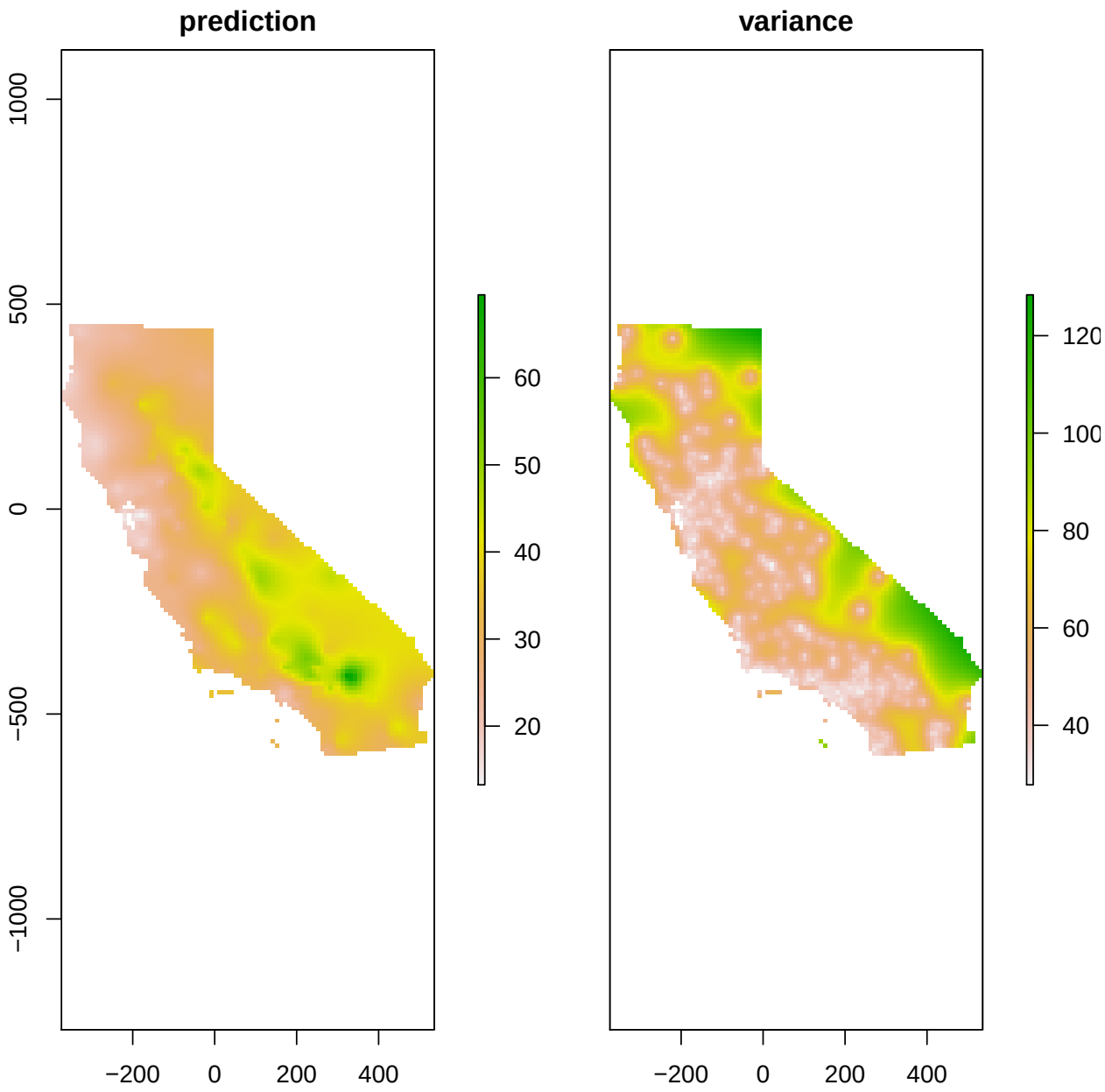
Universal kriging is useful when the variable to map is related to other spatially known variables. For universal kriging, suppose z is a trending surface either linearly or quadratically dependent on x and y. This section shows you how to do both.

```
# set up the x and y coordinates
aq_ta$x <- coordinates(aq_ta)[, 1]
aq_ta$y <- coordinates(aq_ta)[, 2]

# cannot use g, a spatial grid, anymore. The grid here needs to be a spatial pixels data frame!
gpix <- as(r, "SpatialPixelsDataFrame")
## Warning in asMethod(object): object has no values, returning a
## "SpatialPixels" object
```

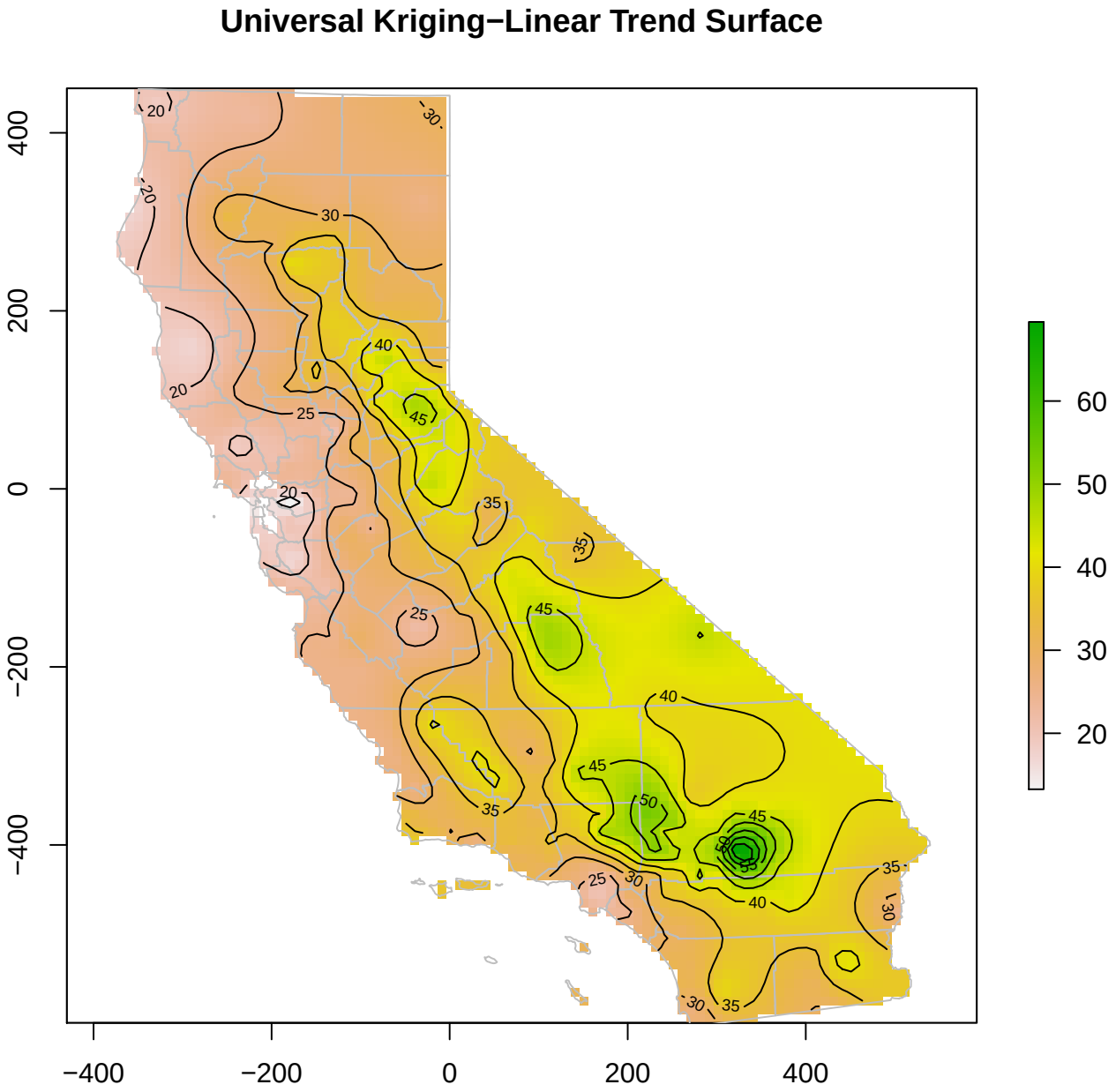
```
# linear trend surface
ukpl <- krige(formula = OZDLYAV ~ x + y, locations = aq_ta, gpix, model = fve)
## [using universal kriging]
spplot(ukpl)
```



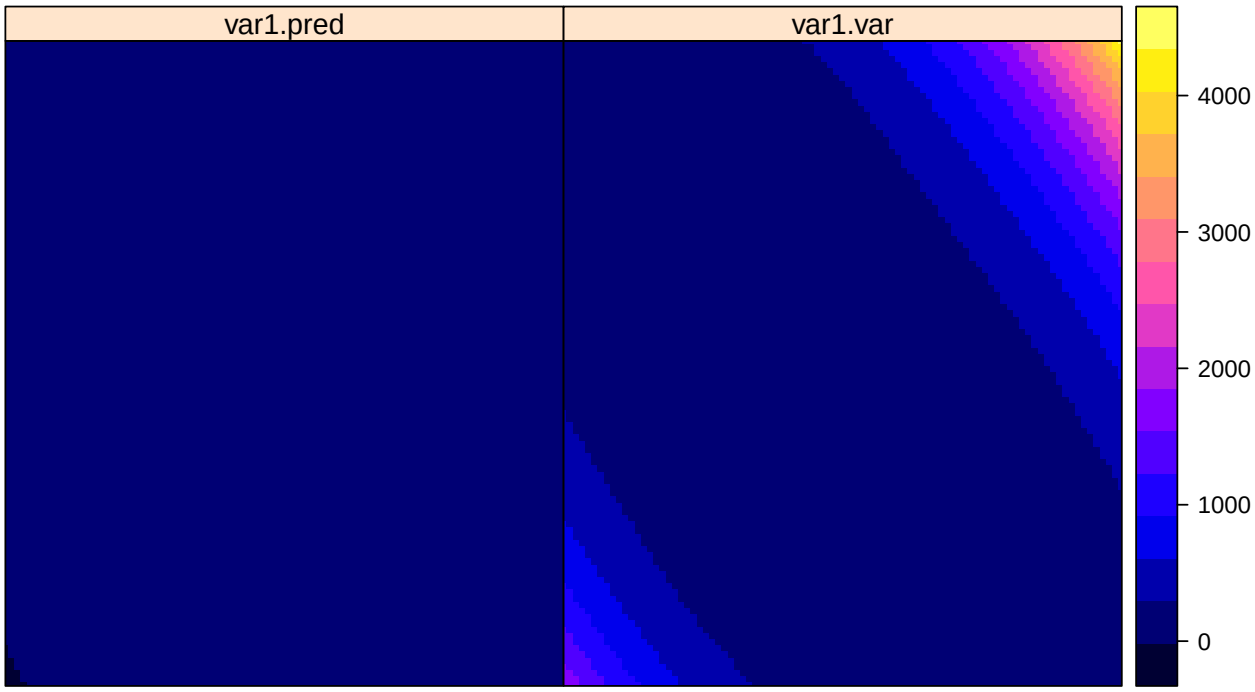

```
# plot nicely
ukl <- brick(ukpl)
ukl <- mask(ukl, ca_ta)
names(ukl) <- c("prediction", "variance")
plot(ukl)
title(main = "Universal Kriging-Linear Trend Surface", line = 6)
```



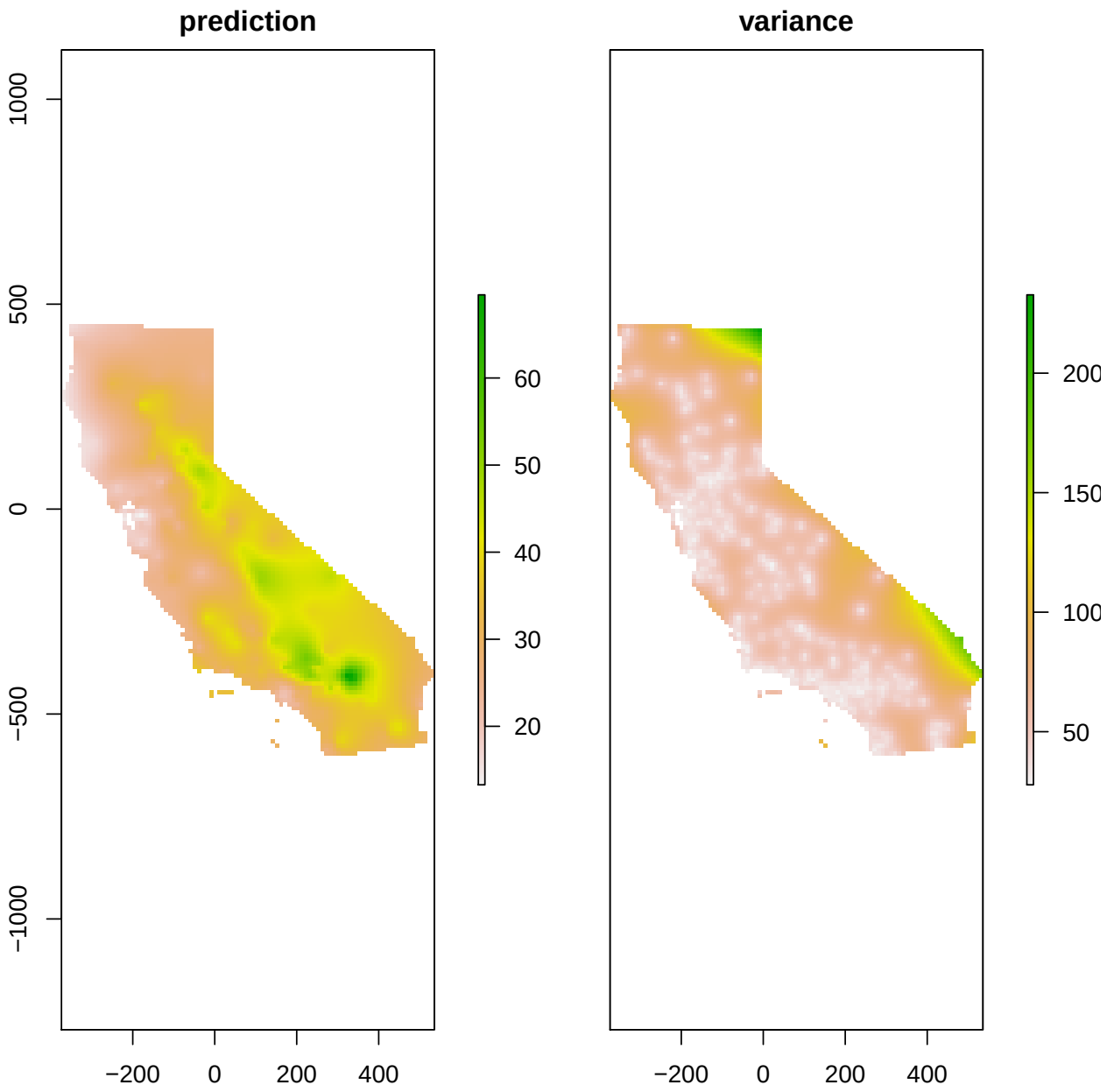

```
plot(ukl$prediction, main = "Universal Kriging-Linear Trend Surface")
plot(ca_ta, border = "gray", add = TRUE)
contour(ukl$prediction, add = TRUE, nlevels = 10)
```



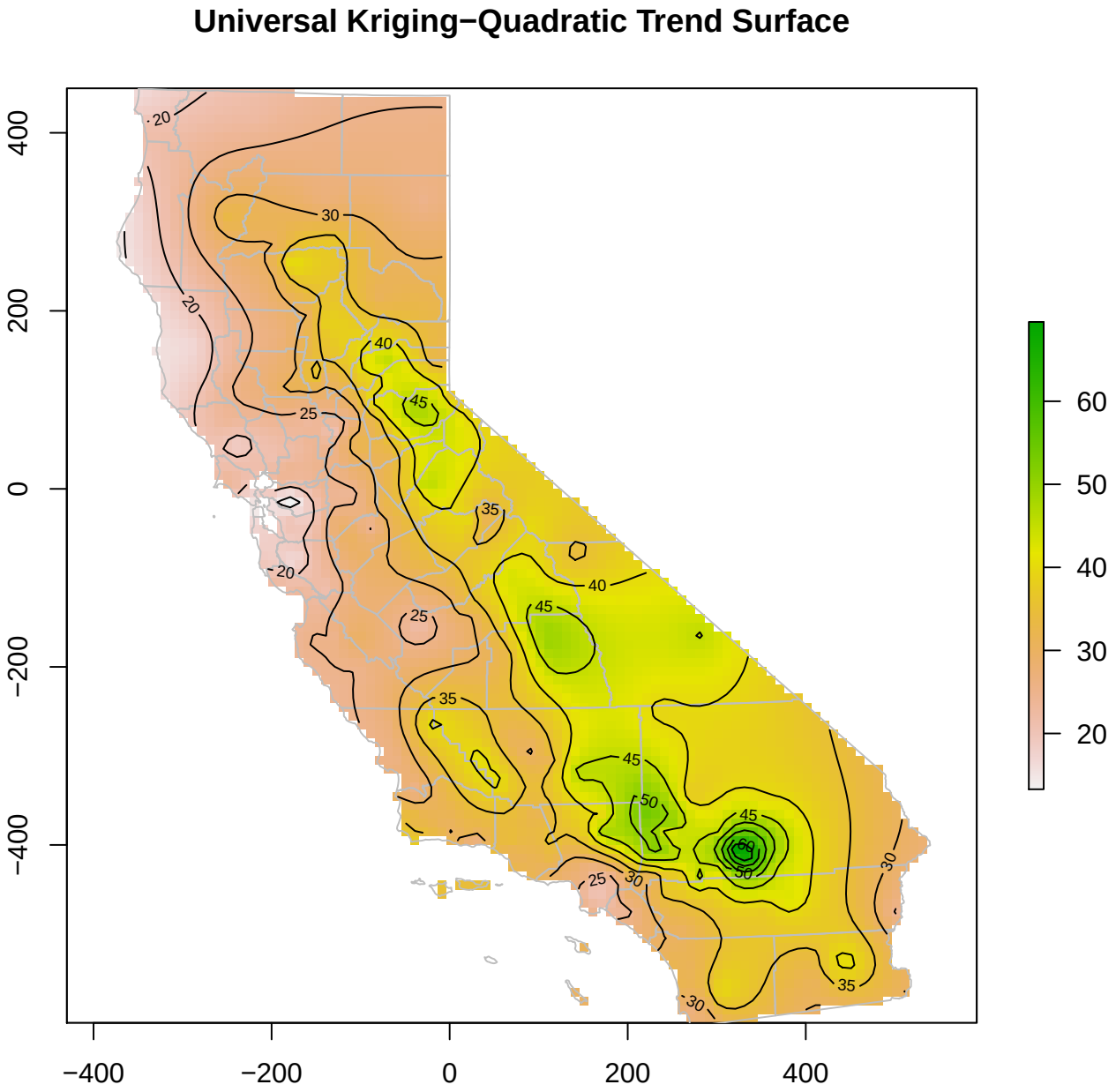

```
# quadratic trend surface
ukpq <- krige(formula = OZDLYAV ~ x + y + I(x * y) + I(x^2) + I(y^2), locations = aq_ta, gpix, model = 
## [using universal kriging]
spplot(ukpq)
```




```
# plot nicely
ukq <- brick(ukpq)
ukq <- mask(ukq, ca_ta)
names(ukq) <- c("prediction", "variance")
plot(ukq)
title(main = "Universal Kriging-Quadratic Trend Surface", line = 6)
```




```
plot(ukq$prediction, main = "Universal Kriging-Quadratic Trend Surface")
plot(ca_ta, border = "gray", add = TRUE)
contour(ukq$prediction, add = TRUE, nlevels = 10)
```

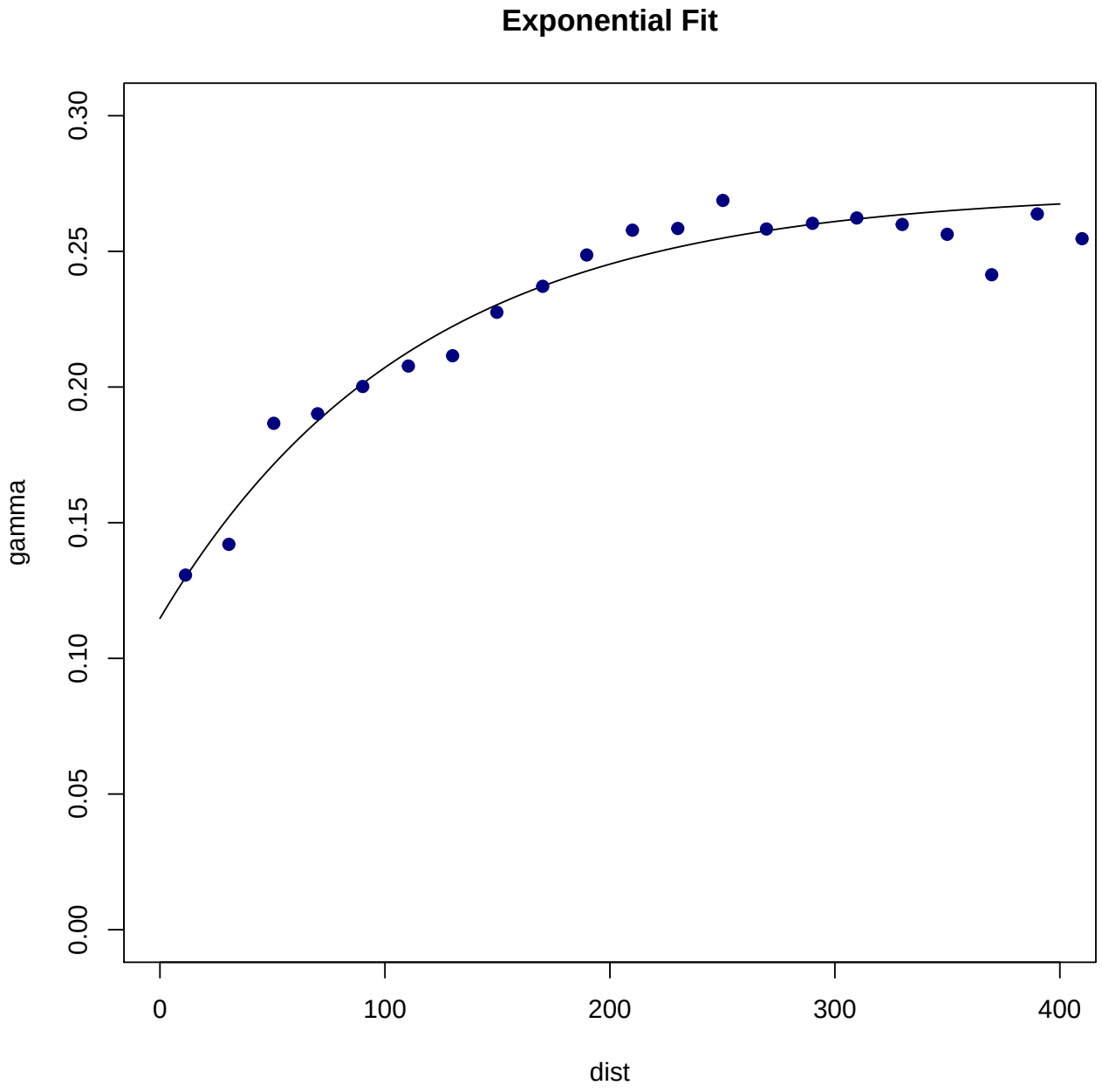


7.4 Indicator Kriging

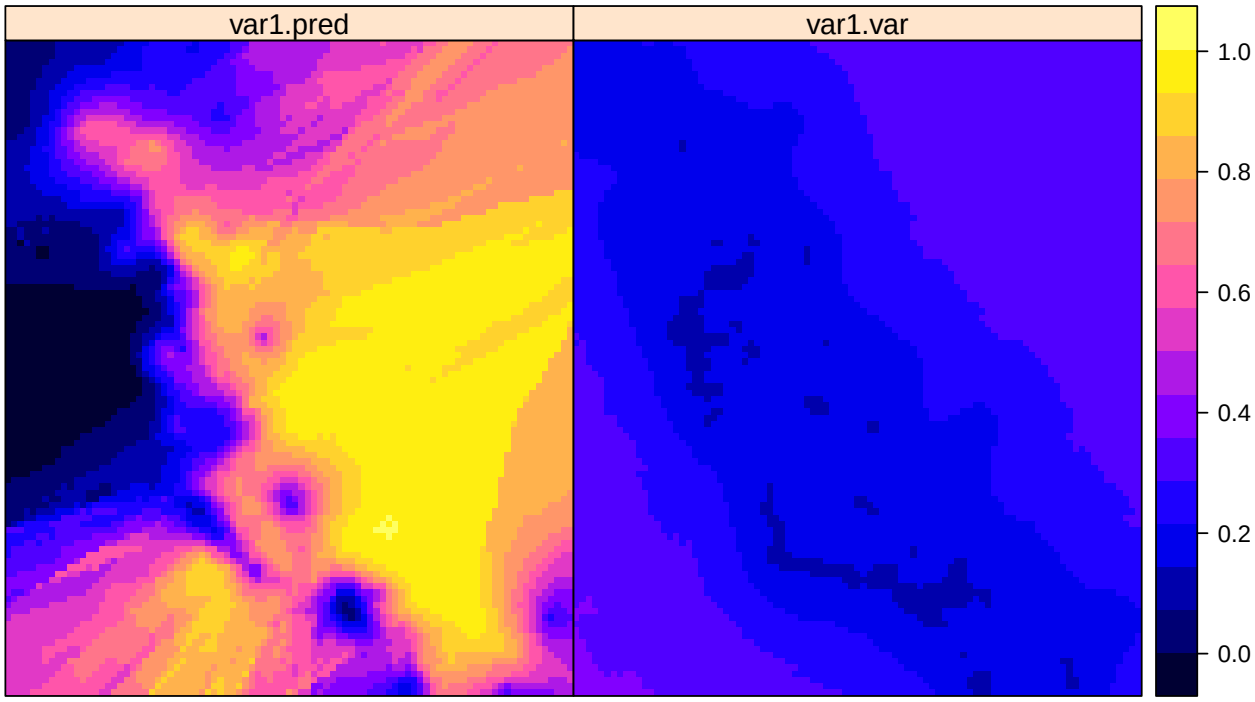
Use indicator kriging to map the probability of ozone exceeding a critical threshold.

```
# pick a threshold, lets look at our variable again
summary(aq_data$OZDLYAV)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.457  23.620  28.350  30.360  35.320  84.650

# develop a variogram for indicator kriging
gsi <- gstat(formula = I(OZDLYAV > 30) ~ 1, locations = aq_ta)
vi <- variogram(gsi, width = 20)
fvis <- fit.variogram(vi, vgm(85, "Exp", 75, 20))
```

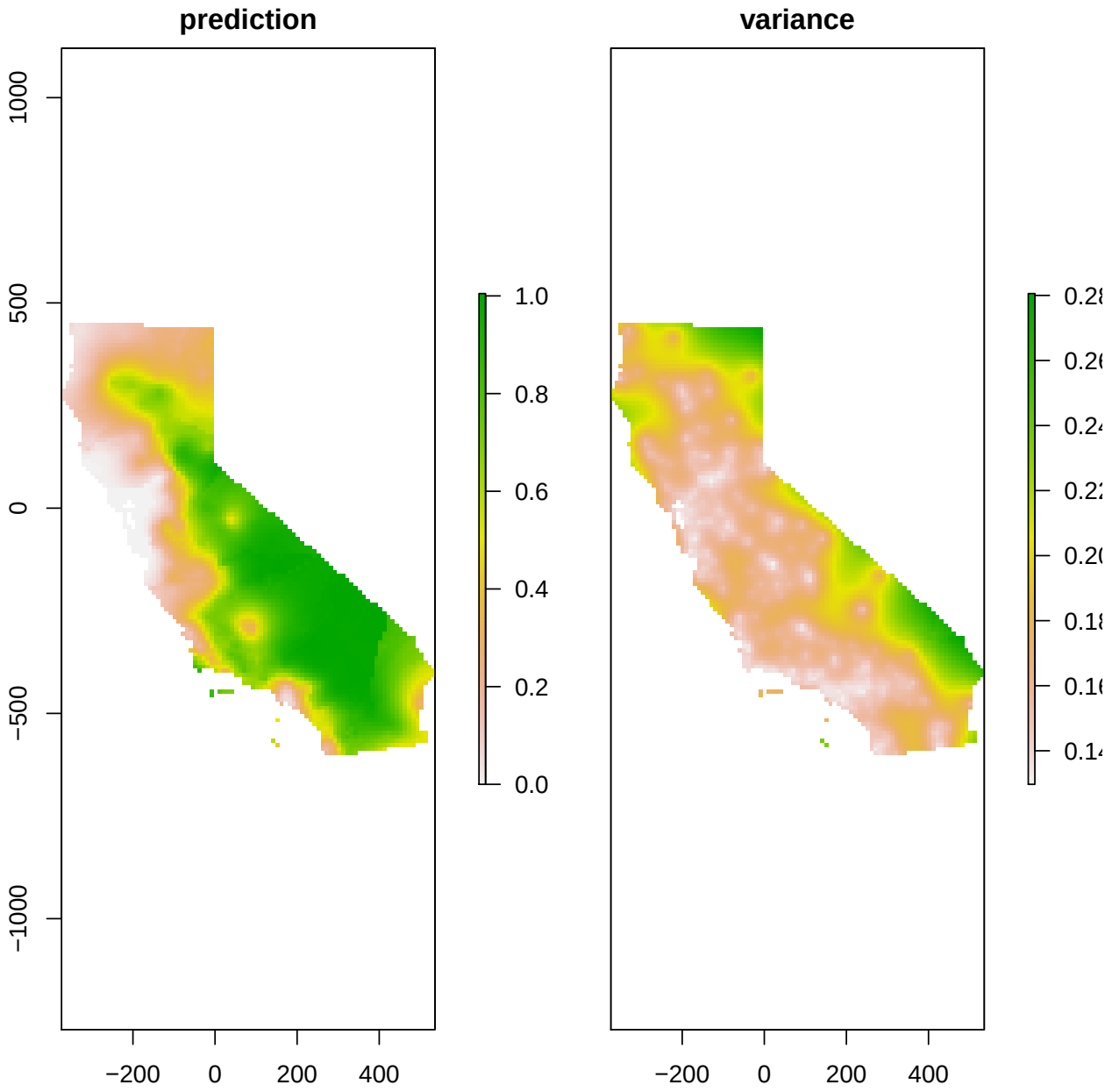
```
plot(variogramLine(fvis, 400), type = "l", ylim = c(0, 0.3), main = "Exponential Fit")
points(vi[, 2:3], pch = 20, col = "navy", cex = 1.5)
```




```
# indicator kriging
ikp <- krige(formula = I(OZDLYAV > 30) ~ 1, locations = aq_ta, g, model = fvis, nmax = 30, indicators =
## [using ordinary kriging]

spplot(ikp)
```




```
# plot nicely
ik <- brick(ikp)
ik <- mask(ik, ca_ta)
names(ik) <- c("prediction", "variance")
plot(ik)
title(main = "Indicator Kriging (i>30ppm)", line = 6)
```

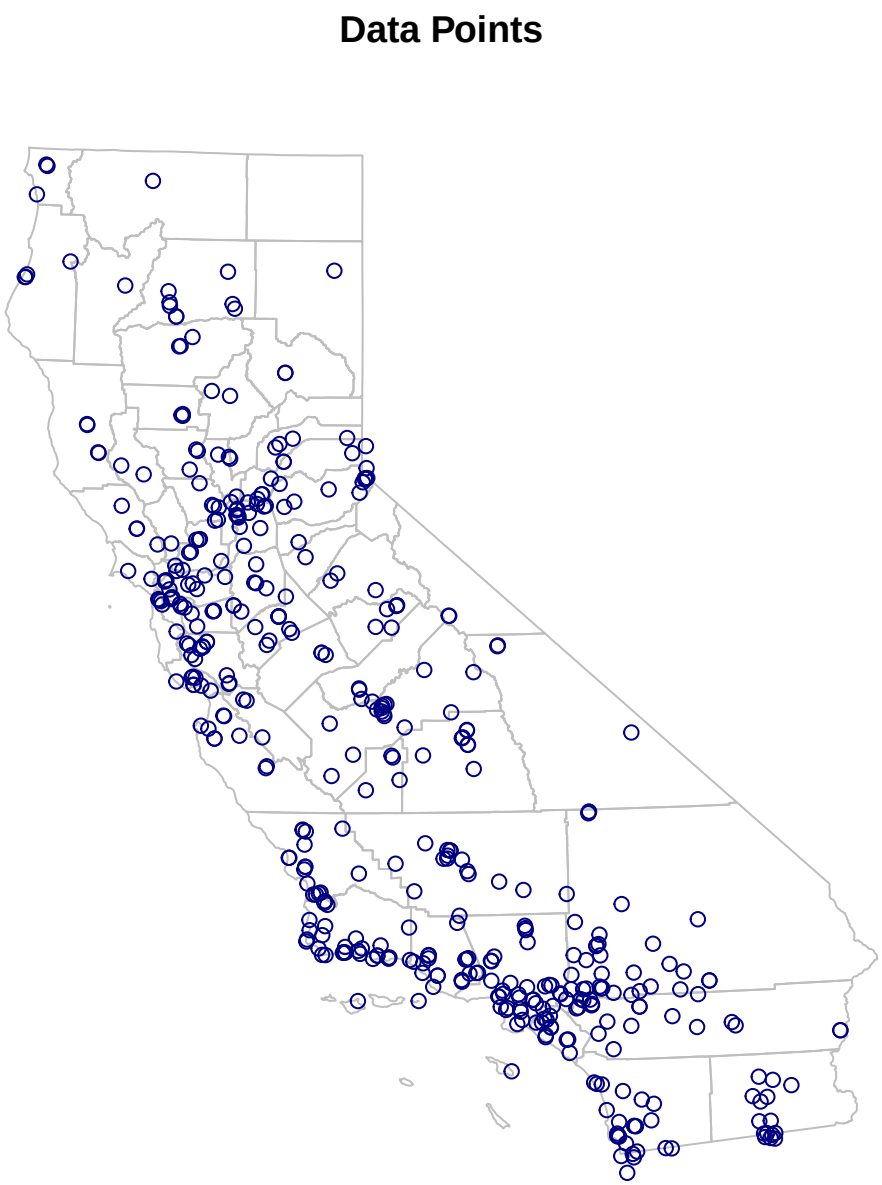


7.5 Local Kriging

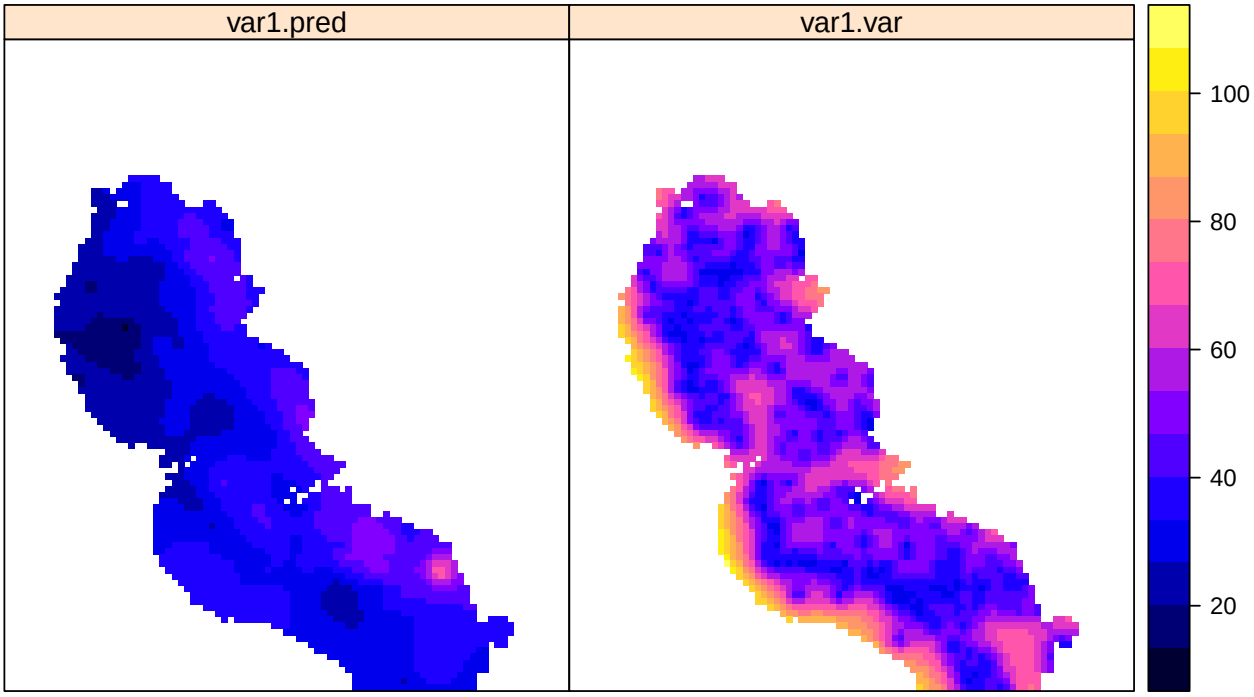
In kriging, by default, all observations are used. But we can use nmin, nmax and or maxdist to restrict the number of observations the kriging estimator can use.
nmax: the max number of nearest observations that should be used for a kriging prediction or simulation, where nearest is defined in terms of the space of the spatial locations.
maxdist: only observations within a distance of maxdist from the prediction location are used for prediction or simulation; if combined with nmax, both criteria apply.
nmin: if the number of nearest observations within distance maxdist is less than nmin, a missing value will be generated; see maxdist.

```
# remember the dimensions of the data
dim(aq_data)
## [1] 452  37
```

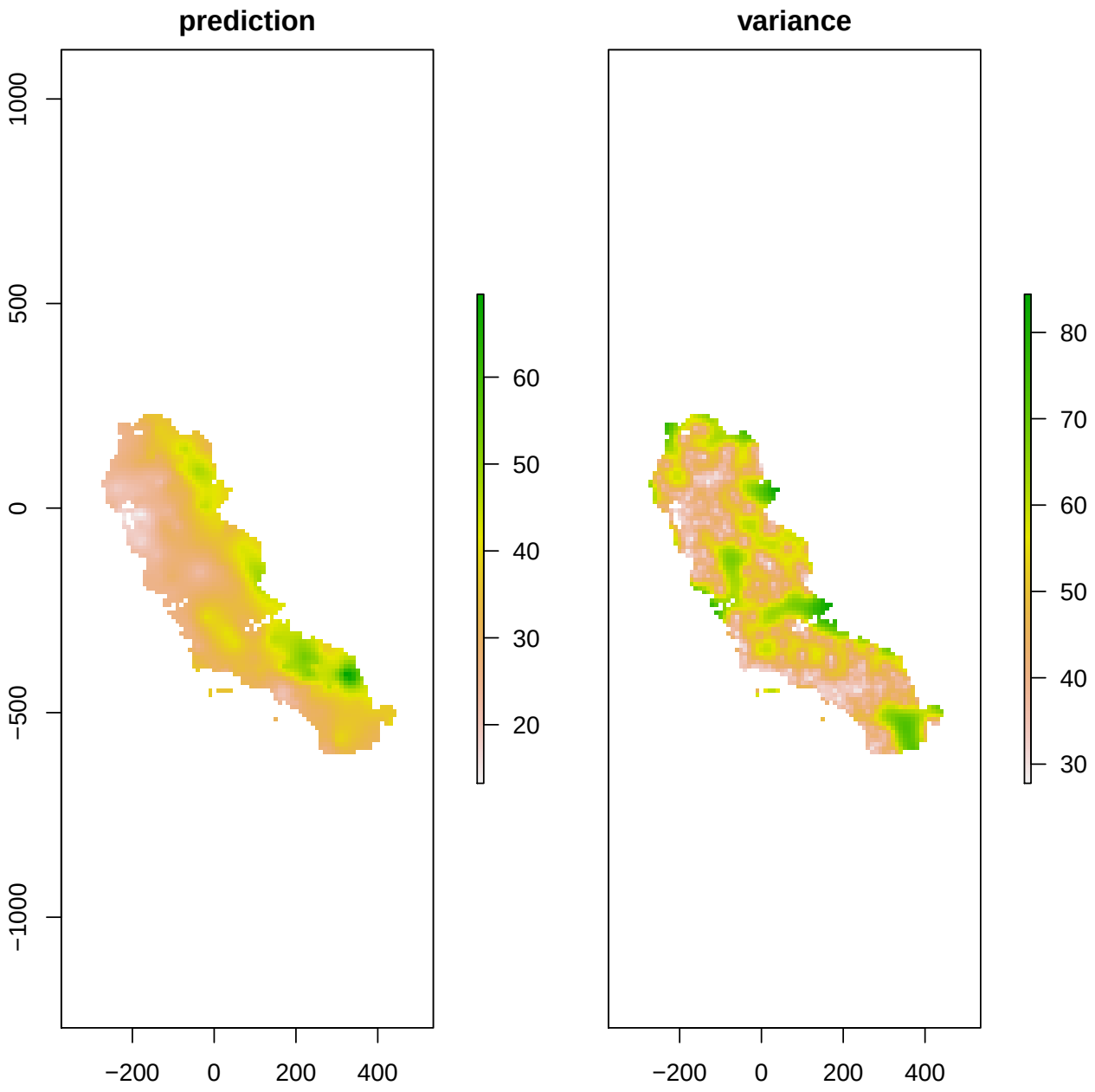
```
# and the clustering of data points
plot(ca_ta, border = "gray", main = "Data Points")
points(aq_ta, col = "navy")
```



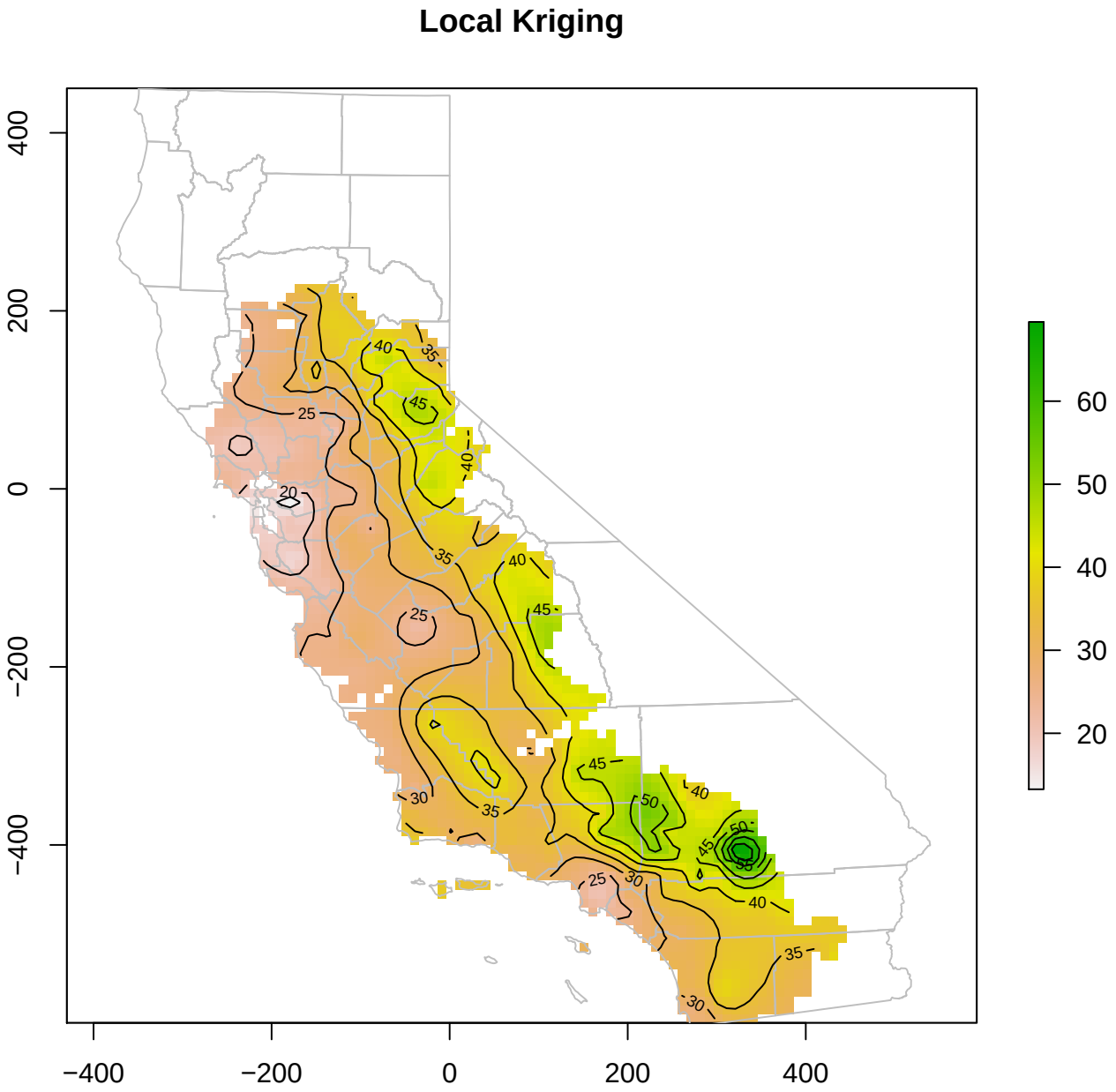

```
# now pick an appropriate nmin, nmax, and or maxdist
lkp <- krige(formula = OZDLYAV ~ 1, locations = aq_ta, g, model = fve, nmin = 20, nmax = 100, maxdist =
## [using ordinary kriging]
spplot(lkp)
```




```
# plot nicely
lk <- brick(lkp)
lk <- mask(lk, ca_ta)
names(lk) <- c("prediction", "variance")
plot(lk)
title(main = "Local Kriging", line = 6)
```




```
plot(lk$prediction, main = "Local Kriging")
plot(ca_ta, border = "gray", add = TRUE)
contour(lk$prediction, add = TRUE, nlevels = 10)
```

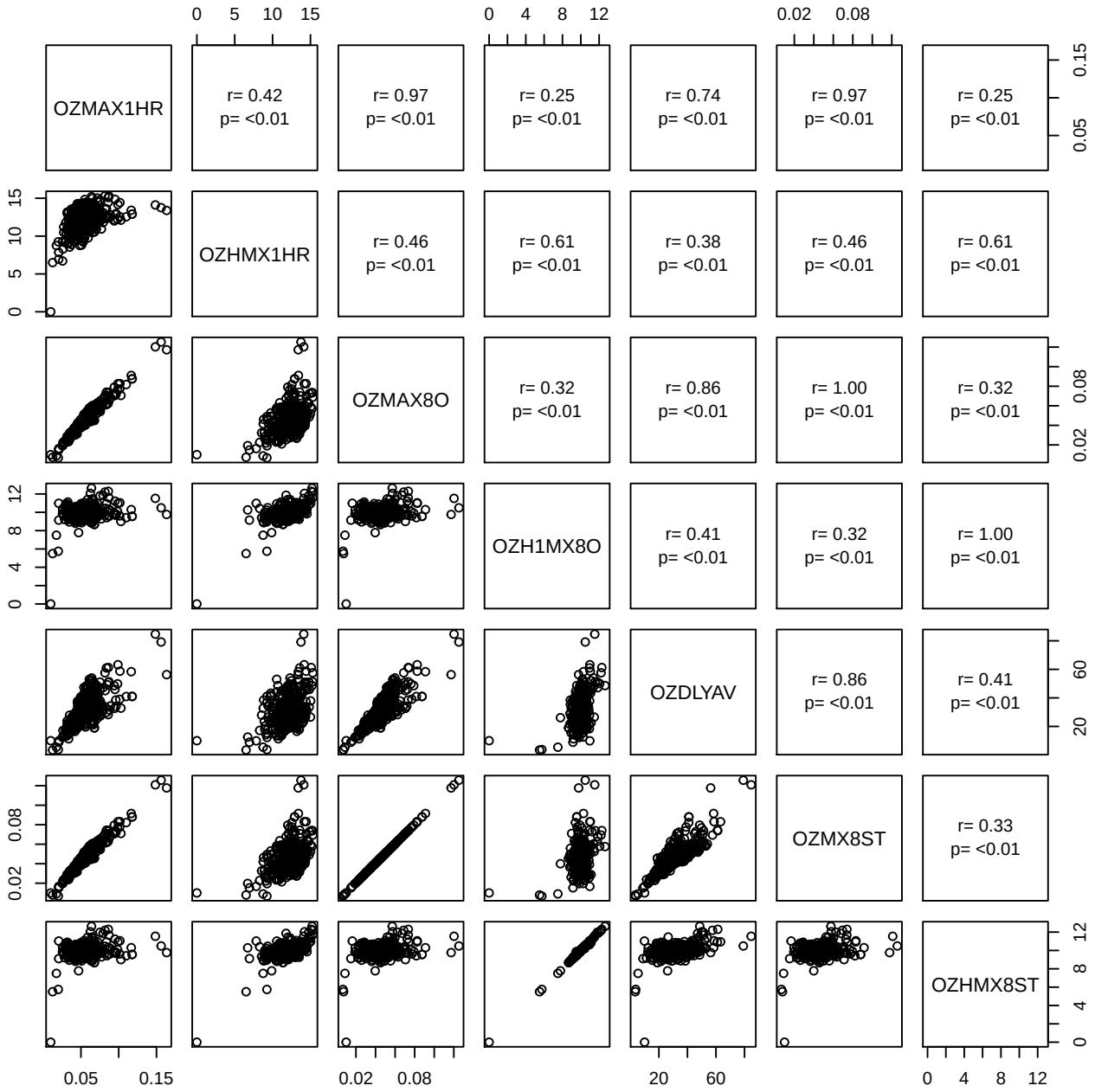


7.6 CoKriging

Use cokriging, or making use of other data sets, to make a better prediction. If the new data is grounded in pysicall processes that influence the variable of interest, the variance in the error prediction can be reduced.

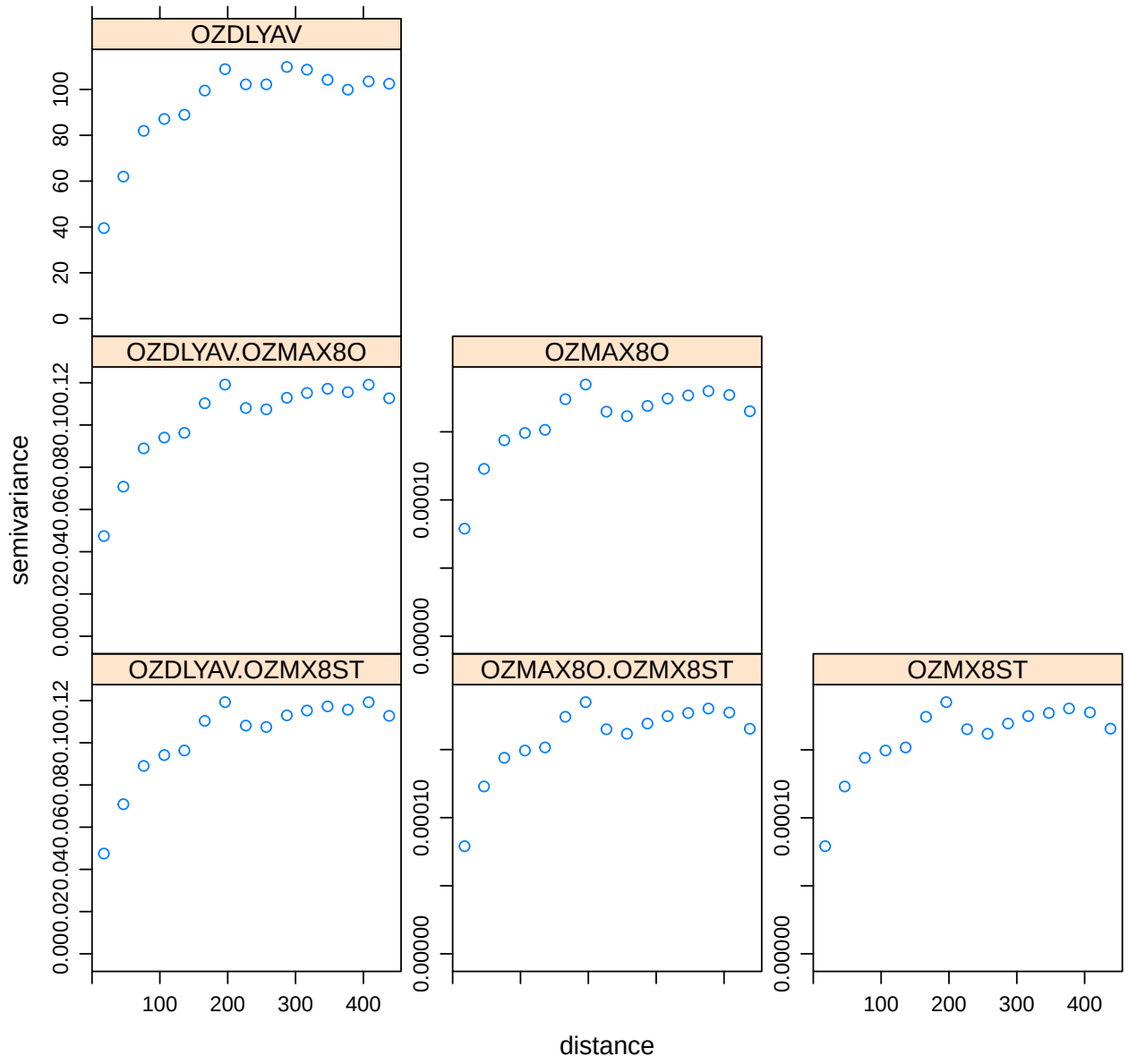
```
# find potentially useful data sets, use scatterplot matrix

# this function will just print the r and pvalue in our scatterplot matrix below
panel.cor <- function(x, y, digits = 2, cex.cor, ...) {
    usr <- par("usr")
    on.exit(par(usr))
    par(usr = c(0, 1, 0, 1))
    # correlation coefficient
    r <- cor(x, y)
```

```
    txt <- format(c(r, 0.123456789), digits = digits)[1]
    txt <- paste("r= ", txt, sep = "")
    text(0.5, 0.6, txt)

    # p-value calculation
    p <- cor.test(x, y)$p.value
    txt2 <- format(c(p, 0.123456789), digits = digits)[1]
    txt2 <- paste("p= ", txt2, sep = "")
    if (p < 0.01)
        txt2 <- paste("p= ", "<0.01", sep = "")
    text(0.5, 0.4, txt2)
}

pairs(aq_data[, 22:28], upper.panel = panel.cor)
```

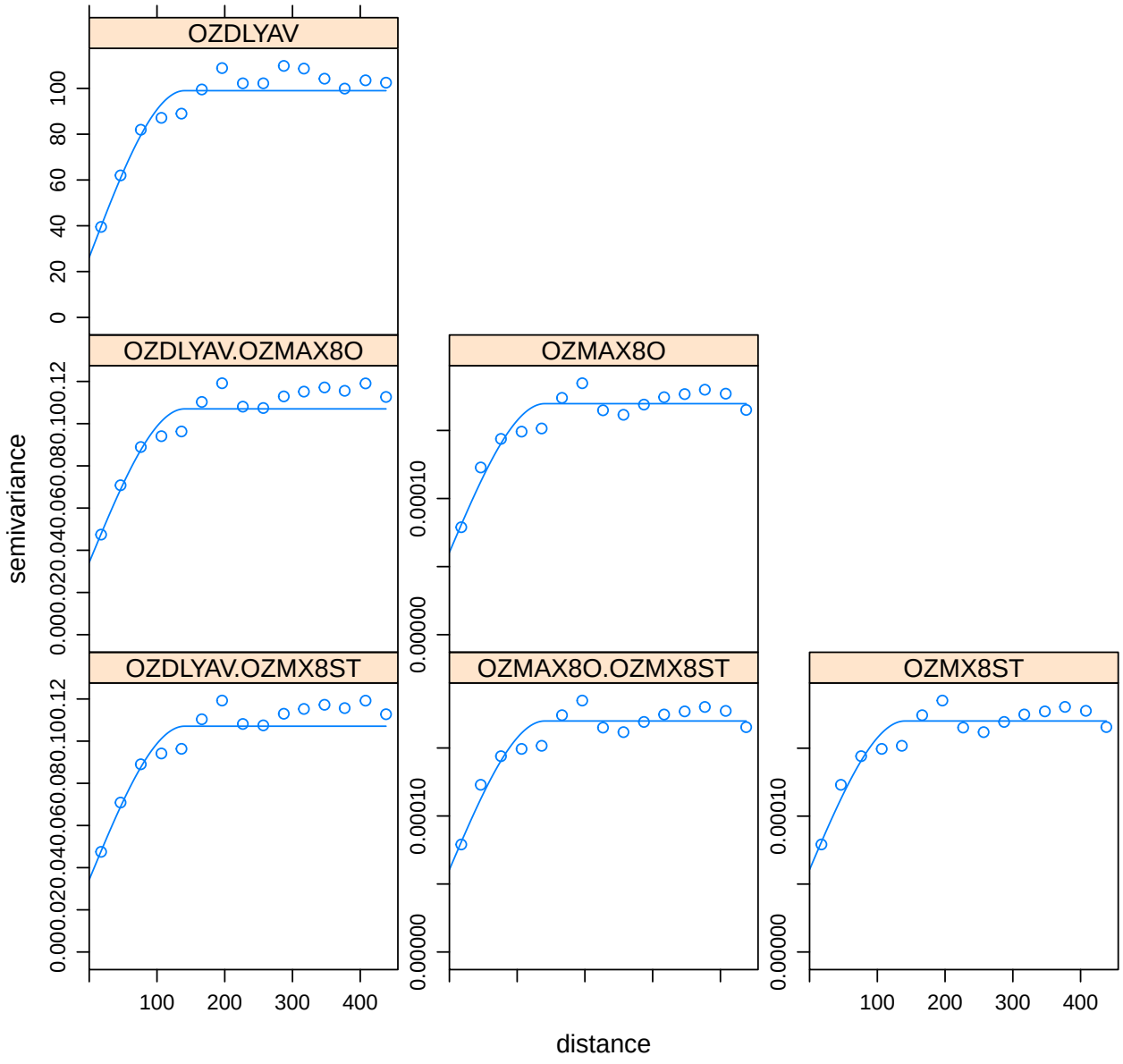



```
# based on the r values in the scatter plot matrix above, I will be using the following data sets
summary(aq_ta$OZMAX8O)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00675 0.03663 0.04458 0.04539 0.05181 0.12520
summary(aq_ta$OZMX8ST)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00675 0.03711 0.04503 0.04586 0.05227 0.12560

# now to create a gstat object and the variogram model
gsc1 <- gstat(id = "OZDLYAV", formula = OZDLYAV ~ 1, locations = aq_ta)
gsc2 <- gstat(gsc1, id = "OZMAX8O", formula = OZMAX8O ~ 1, locations = aq_ta)
gsc3 <- gstat(gsc2, id = "OZMX8ST", formula = OZMX8ST ~ 1, locations = aq_ta)
```

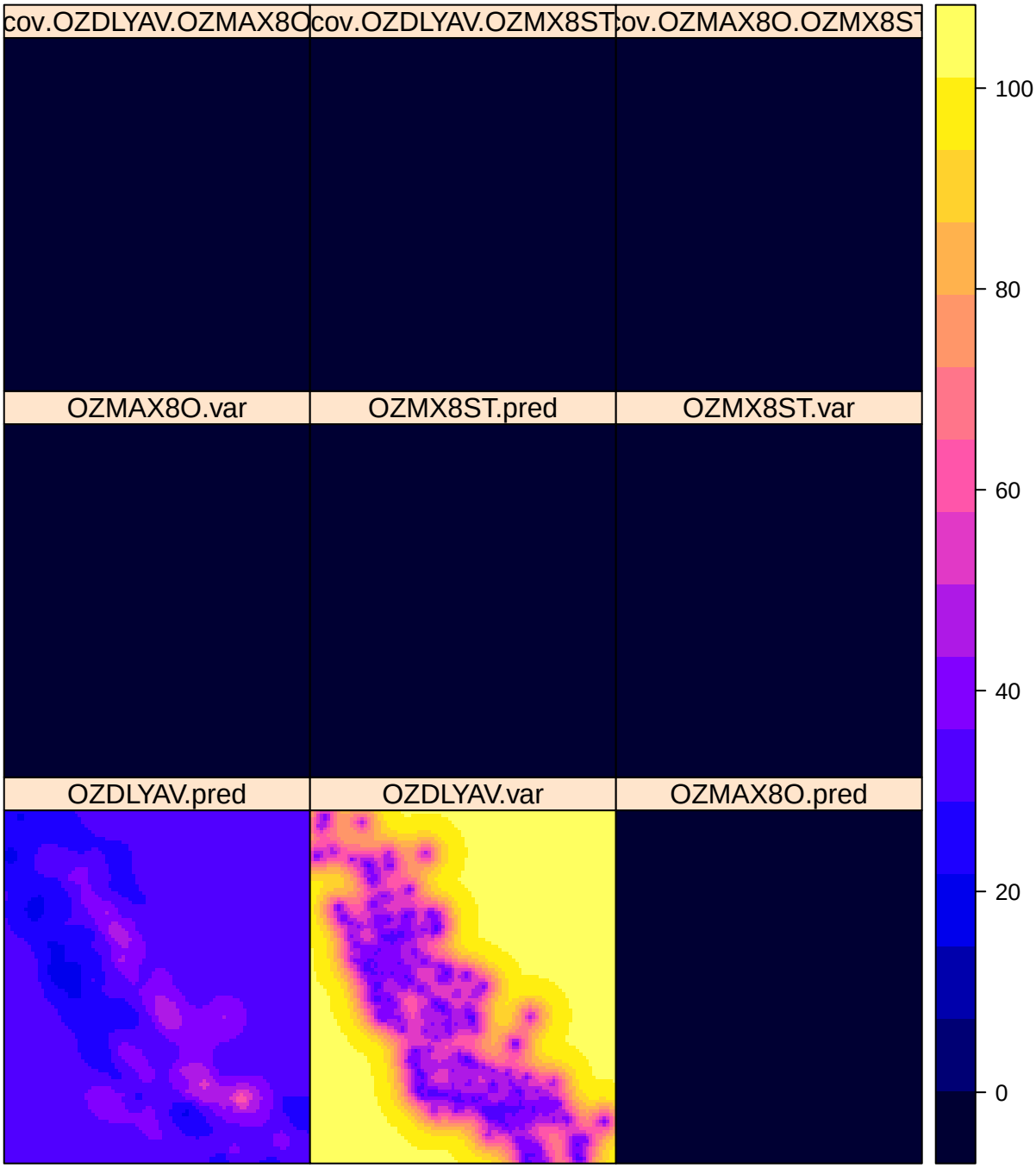
```
# plot the variograms and cross-variograms
plot(variogram(gsc3))
```



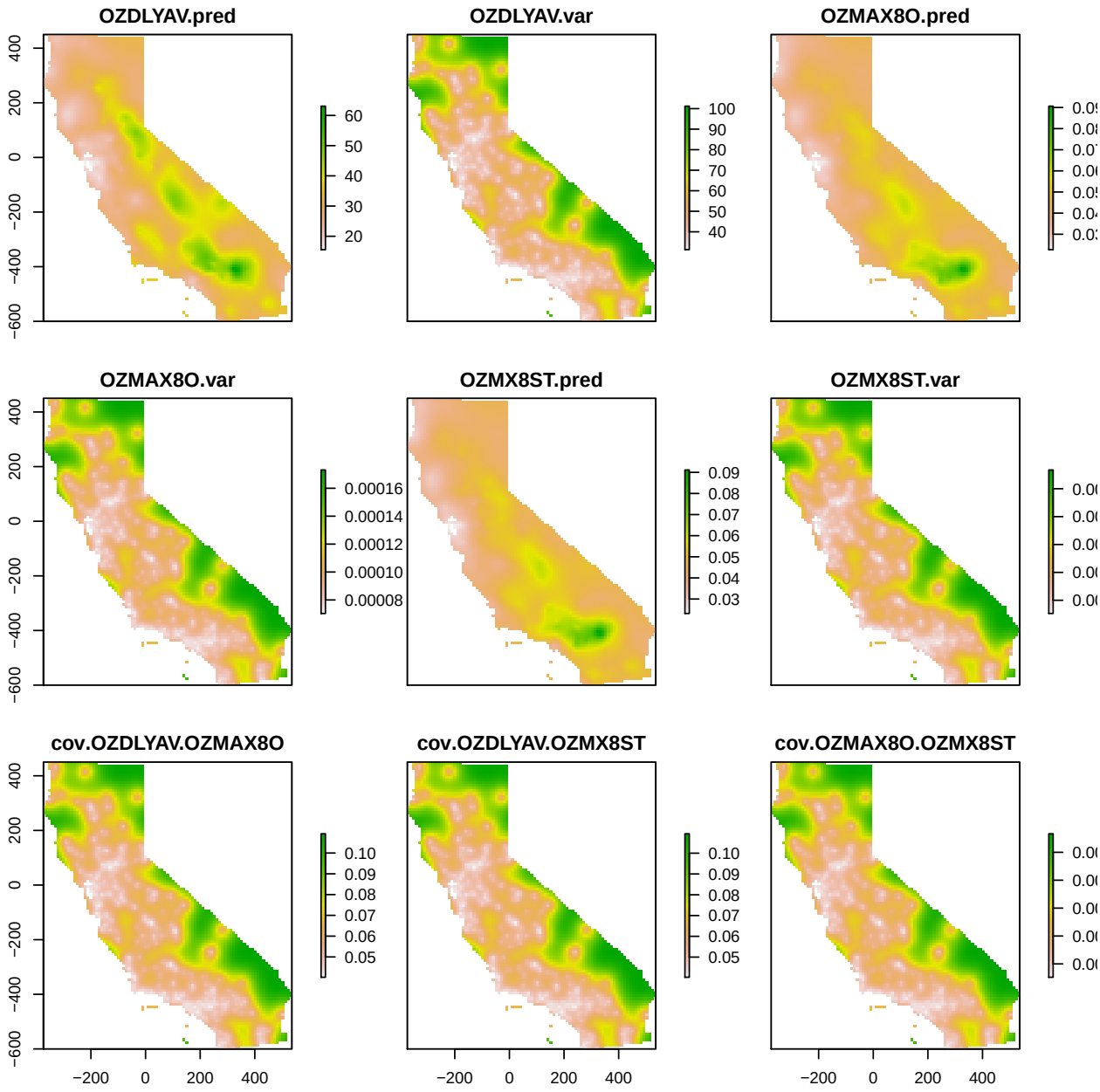

```
# fit a model variogram to the target variogram
fvsc <- fit.variogram(variogram(gsc1), vgm(85, "Sph", 75, 20))

# fit a model variogram to all the variograms
vmc <- variogram(gsc3)
fvmc <- fit.lmc(vmc, gsc3, model = fvsc)

# plot the fitted variograms to all the sample variograms:
plot(variogram(gsc3), fvmc)
```



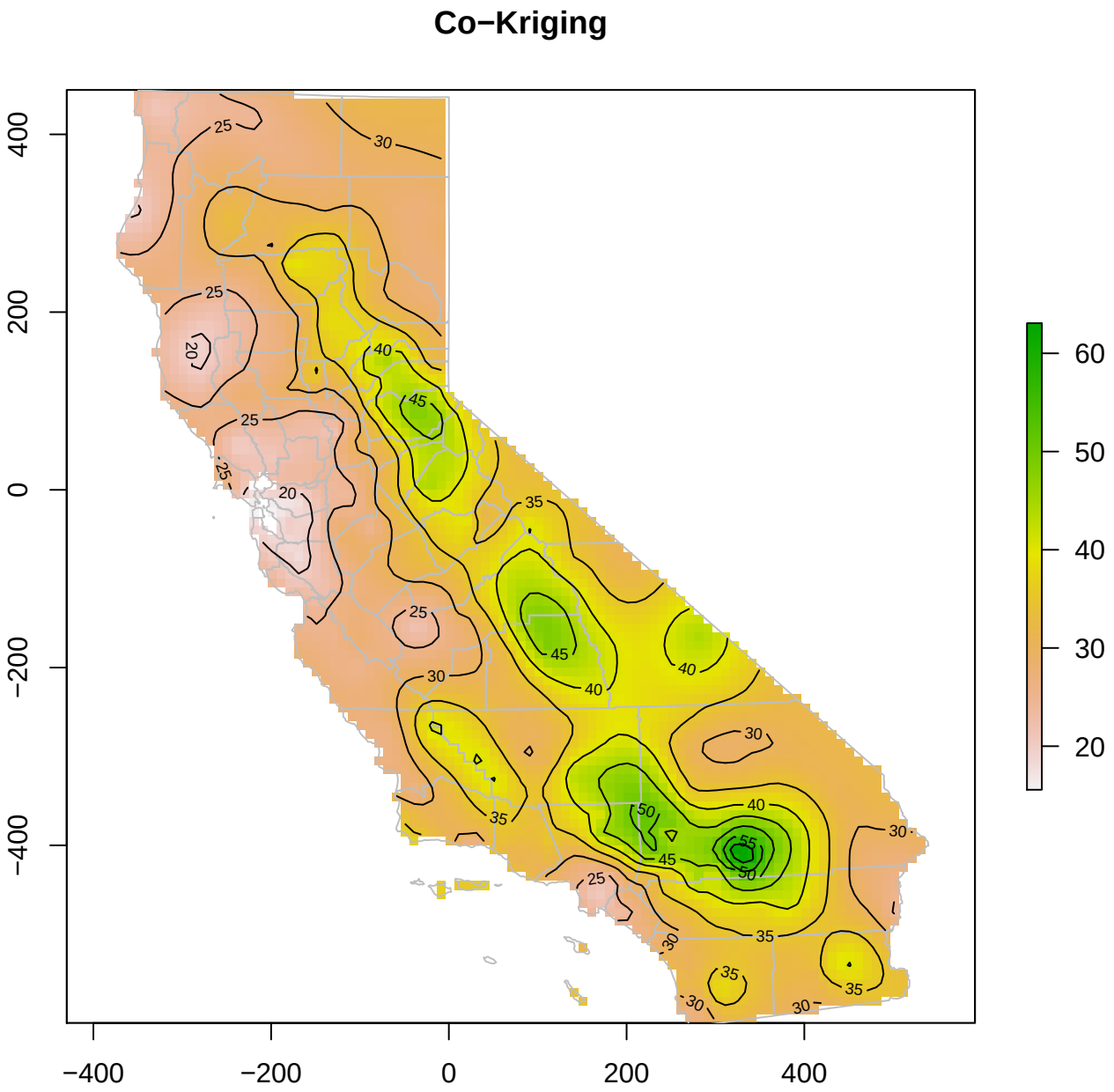

```
# co-kriging, WARNING: this may take a few minutes!
ckp <- predict(fvmc, g)
## Linear Model of Coregionalization found. Good.
## [using ordinary cokriging]
spplot(ckp)
```




```
# plot nicely
ck <- brick(ckp)
ck <- mask(ck, ca_ta)
plot(ck)
```



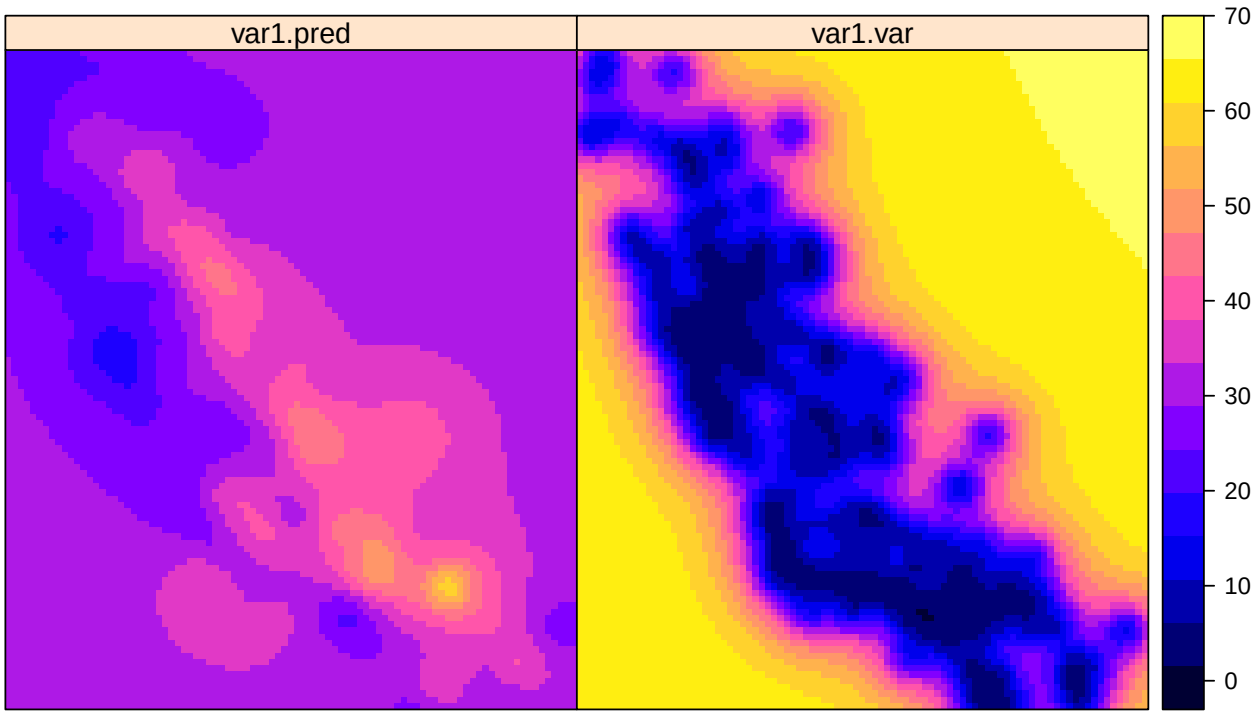

```
plot(ck$OZDLYAV.pred, main = "Co-Kriging")
plot(ca_ta, border = "gray", add = TRUE)
contour(ck$OZDLYAV.pred, add = TRUE, nlevels = 10)
```



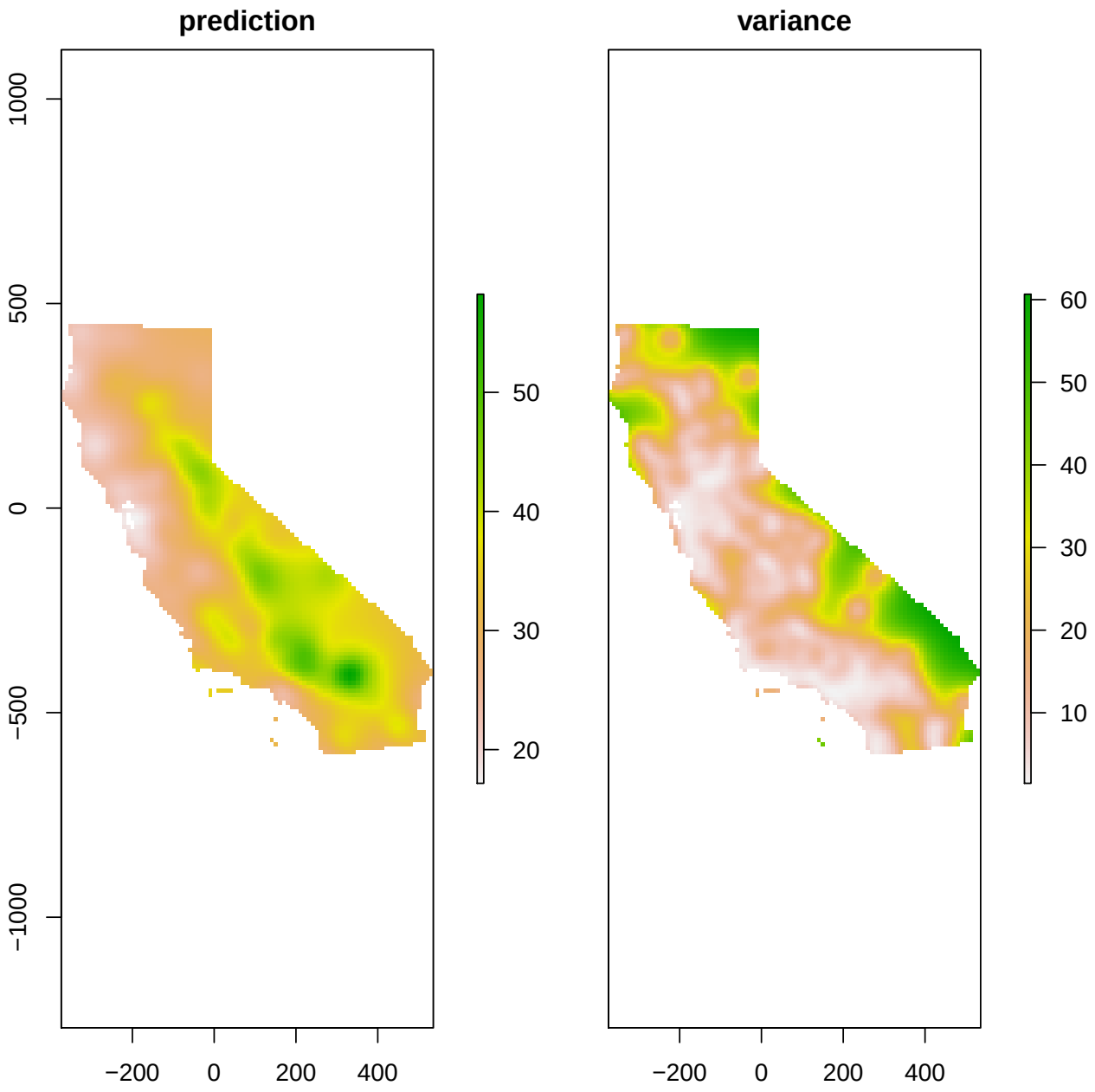
7.7 Block Kriging –for rectangular or irregular blocks

Use block kriging to smoothe out the surface, and have a less computationally expensive model.
block size: a vector with 1, 2 or 3 values containing the size of a rectangular in x-, y- and z-dimension respectively (0 if not set), or a data frame with 1, 2 or 3 columns, containing the points that discretize the block in the x-, y- and z-dimension to define irregular blocks relative to (0,0) or (0,0,0)—see also the details section of predict.gstat.

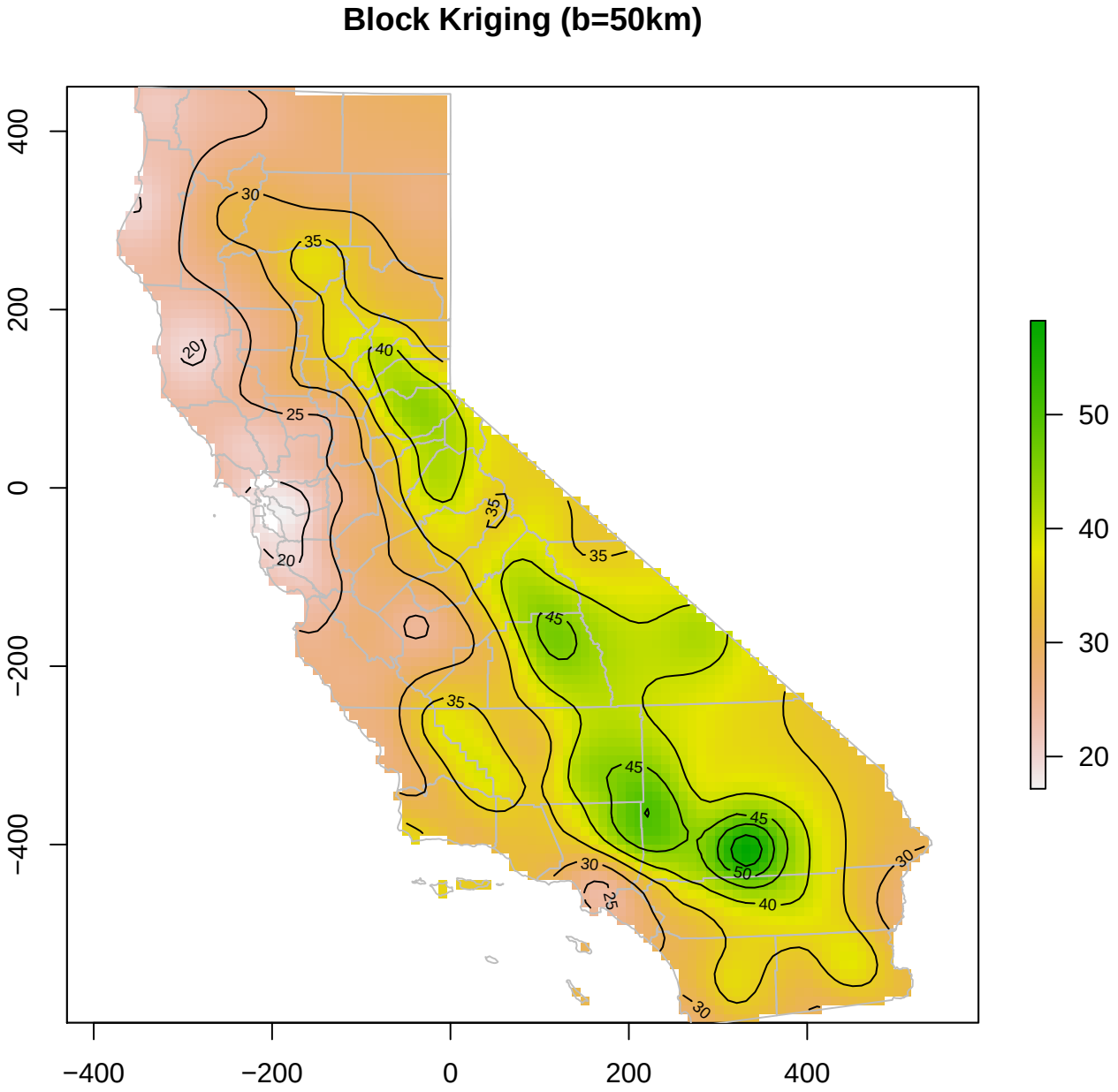
```
# let's try smaller blocks first
bkp_small <- krige(formula = OZDLYAV ~ 1, locations = aq_ta, g, model = fve, block = c(50, 50))
## [using ordinary kriging]
spplot(bkp_small)
```



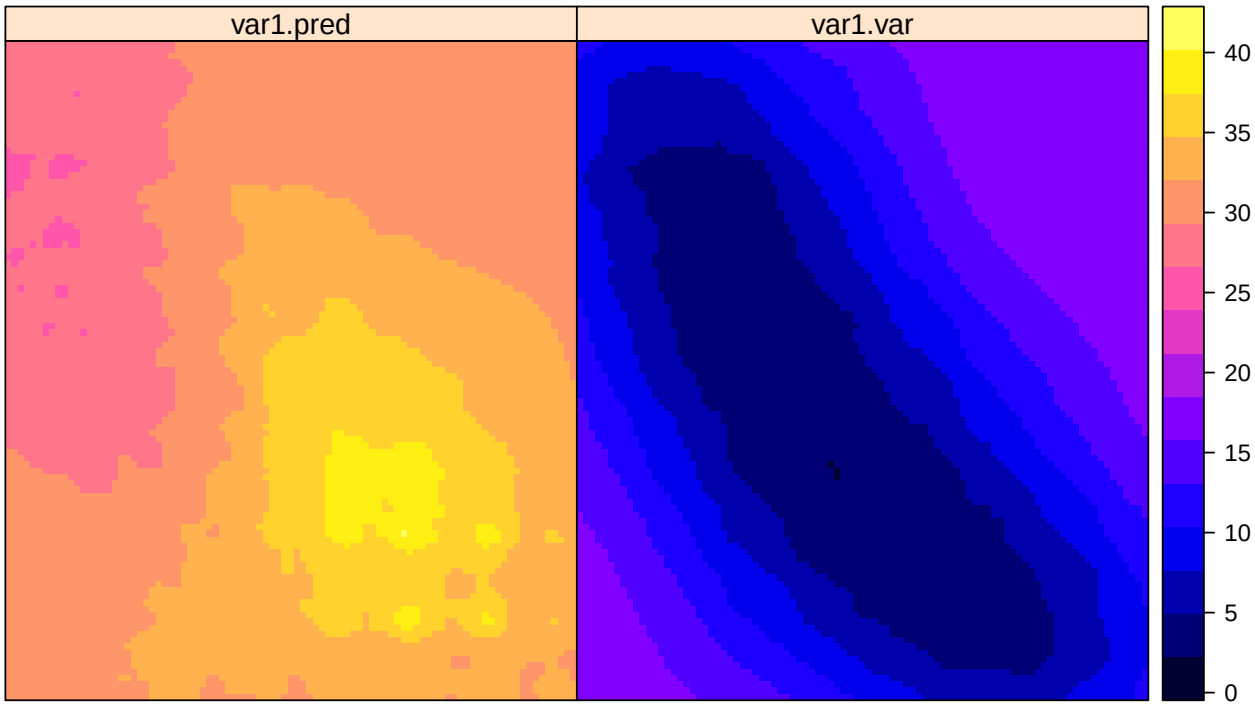

```
# plot nicely
bk_small <- brick(bkp_small)
bk_small <- mask(bk_small, ca_ta)
names(bk_small) <- c("prediction", "variance")
plot(bk_small)
title(main = "Block Kriging (b=50km)", line = 6)
```



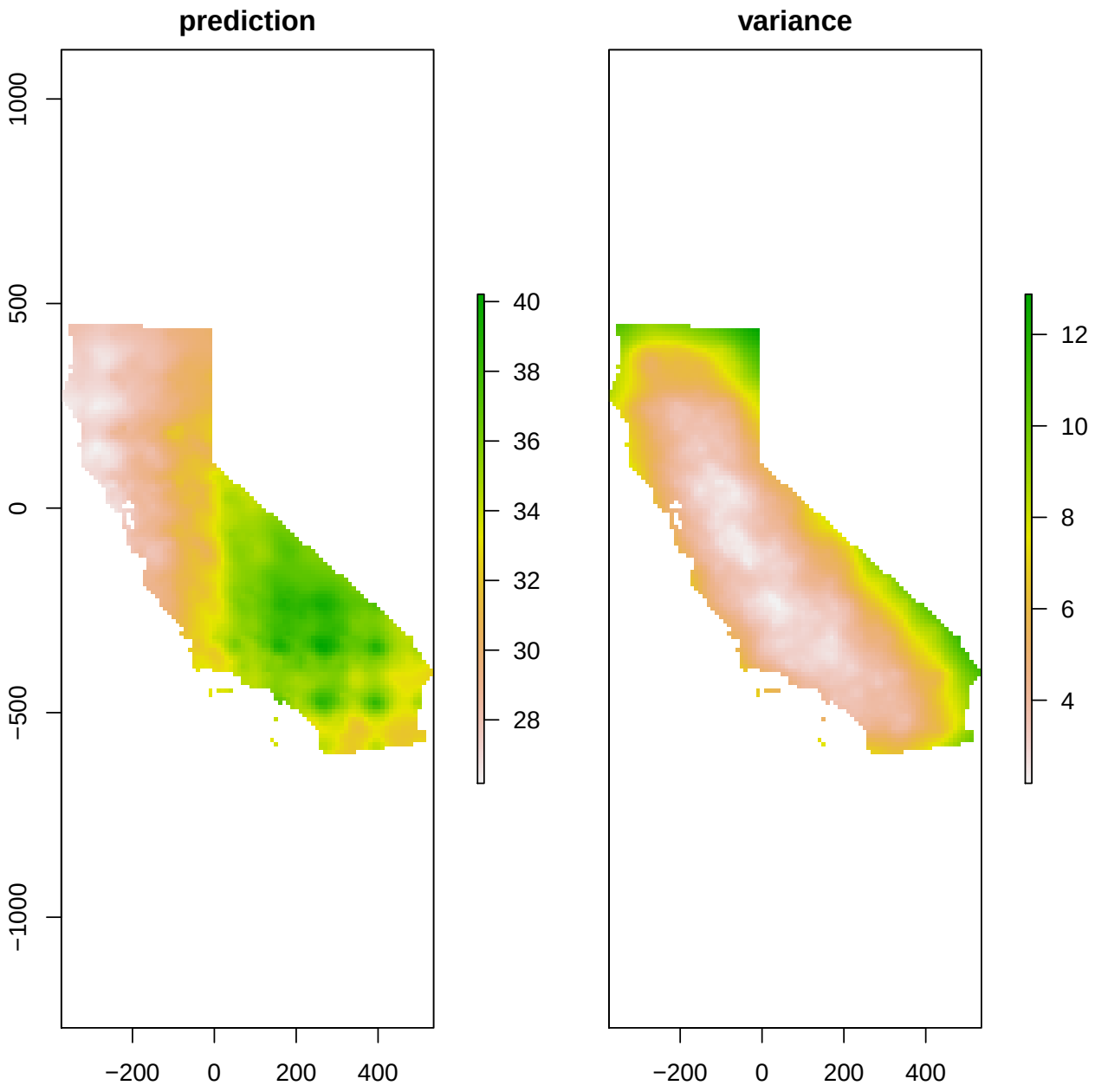

```
plot(bk_small$prediction, main = "Block Kriging (b=50km)")
plot(ca_ta, border = "gray", add = TRUE)
contour(bk_small$prediction, add = TRUE, nlevels = 10)
```



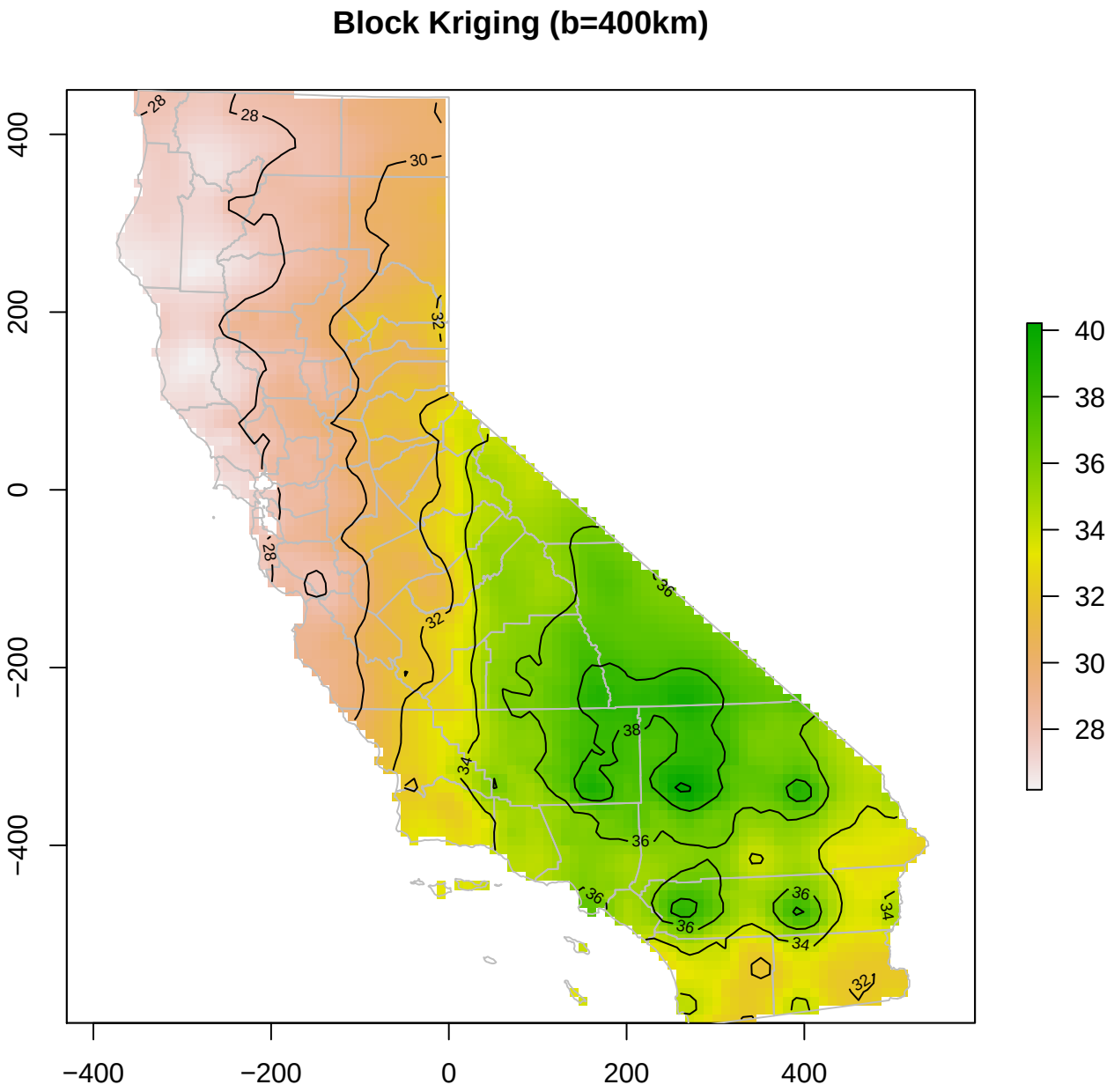

```
# let's try larger blocks
bkp_large <- krige(formula = OZDLYAV ~ 1, locations = aq_ta, g, model = fve, block = c(400, 400))
## [using ordinary kriging]
spplot(bkp_large)
```




```
# plot nicely
bk_large <- brick(bkp_large)
bk_large <- mask(bk_large, ca_ta)
names(bk_large) <- c("prediction", "variance")
plot(bk_large)
title(main = "Block Kriging (b=400km)", line = 6)
```




```
plot(bk_large$prediction, main = "Block Kriging (b=400km)")
plot(ca_ta, border = "gray", add = TRUE)
contour(bk_large$prediction, add = TRUE, nlevels = 10)
```

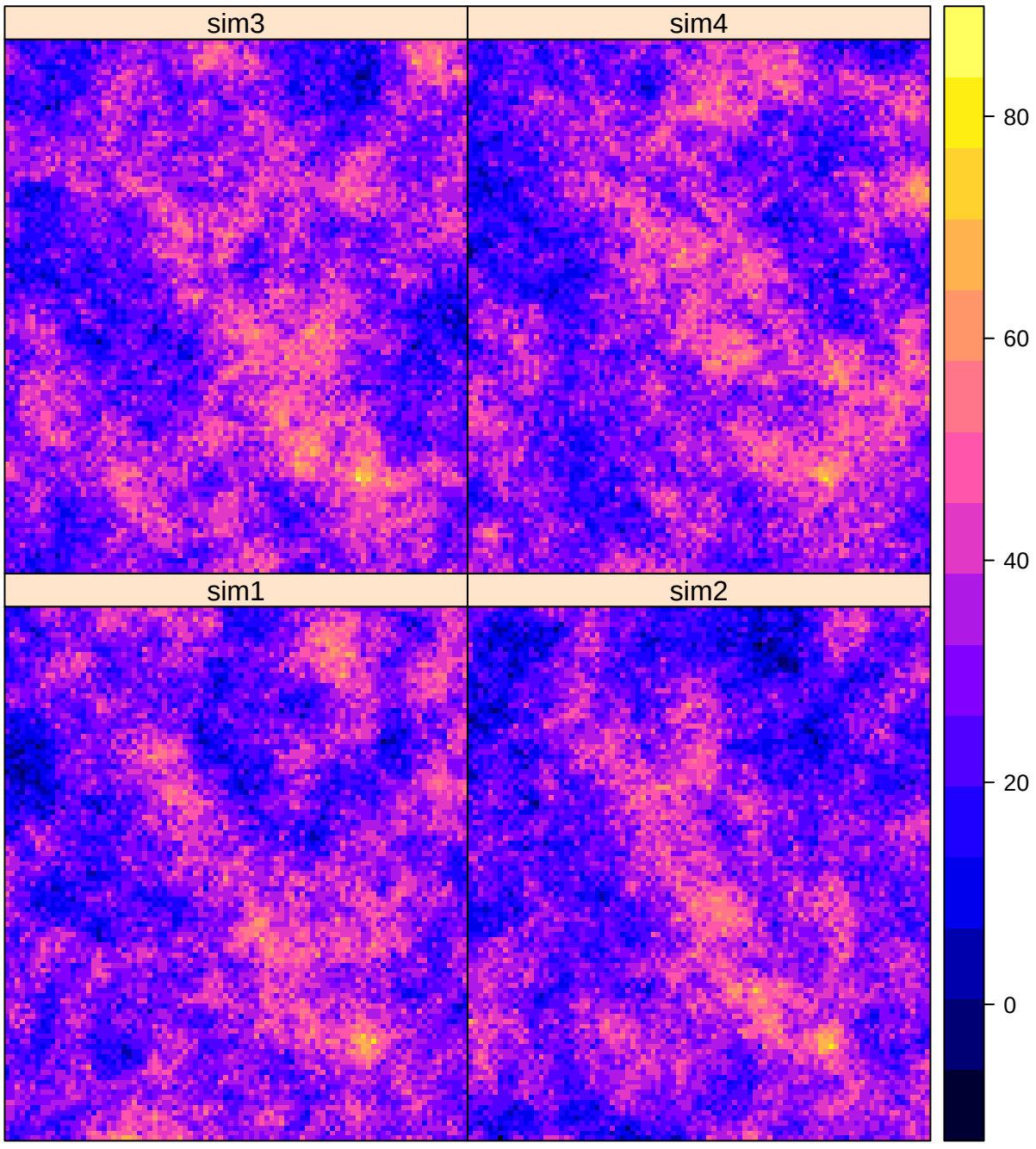


8.0 Sequential Gaussian Simulation

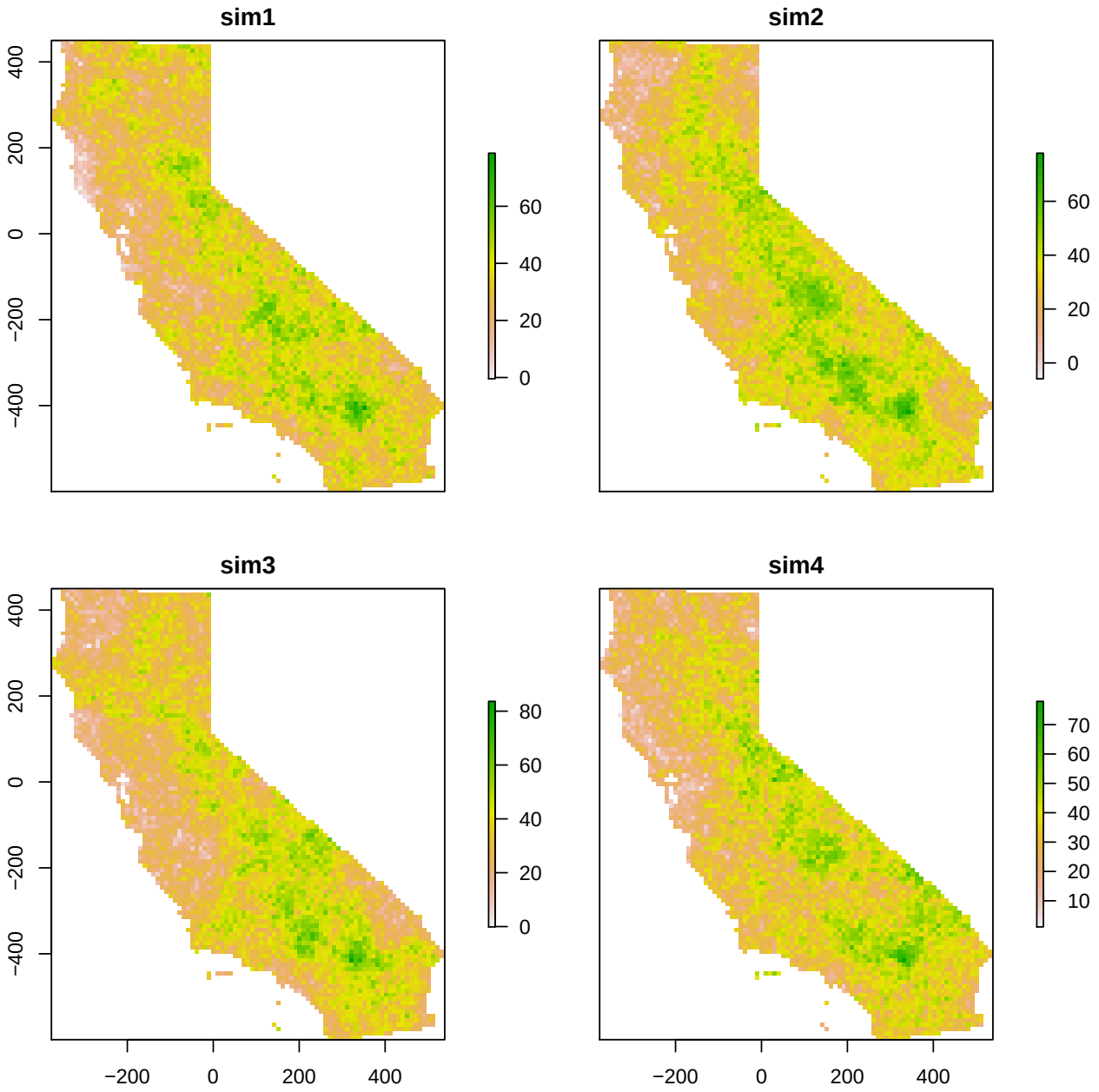
Sequential Gaussian simulation (SGS) simulates continuous variables, such as our ozone concentration data. Sequential indicator simulation (SIS) simulates discrete variables, such as an ozone indicator variable that splits the ozone concentration to high and low values based on a threshold value.
nsim: integer; if set to a non-zero value, conditional simulation is used instead of kriging interpolation. For this, sequential Gaussian or indicator simulation is used (depending on the value of indicators), following a single random path through the data.

```
# SGS
condsimp <- krige(formula = OZDLYAV ~ 1, locations = aq_ta, g, model = fve, nmax = 30, nsim = 4)
## drawing 4 GLS realisations of beta...
## [using conditional Gaussian simulation]
```

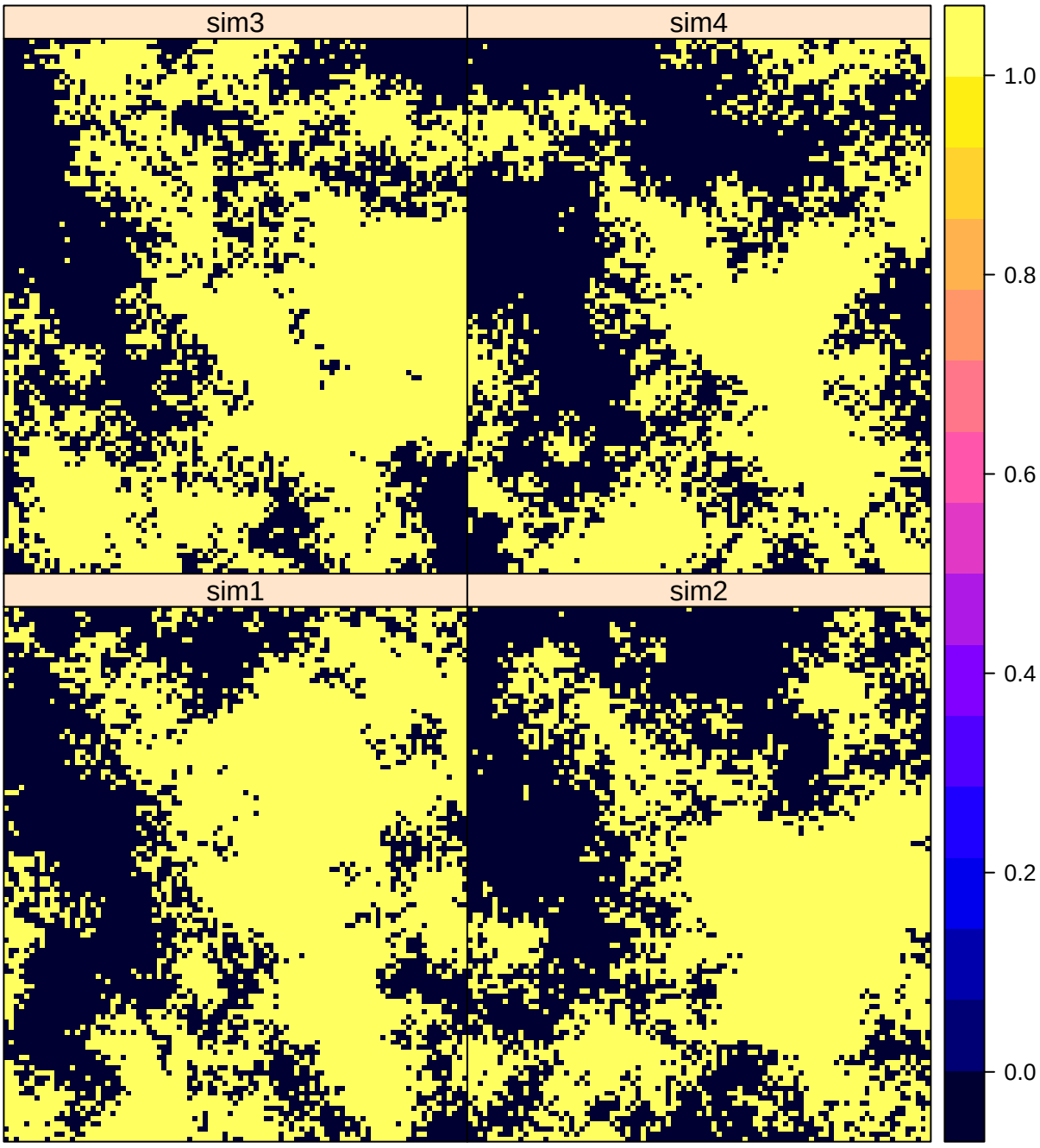
```
spplot(condsimp, main = "four conditional simulations")
```



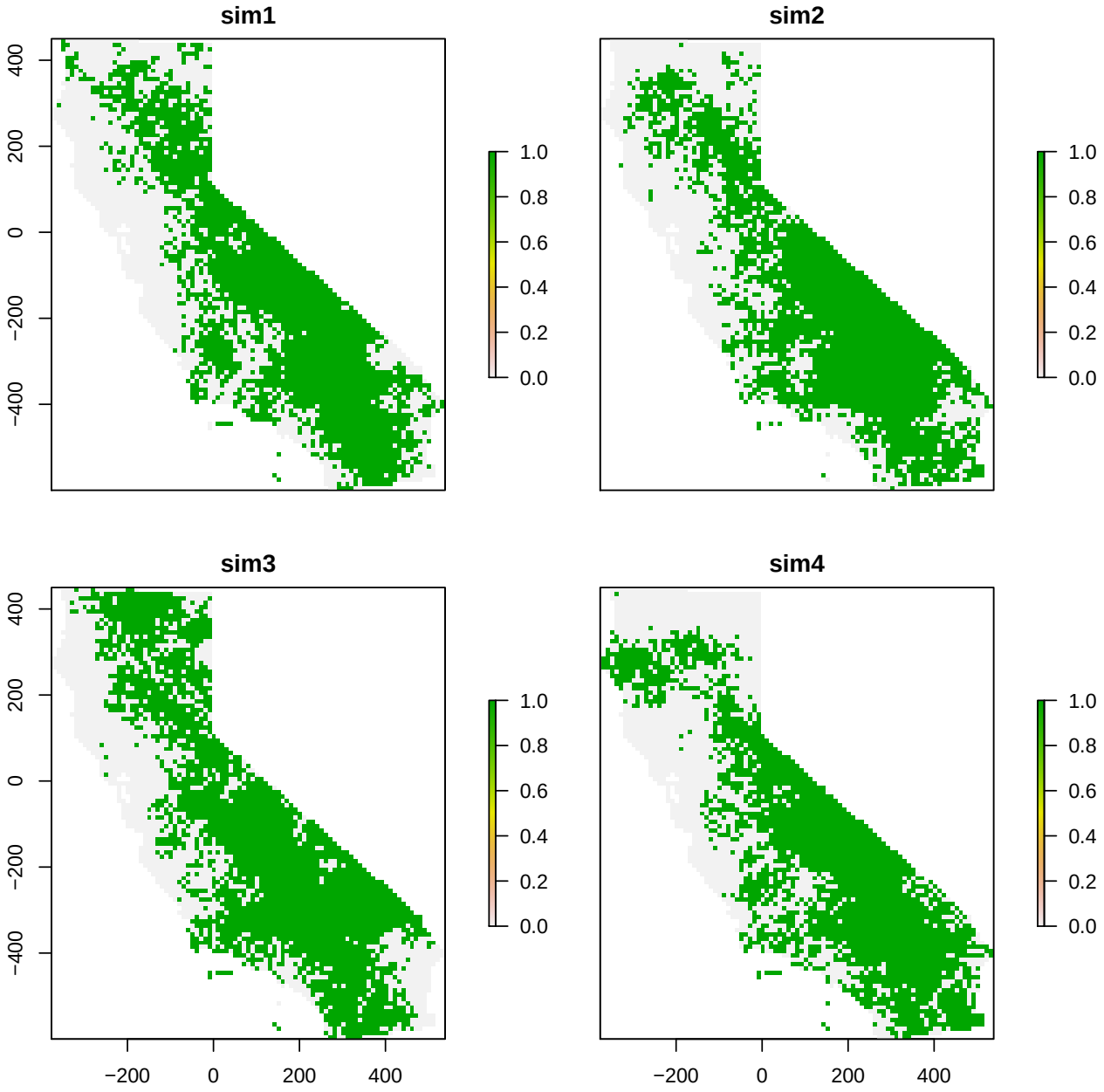

```
# plot nicely
condsim <- brick(condsimp)
condsim <- mask(condsim, ca_ta)
plot(condsim)
```




```
# SIS, note that the model should be set to the fit to the indicator variogram!
condsimp2 <- krige(formula = I(OZDLYAV > 30) ~ 1, locations = aq_ta, g, model = fvis, nmax = 30, nsim =
## drawing 4 GLS realisations of beta...
## [using conditional indicator simulation]
spplot(condsimp2, main = "four conditional simulations")
```




```
# plot nicely
condsim2 <- brick(condsimp2)
condsim2 <- mask(condsim2, ca_ta)
plot(condsim2)
```



9.0 Comparing Models

To recap, the models developed are: Average Model(am), Inverse Distance Weighted (idw), Simple Kriging (sk), Ordinary Kriging (ok), Unversal Kriging with a Linear Trend (ukl), Unversal Kriging with a Quadratic Trend (ukq), Indicator Kriging (ik), Co-Kriging (ck), Block Kriging (bk).

note: I left co-kriging out of the cross validation, because the code woould take a looooong time to run.

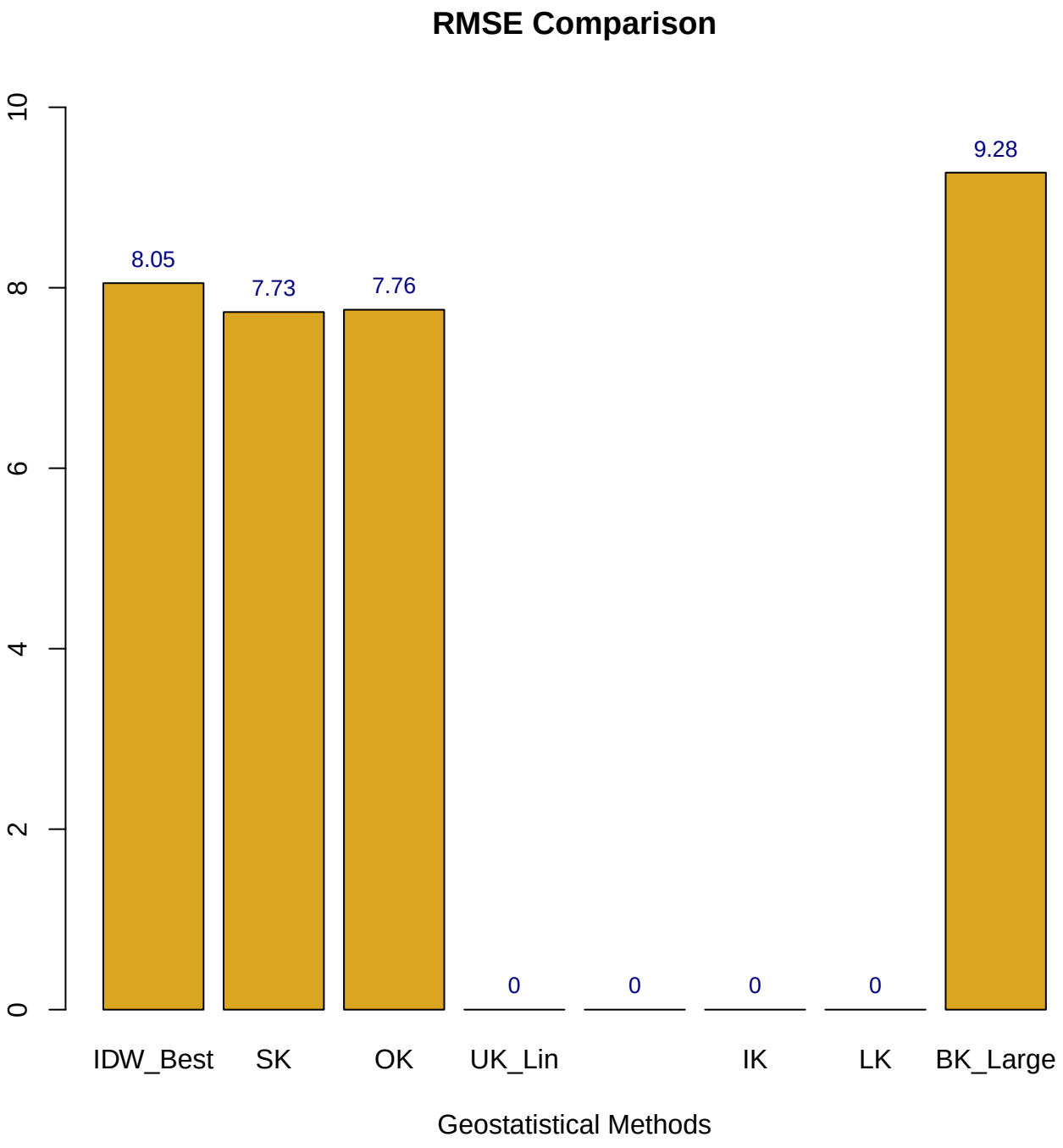
9.1 Comparing RMSE and NSE

Use k-fold crossvalidation to evaluate the models.

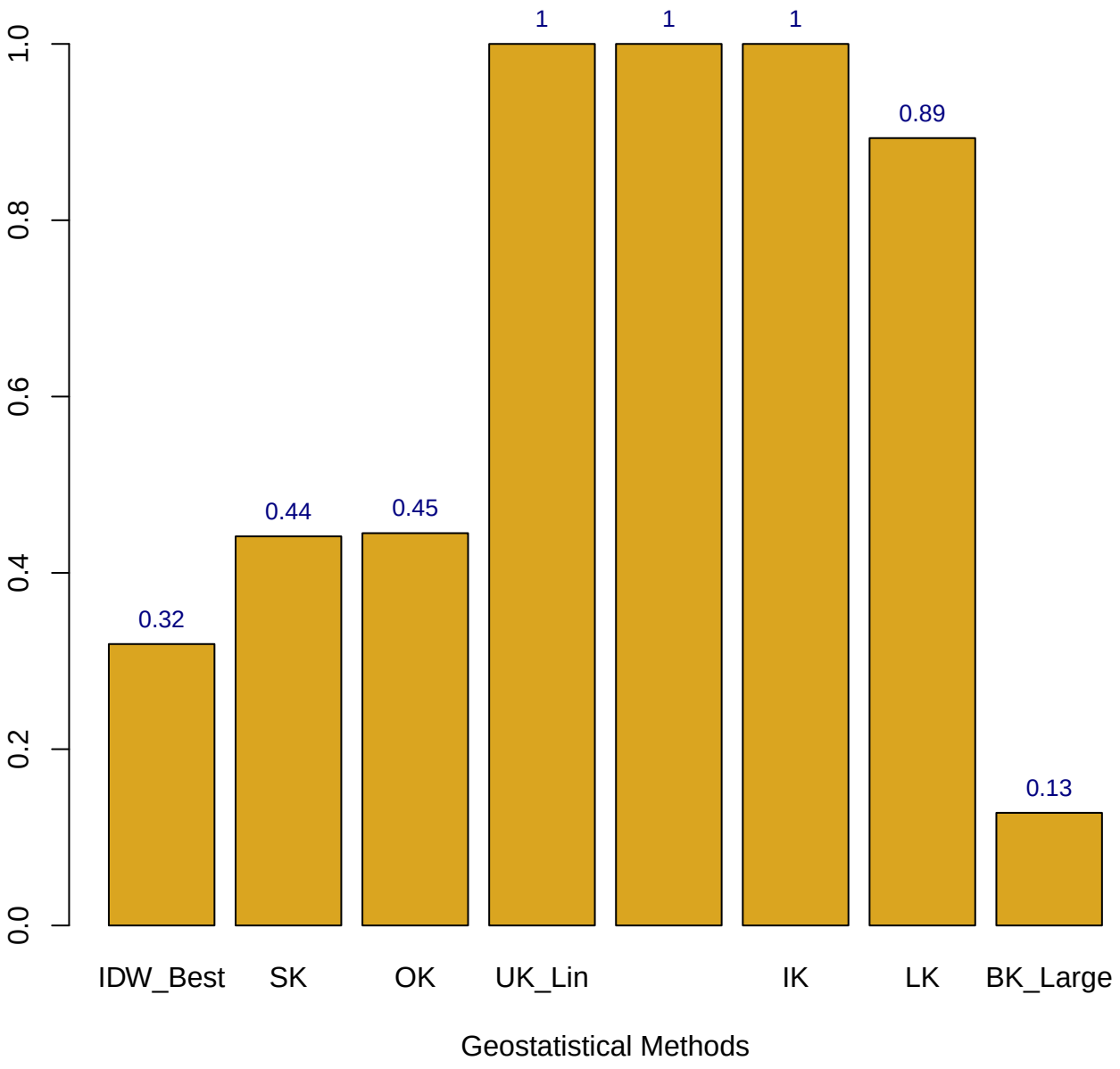
```r
library(dismo)

nfolds <- 5
k <- kfold(aq_ta, nfolds)

# define vectors to be populated later
ensrmse <- idwrmse <- skrmse <- okrmse <- uklrmse <- ukqrmse <- ikrmse <- lkrmse <- bk_lrmse <- rep(NA,
ensrsq <- idwrsq <- skrsq <- okrsq <- uklrsq <- ukqrsq <- ikrsq <- lkrsq <- bk_lrsq <- rep(NA, 5)

# cross validation loop
for (i in 1:nfolds) {
    test <- aq_ta[k != i, ]
    train <- aq_ta[k == i, ]

    # idw_best
    m <- gstat(formula = OZDLYAV ~ 1, locations = train, nmax = opt$par[1], set = list(idp = opt$par[2])
    p1 <- predict(m, newdata = test, debug.level = 0)$var1.pred
    idwrmse[i] <- RMSE(test$OZDLYAV, p1)
    idwrsq[i] <- R2(test$OZDLYAV, p1)

    # sk
    m <- gstat(formula = OZDLYAV ~ 1, locations = train, model = fve, beta = mean(aq_ta$OZDLYAV))
    p2 <- predict(m, newdata = test, debug.level = 0)$var1.pred
    skrmse[i] <- RMSE(test$OZDLYAV, p2)
    skrsq[i] <- R2(test$OZDLYAV, p2)

    # ok
    m <- gstat(formula = OZDLYAV ~ 1, locations = train, model = fve)
    p3 <- predict(m, newdata = test, debug.level = 0)$var1.pred
    okrmse[i] <- RMSE(test$OZDLYAV, p3)
    okrsq[i] <- R2(test$OZDLYAV, p3)

    # ukl
    m <- gstat(formula = OZDLYAV ~ x + y, locations = aq_ta, model = fve)
    p4 <- predict(m, newdata = test, debug.level = 0)$var1.pred
    uklrmse[i] <- RMSE(test$OZDLYAV, p4)
    uklrsq[i] <- R2(test$OZDLYAV, p4)

    # ukq
    m <- gstat(formula = OZDLYAV ~ x + y + I(x * y) + I(x^2) + I(y^2), locations = aq_ta, model = fve)
    p5 <- predict(m, newdata = test, debug.level = 0)$var1.pred
    ukqrmse[i] <- RMSE(test$OZDLYAV, p5)
    ukqrsq[i] <- R2(test$OZDLYAV, p5)

    # ik
    m <- gstat(formula = I(OZDLYAV > 30) ~ 1, locations = aq_ta, model = fvis, nmax = 30)
    p6 <- predict(m, newdata = test, debug.level = 0, indicators = TRUE)$var1.pred
    test$indicator <- ifelse(test$OZDLYAV > 30, 1, 0)
    ikrmse[i] <- RMSE(test$indicator, p6)
    ikrsq[i] <- R2(test$indicator, p6)

    # lk
    m <- gstat(formula = OZDLYAV ~ 1, locations = aq_ta, model = fve, nmin = 20, nmax = 100, maxdist =
```

```
    p7 <- predict(m, newdata = test, debug.level = 0)$var1.pred
    lkrmse[i] <- RMSE(test$OZDLYAV, p7)
    lkrsq[i] <- R2(test$OZDLYAV, p7)

    # bk
    m <- gstat(formula = OZDLYAV ~ 1, locations = aq_ta, model = fve)
    p8 <- predict(m, newdata = test, debug.level = 0, block = c(400, 400))$var1.pred
    bk_lrmse[i] <- RMSE(test$OZDLYAV, p8)
    bk_lrsq[i] <- R2(test$OZDLYAV, p8)

    w <- c(idwrsq[i], skrsq[i], okrsq[i], uklrsq[i], ukqrsq[i], ikrsq[i], lkrsq[i], bk_lrsq[i])
    weights <- w/sum(w)

    # ensemble model
    ensemble <- p1 * weights[1] + p2 * weights[2] + p3 * weights[3] + p4 * weights[4] + p5 * weights[5]
        p7 * weights[7] + p8 * weights[8]
    ensrmse[i] <- RMSE(test$OZDLYAV, ensemble)
    ensrsq[i] <- R2(test$OZDLYAV, ensemble)
}

idw_rmse <- mean(idwrmse)
sk_rmse <- mean(skrmse)
ok_rmse <- mean(okrmse)
ukl_rmse <- mean(uklrmse)
ukq_rmse <- mean(ukqrmse)
ik_rmse <- mean(ikrmse)
lk_rmse <- mean(lkrmse)
bk_lrmse <- mean(bk_lrmse)

idw_rsq <- mean(idwrsq)
sk_rsq <- mean(skrsq)
ok_rsq <- mean(okrsq)
ukl_rsq <- mean(uklrsq)
ukq_rsq <- mean(ukqrsq)
ik_rsq <- mean(ikrsq)
lk_rsq <- mean(lkrsq)
bk_lrsq <- mean(bk_lrsq)

rms <- c(idw_rmse, sk_rmse, ok_rmse, ukl_rmse, ukq_rmse, ik_rmse, lk_rmse, bk_lrmse)
rsq <- c(idw_rsq, sk_rsq, ok_rsq, ukl_rsq, ukq_rsq, ik_rsq, lk_rsq, bk_lrsq)
names(rms) <- names(rsq) <- c("IDW_Best", "SK", "OK", "UK_Lin", "UK_Quad", "IK", "LK", "BK_Large")
rms
##     IDW_Best           SK           OK       UK_Lin      UK_Quad
## 8.051138e+00 7.730628e+00 7.756422e+00 2.678536e-14 2.631330e-14
##           IK           LK     BK_Large
## 1.561970e-16 5.152618e-15 9.275791e+00
rsq
##  IDW_Best        SK        OK    UK_Lin   UK_Quad        IK        LK
## 0.3192355 0.4413978 0.4450255 1.0000000 1.0000000 1.0000000 0.8931859
##  BK_Large
## 0.1277716

# barplot the RMSE, store and print the rmse on the bars
```

```r
ylim <- c(0, 1.1 * max(rms))
xx <- barplot(rms, main = "RMSE Comparison", xlab = "Geostatistical Methods", ylim = ylim, col = "golde
text(x = xx, y = rms, label = round(rms, digits = 2), pos = 3, cex = 0.8, col = "navy")
```



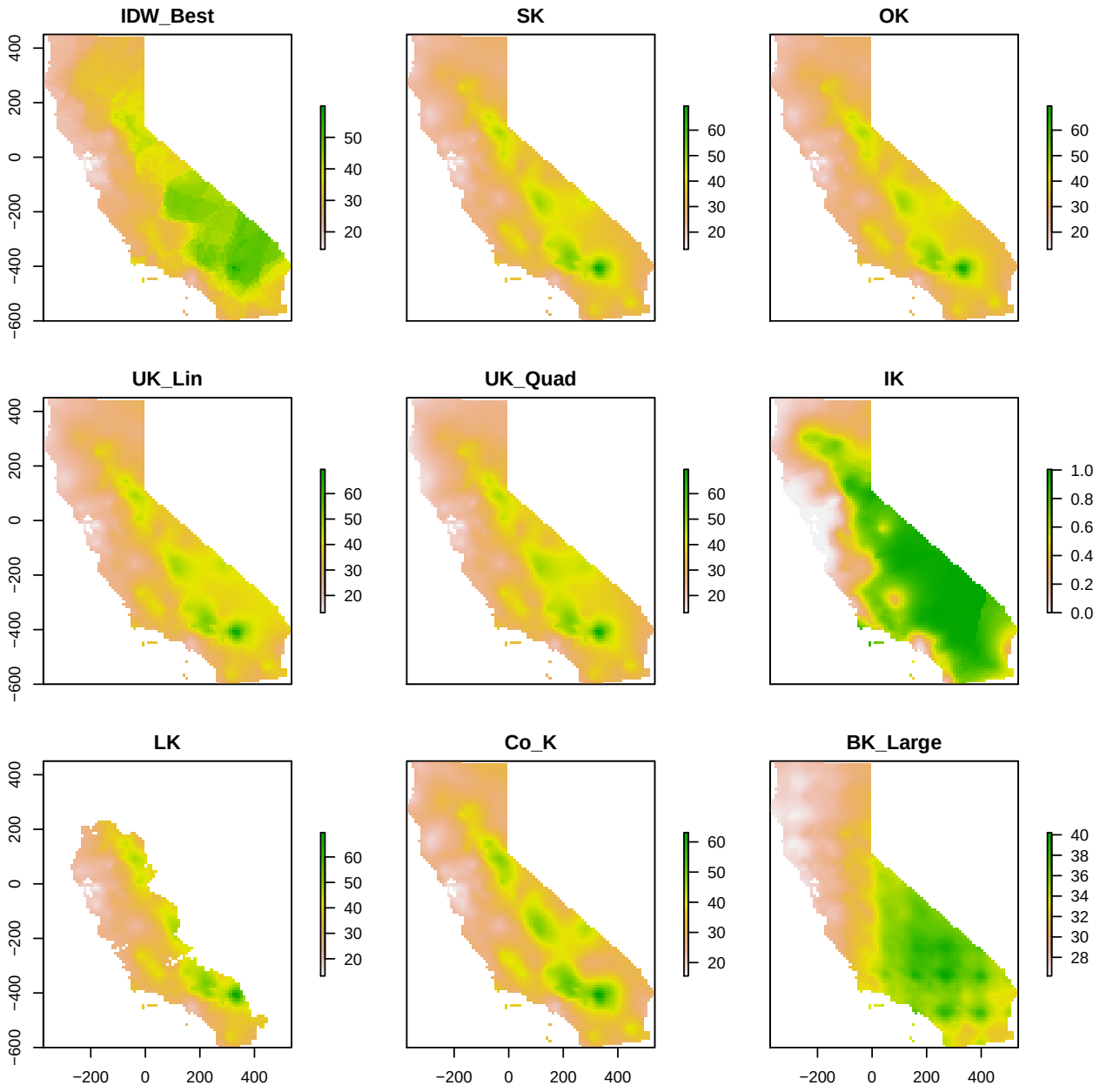

```r
# barplot the RSQ, store and print the rmse on the bars
ylim <- c(0, 1.1 * max(rsq))
xx <- barplot(rsq, main = "R^2 Comparison", xlab = "Geostatistical Methods", ylim = ylim, col = "goldeni
text(x = xx, y = rsq, label = round(rsq, digits = 2), pos = 3, cex = 0.8, col = "navy")
```



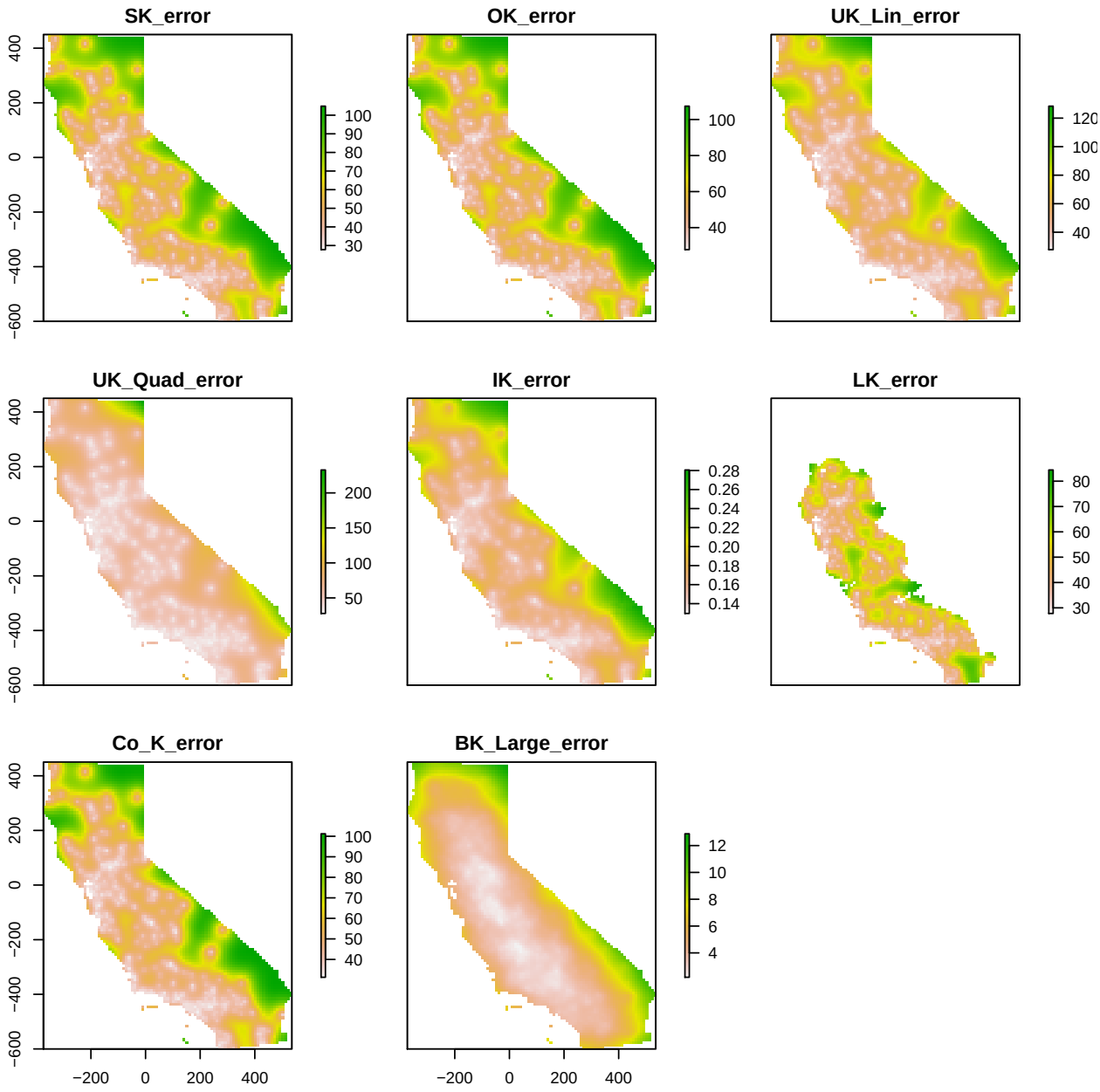
9.2 Comparing Plots

Use plots to identify spatial differences and maybe go back to tweek model parameters.

```
# compare prediction plots
s <- stack(idw_best, sk[[1]], ok[[1]], ukl[[1]], ukq[[1]], ik[[1]], lk[[1]], ck[[1]], bk_large[[1]])
names(s) <- c("IDW_Best", "SK", "OK", "UK_Lin", "UK_Quad", "IK", "LK", "Co_K", "BK_Large")
plot(s)
```




```
# compare errors plots
s_stdev <- stack(sk[[2]], ok[[2]], ukl[[2]], ukq[[2]], ik[[2]], lk[[2]], ck[[2]], bk_large[[2]])
names(s_stdev) <- c("SK_error", "OK_error", "UK_Lin_error", "UK_Quad_error", "IK_error", "LK_error", "C
plot(s_stdev)
```

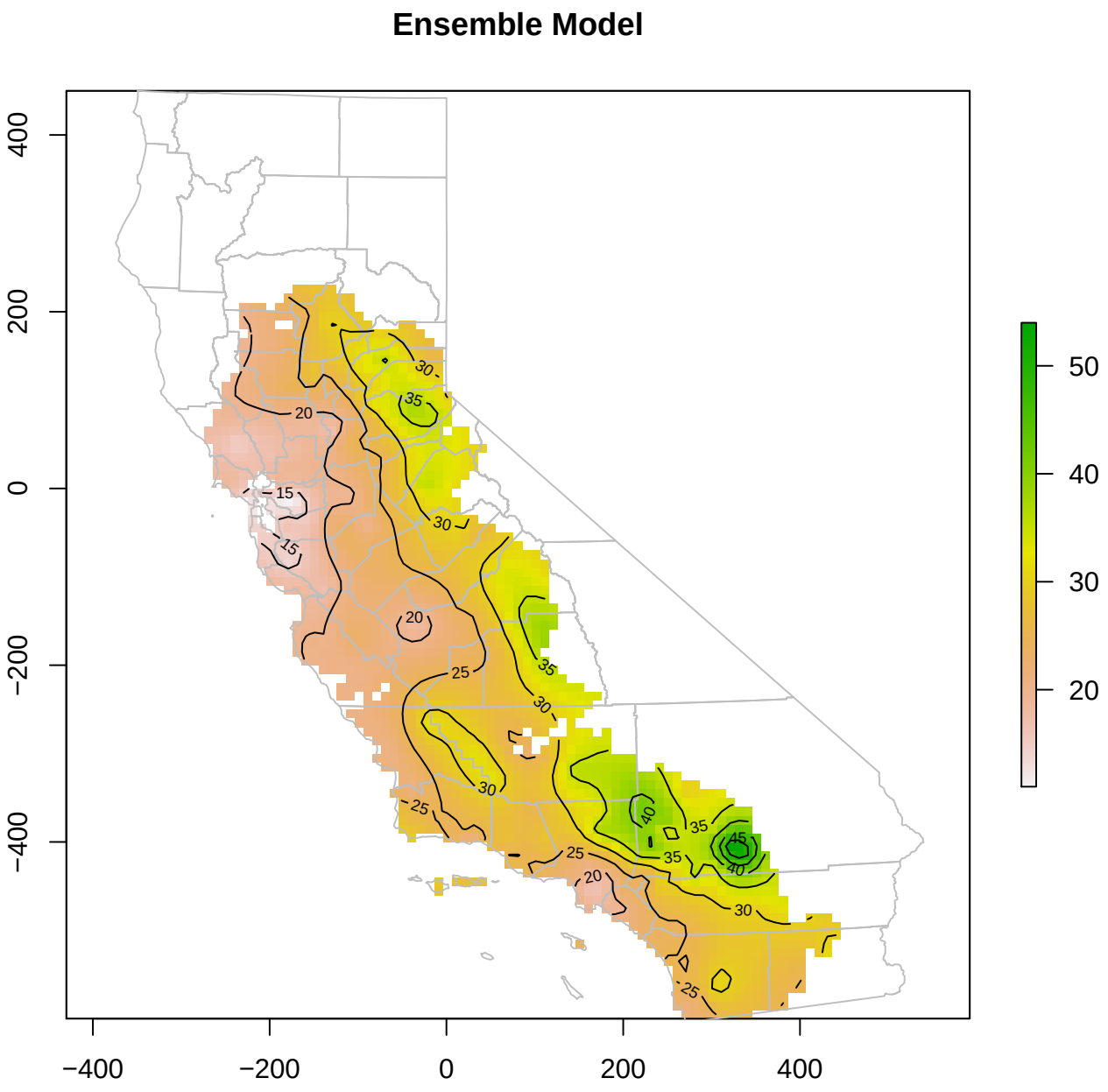


10.0 Ensemble Model

We can combine any or all models by taking a weighted average of all the model predictions with R^2 as the weights.

```
# from cross validation we have:
rmse_e <- mean(ensrmse)
rmse_e
## [1] 6.896585

rsq_e <- mean(ensrsq)
rsq_e
## [1] 0.6851063
```

```r
# weights were calculated in the cv loop
names(weights) <- c("IDW_Best", "SK", "OK", "UK_Lin", "UK_Quad", "IK", "LK", "BK_Large")
weights
##   IDW_Best         SK         OK     UK_Lin    UK_Quad         IK
## 0.05582592 0.05935457 0.05944700 0.20542852 0.20542852 0.20542852
##         LK   BK_Large
## 0.18223413 0.02685281

models <- stack(idw_best, sk[[1]], ok[[1]], ukl[[1]], ukq[[1]], ik[[1]], lk[[1]], bk_large[[1]])
ensemble_model <- sum(models * weights)

plot(ensemble_model, main = "Ensemble Model")
plot(ca_ta, border = "gray", add = TRUE)
contour(ensemble_model, add = TRUE, nlevels = 10)
```



11.0 Transition Probability Markov Chain

Much like the TPROGS software, the spMC lirary is used for modelling of transition probability matrices, and implementing efficient algorithms for improving predictions and simulations of categorical random fields. It deals with both one-dimensional and multidimensional continuous lag models.

Three different fitting methods were implemented in the package. The first is based on the estimates of the main features that characterise the process, the second focuses on the minimisation of the discrepancies between the empirical and theoretical transition probabilities, and the third follows the maximum entropy approach. Once the model parameters are properly estimated, transition probabilities are calculated through the matrixvalued exponential function.

For the estimations of one-dimensional continuous lag models use:
transiogram: Empirical transition probabilities estimation
tpfit: One-dimensional model parameters estimation
tpfit_ils: Iterated least squares method for one-dimensional model parameters estimation
tpfit_me: Maximum entropy method for one-dimensional model parameters estimation
tpfit_ml: Mean length method for one-dimensional model parameters estimation

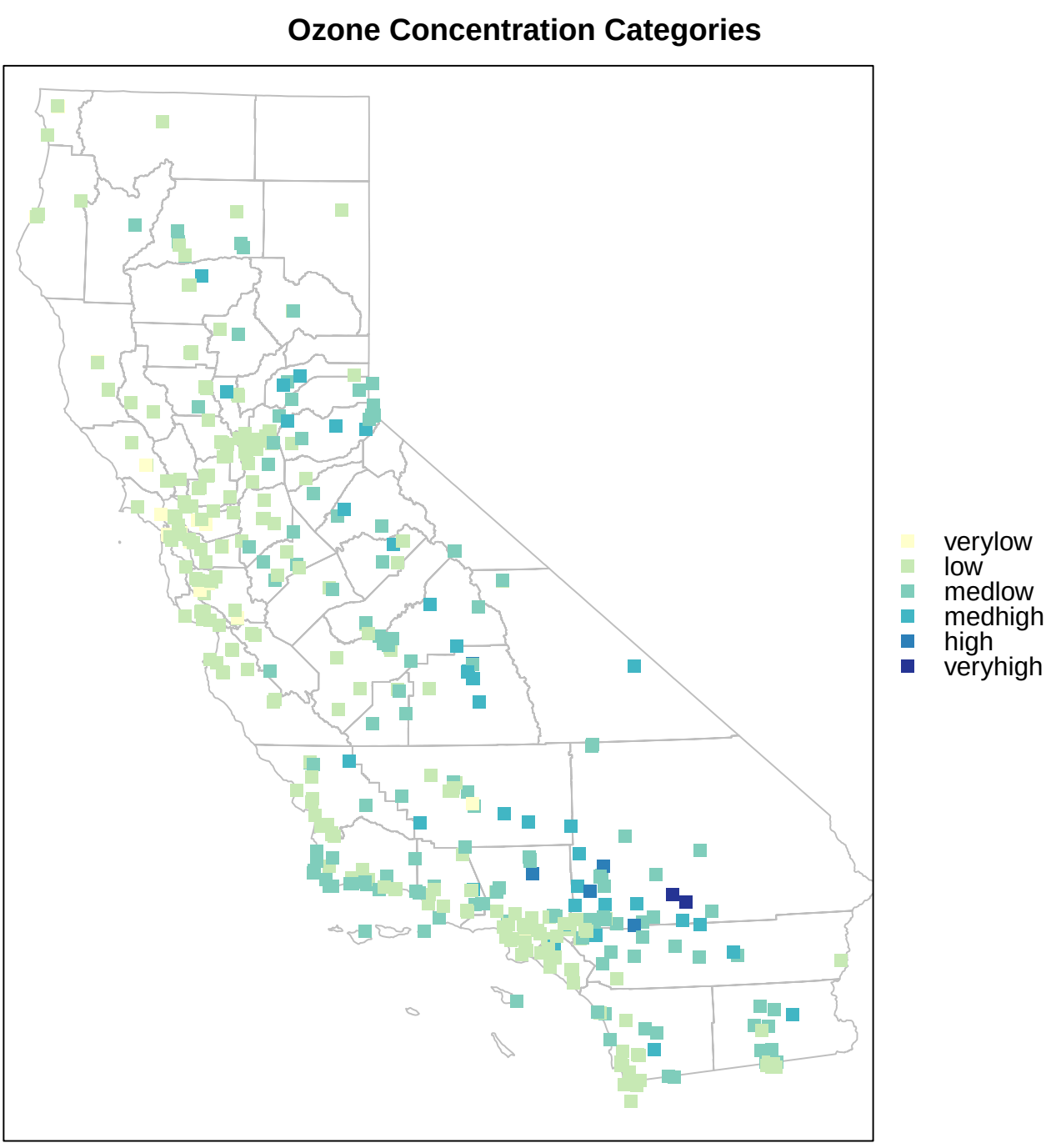
For the estimations of multidimensional continuous lag models use:
pemt: Pseudo-empirical multidimensional transiograms estimation
multi_tpfit: Multidimensional model parameters estimation
multi_tpfit_ils: Iterated least squares method for multidimensional model parameters estimation
multi_tpfit_me: Maximum entropy method for multidimensional model parameters estimation
multi_tpfit_ml: Mean length method for multidimensional model parameters estimation

These transition probabilities are then combined to predict the category in an unsampled position. Three algorithms are used to simulate spatial random fields; those based on the kriging techniques (Carle and Fogg, 1996), those using fixed and random path methods (Li, 2007a; Li and Zhang, 2007), or those using multinomial categorical simulation proposed by Allard et al. (2011).
For categorical spatial random field simulation and prediction use:
sim: Random field simulation
sim_ck: Conditional simulation based on indicator cokriging
sim_ik: Conditional simulation based on indicator kriging
sim_mcs: Multinomial categorical simulation
sim_path: Conditional simulation based on path algorithms

For more information see the spMC package (Sartore, 2016).

```
# change the ozone concentration data into categorical data

# make the groups, you can see the interval categories now in the data set
aq_ta$group <- cut(aq_ta$OZDLYAV, breaks = 6)
table(aq_ta$group)
##
##   (3.38,17]   (17,30.5] (30.5,44.1] (44.1,57.6] (57.6,71.1] (71.1,84.7]
##          19         245         143          38           5           2

# we can lump high and very high, because we don't have enough data, but I will leave it for now

# make the groupings into categories: verylow, low, etc.
aq_ta$categories <- cut(aq_ta$OZDLYAV, breaks = 6, labels = c("verylow", "low", "medlow", "medhigh", "hi
table(aq_ta$categories)
##
##  verylow      low   medlow  medhigh     high veryhigh
##       19      245      143       38        5        2
```

```
# plot the categorical variable now
library(RColorBrewer)
brwCol <- brewer.pal(nlevels(aq_ta$categories), "YlGnBu")
spplot(aq_ta, "categories", col.regions = brwCol, key.space = "right", main = "Ozone Concentration Categ
    ca_ta, col = "grey"), pch = 15)
```



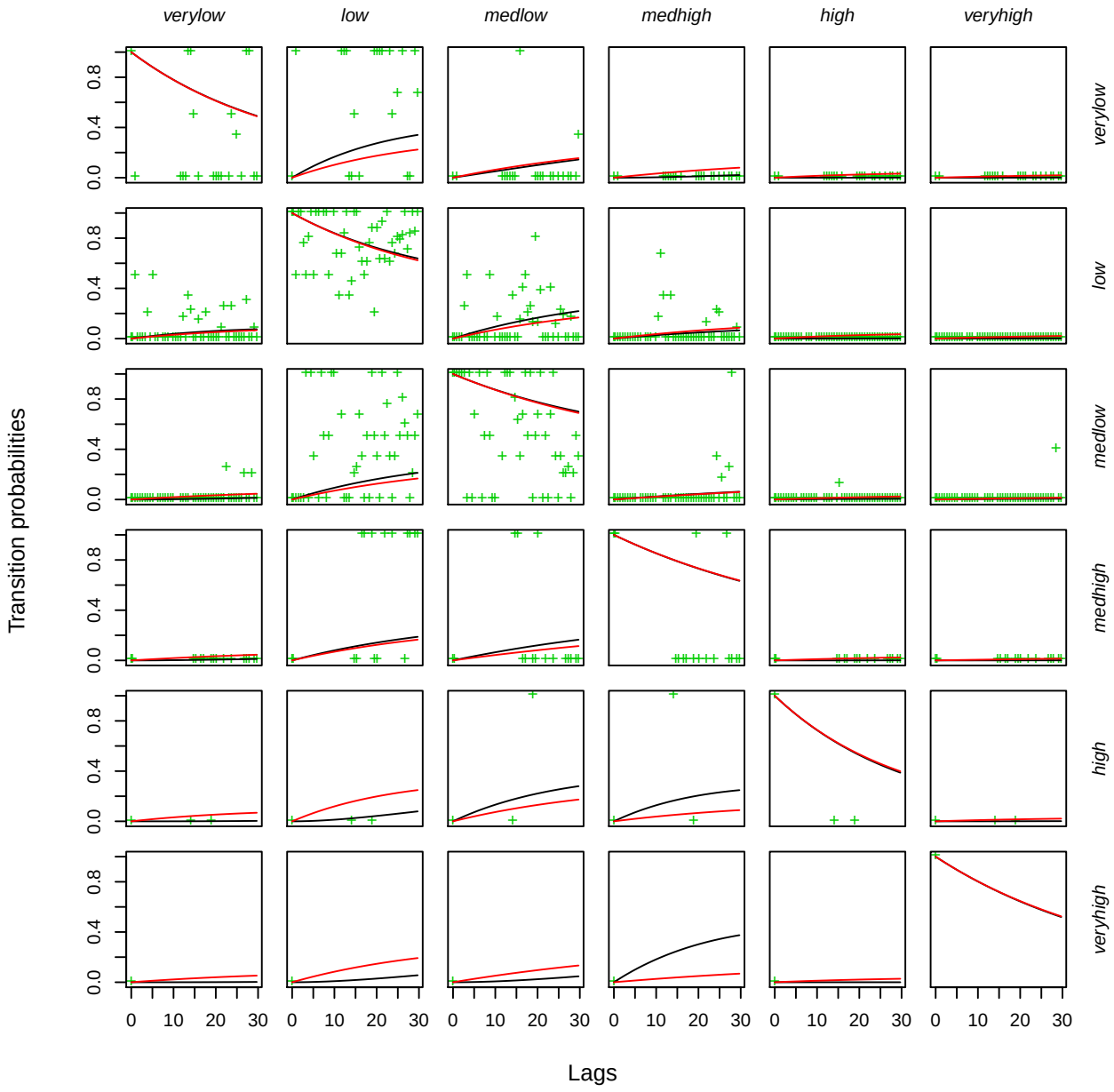

```
# now we can look at the spatial transition probabilities in these categories
library(spMC)
## Warning: package 'spMC' was built under R version 3.2.5
```

```
## Package built without openMP.

# make two transiograms to estimate two transition rate matrices, i.e. the model coefficients, along th
# first, all couples of points along axial directions are chosen such that their lag-length is less tha
# we calculate the empirical transition probabilities for twenty points within the maximum distance
Trg <- list()
Trg[[1]] <- transiogram(aq_ta$categories, aq_ta@coords, max.dist = 30, mpoints = 50, direction = c(1, 0)
Trg[[2]] <- transiogram(aq_ta$categories, aq_ta@coords, max.dist = 30, mpoints = 50, direction = c(0, 1)

# if we want to compare these probabilities with the theoretical one, we first need to estimate two tra
# matrices, i.e. the model coefficients, along the axial directions

# estimate the transition rates through the mean lengths; estimate the parameters of the MC model
RTm <- list()
RTm[[1]] <- tpfit_ml(aq_ta$categories, aq_ta@coords, direction = c(1, 0))
RTm[[2]] <- tpfit_ml(aq_ta$categories, aq_ta@coords, direction = c(0, 1))
RTm
## [[1]]
## Estimated transition rates:
##
##               verylow         low       medlow      medhigh          high
## verylow  -0.025412588  0.02033007  0.005082518  0.000000000  0.0000000000
## low       0.004670153 -0.01868061  0.010897023  0.003113435  0.0000000000
## medlow    0.000000000  0.01085148 -0.014106919  0.002532011  0.0003617159
## medhigh   0.000000000  0.00906826  0.007053091 -0.016121351  0.0000000000
## high      0.000000000  0.00000000  0.016006568  0.016006568 -0.0320131353
## veryhigh  0.000000000  0.00000000  0.000000000  0.022042417  0.0000000000
##               veryhigh
## verylow   0.0000000000
## low       0.0000000000
## medlow    0.0003617159
## medhigh   0.0000000000
## high      0.0000000000
## veryhigh -0.0220424173
##
## Estimated proportions:
##
##     verylow         low      medlow     medhigh        high    veryhigh
## 0.042035398 0.542035398 0.316371681 0.084070796 0.011061947 0.004424779
##
##
## [[2]]
## Estimated transition rates:
##
##                verylow          low       medlow       medhigh
## verylow  -0.0185180771  0.015872638  0.002645440  0.0000000000
## low       0.0044378767 -0.014977834  0.009985223  0.0005547346
## medlow    0.0004667647  0.008401764 -0.013069411  0.0042008820
## medhigh   0.0000000000  0.007996124  0.006542283 -0.0159922483
## high      0.0000000000  0.004130896  0.008261791  0.0000000000
## veryhigh  0.0000000000  0.000000000  0.000000000  0.0222112178
##                   high      veryhigh
## verylow   0.0000000000  0.0000000000
```

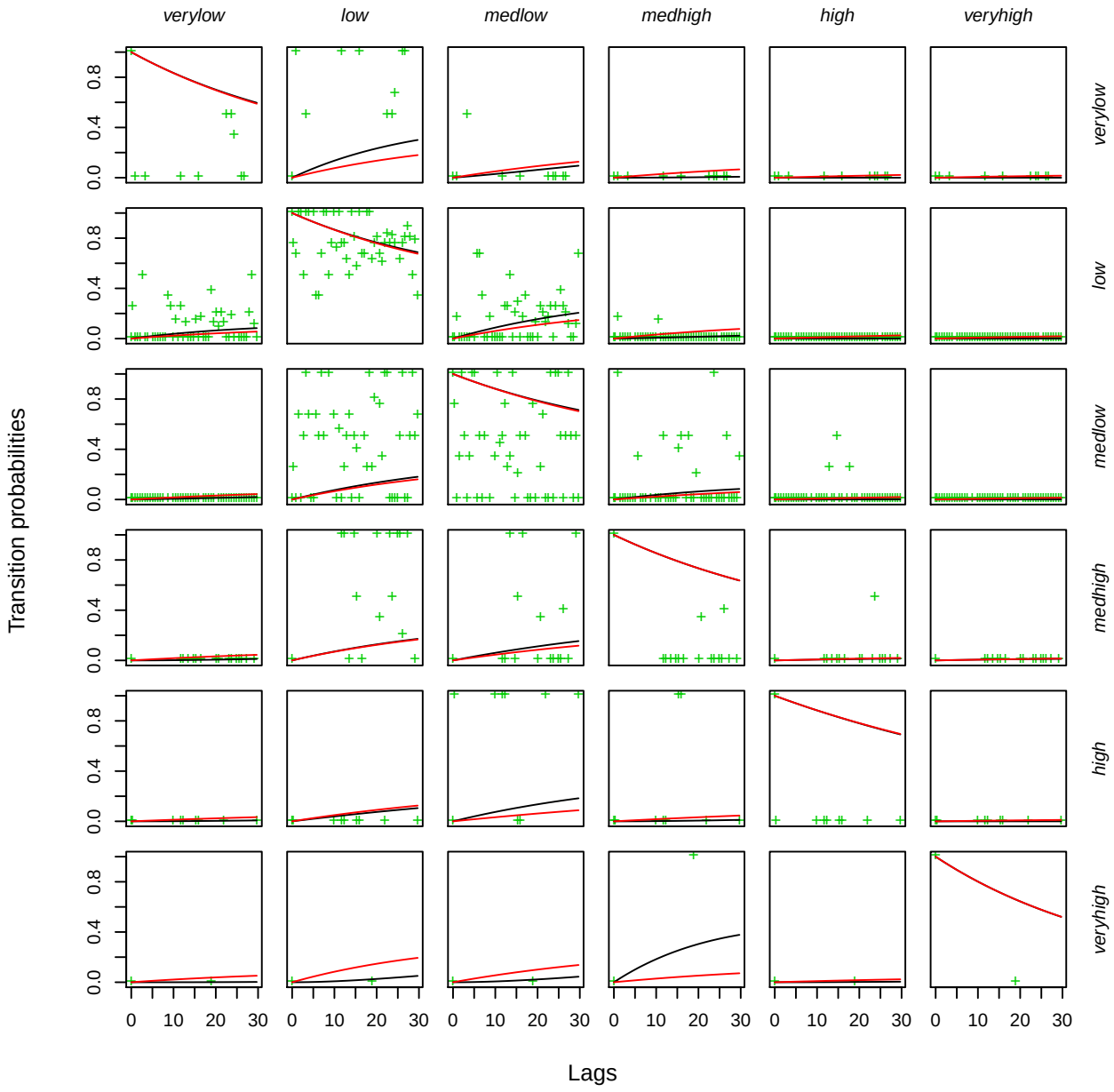
```
## low       0.0000000000  0.0000000000
## medlow    0.0000000000  0.0000000000
## medhigh   0.0007269204  0.0007269204
## high     -0.0123926871  0.0000000000
## veryhigh  0.0000000000 -0.0222112178
##
## Estimated proportions:
##
##     verylow         low      medlow     medhigh        high    veryhigh
## 0.042035398 0.542035398 0.316371681 0.084070796 0.011061947 0.004424779

# estimate the transition rate matrices through the maximum entropy approach
ETm <- list()
ETm[[1]] <- tpfit_me(aq_ta$categories, aq_ta@coords, direction = c(1, 0))
ETm[[2]] <- tpfit_me(aq_ta$categories, aq_ta@coords, direction = c(0, 1))
ETm
## [[1]]
## Estimated transition rates:
##
##               verylow          low       medlow      medhigh          high
## verylow  -0.025412588  0.012101501  0.007024527  0.003595556  0.001774365
## low       0.003588413 -0.018680610  0.007964455  0.004076664  0.002011786
## medlow    0.002071516  0.007920707 -0.014106919  0.002353373  0.001161362
## medhigh   0.002042389  0.007809338  0.004533066 -0.016121351  0.001145033
## high      0.003780123  0.014453787  0.008389952  0.004294458 -0.032013135
## veryhigh  0.002522068  0.009643453  0.005597710  0.002865229  0.001413957
##               veryhigh
## verylow   0.0009166391
## low       0.0010392914
## medlow    0.0005999611
## medhigh   0.0005915253
## high      0.0010948151
## veryhigh -0.0220424173
##
## Estimated proportions:
##
##     verylow         low      medlow     medhigh        high    veryhigh
## 0.042035398 0.542035398 0.316371681 0.084070796 0.011061947 0.004424779
##
##
## [[2]]
## Estimated transition rates:
##
##               verylow          low       medlow      medhigh           high
## verylow  -0.018518077  0.008577427  0.005457316  0.002877398  0.0008709866
## low       0.002659054 -0.014977834  0.006762885  0.003565766  0.0010793551
## medlow    0.001844172  0.007371979 -0.013069411  0.002473017  0.0007485806
## medhigh   0.001929282  0.007712201  0.004906824 -0.015992248  0.0007831281
## high      0.001343483  0.005370502  0.003416937  0.001801598 -0.0123926871
## veryhigh  0.002391467  0.009559762  0.006082318  0.003206934  0.0009707369
##               veryhigh
## verylow   0.0007349496
## low       0.0009107736
```

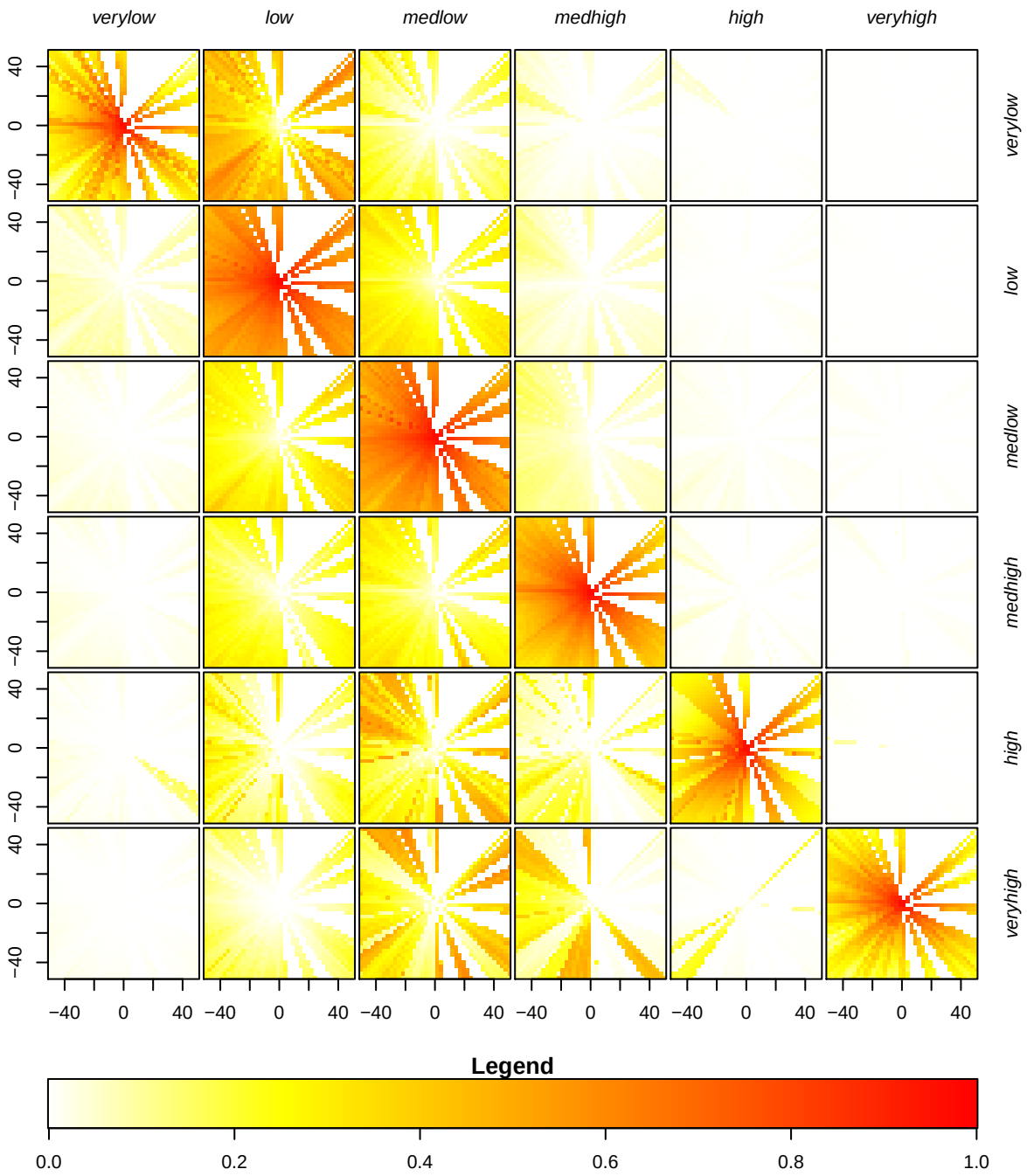
```
## medlow    0.0006316618
## medhigh   0.0006608135
## high      0.0004601670
## veryhigh -0.0222112178
##
## Estimated proportions:
##
##     verylow         low      medlow     medhigh        high    veryhigh
## 0.042035398 0.542035398 0.316371681 0.084070796 0.011061947 0.004424779
```

```
# given the model coefficients, the transition probabilities for some specific lags are calculated
RTr <- list()
ETr <- list()
for (i in 1:2) {
    RTr[[i]] <- predict(RTm[[i]], lags = Trg[[i]]$lags)
    ETr[[i]] <- predict(ETm[[i]], lags = Trg[[i]]$lags)
}

# Since these probabilities are calculated with respect to some fixed directions, i.e. by considering a
# perspective, they can be graphically compared

# a transition probability matrix plot, it shows the probability dynamic related to one-dimensional lag
# specified direction

for (i in 1:2) {
    mixplot(list(Trg[[i]], RTr[[i]], ETr[[i]]), type = c("p", "l", "l"), pch = "+", col = c(3, 1, 2), l
        c("(X-axis)", "(Y-axis)")[i]))
}
```

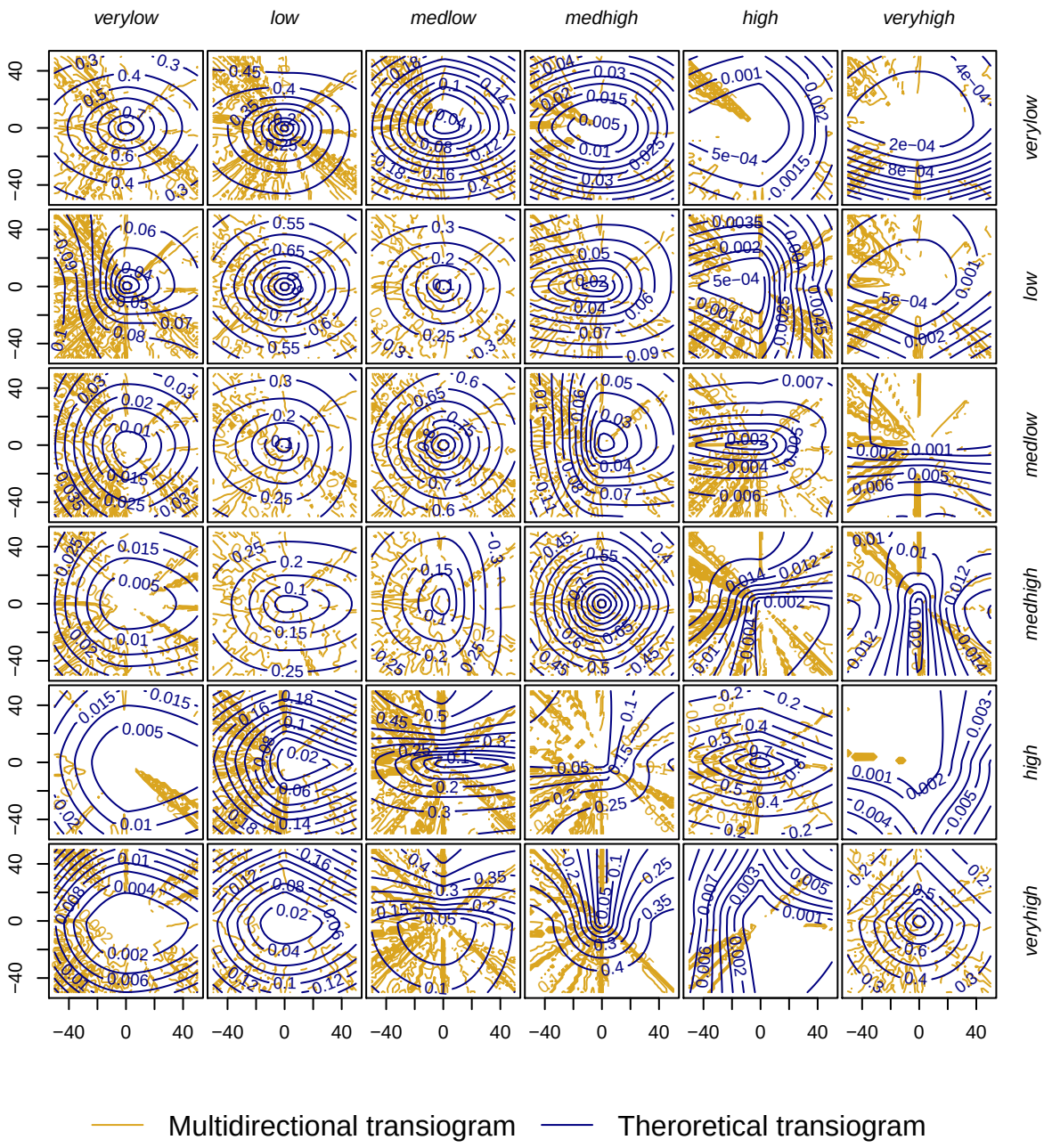



```
# The function pemt() can be considered as another tool to check the anisotropy of the process. It esti
# transition rate matrix for each multidimensional lag direction and computes the transition probabilit
# time the function calculates other probabilities through the transition rates. Then the probabilities
psEmpTr <- pemt(aq_ta$categories, aq_ta@coords, mpoints = 40, max.dist = c(50, 50))

# dev.off() # uncomment and run this line if plot is not running
image(psEmpTr, col = rev(heat.colors(500)), useRaster = TRUE, breaks = c(0:500)/500, contour = FALSE, ma
    0.1))
```

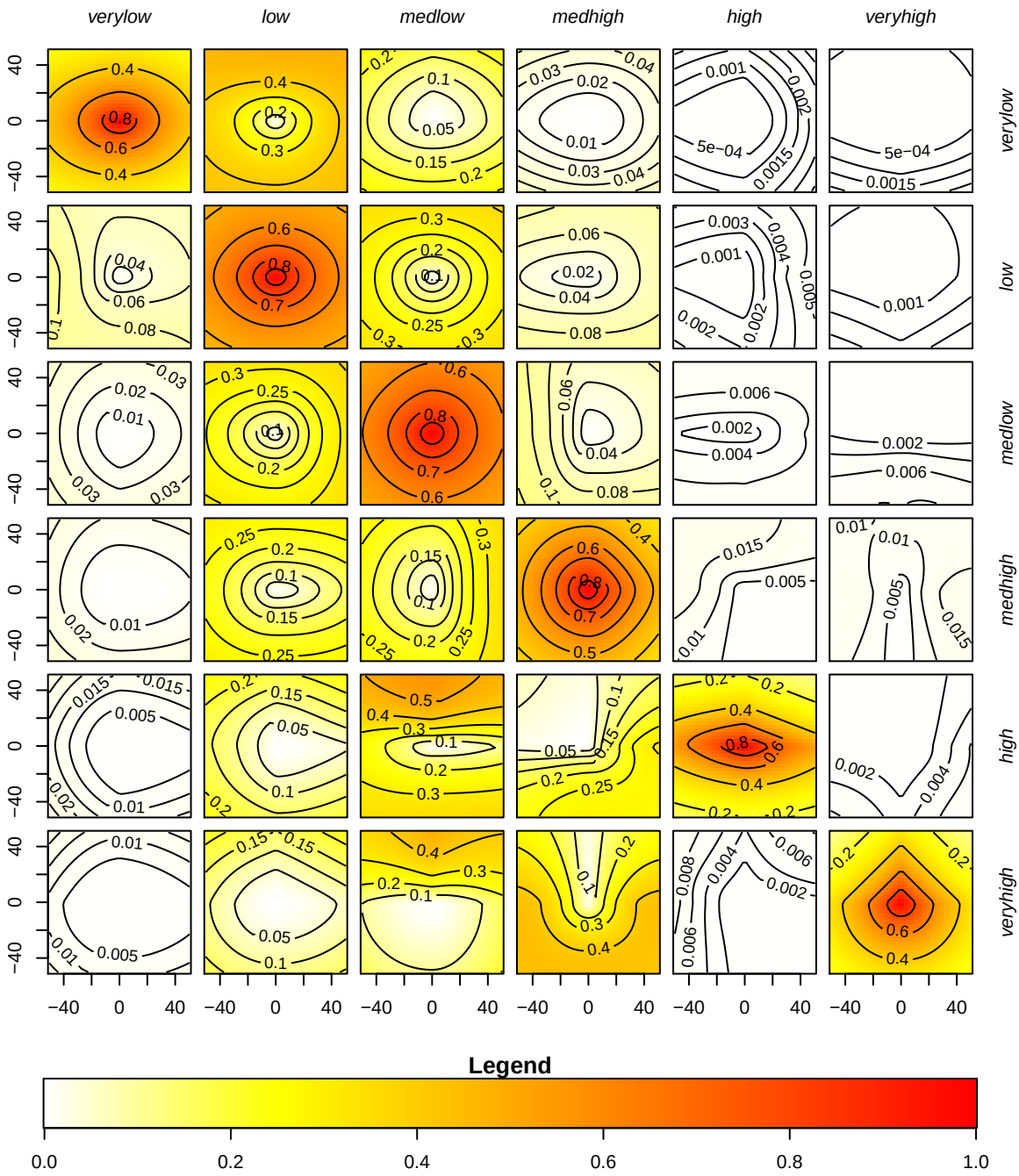

One−dimensional transiograms (Y−axis)
verylow
low
medlow
medhigh
high
veryhigh
Transition probabilities
0.0
0.4
0.8
0
10
20
30
Lags



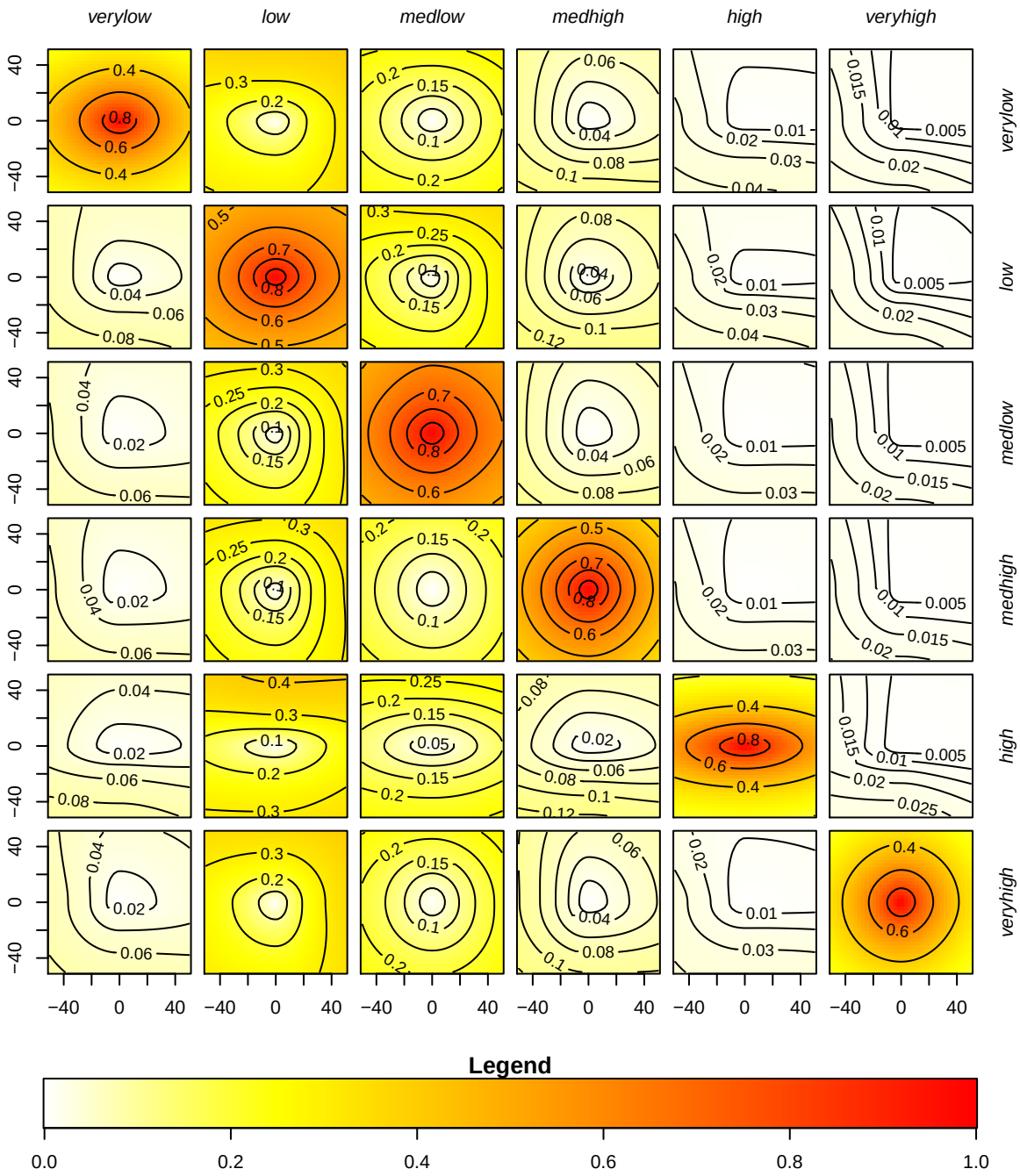

```
# plot again with contour lines that are displayed for both the pseudo-empirical and the theoretical pr
# dev.off()
contour.pemt(psEmpTr, mar = c(0.1, 0.1, 0.1, 0.1), col = c("goldenrod", "navy"))
```




```
# methods to estimate transition rate matrices along axial directions. These functions implement algori
# mean lengths and maximum entropy
MTr <- list()
MTr$average <- multi_tpfit_ml(aq_ta$categories, aq_ta@coords)
MTr$entropy <- multi_tpfit_me(aq_ta$categories, aq_ta@coords)

# with the output of these functions, we can draw the theoretical transition probability maps dev.off()
image(MTr$average, 40, max.dist = 50, col = rev(heat.colors(500)), nlevels = 5, breaks = 0:500/500)
```

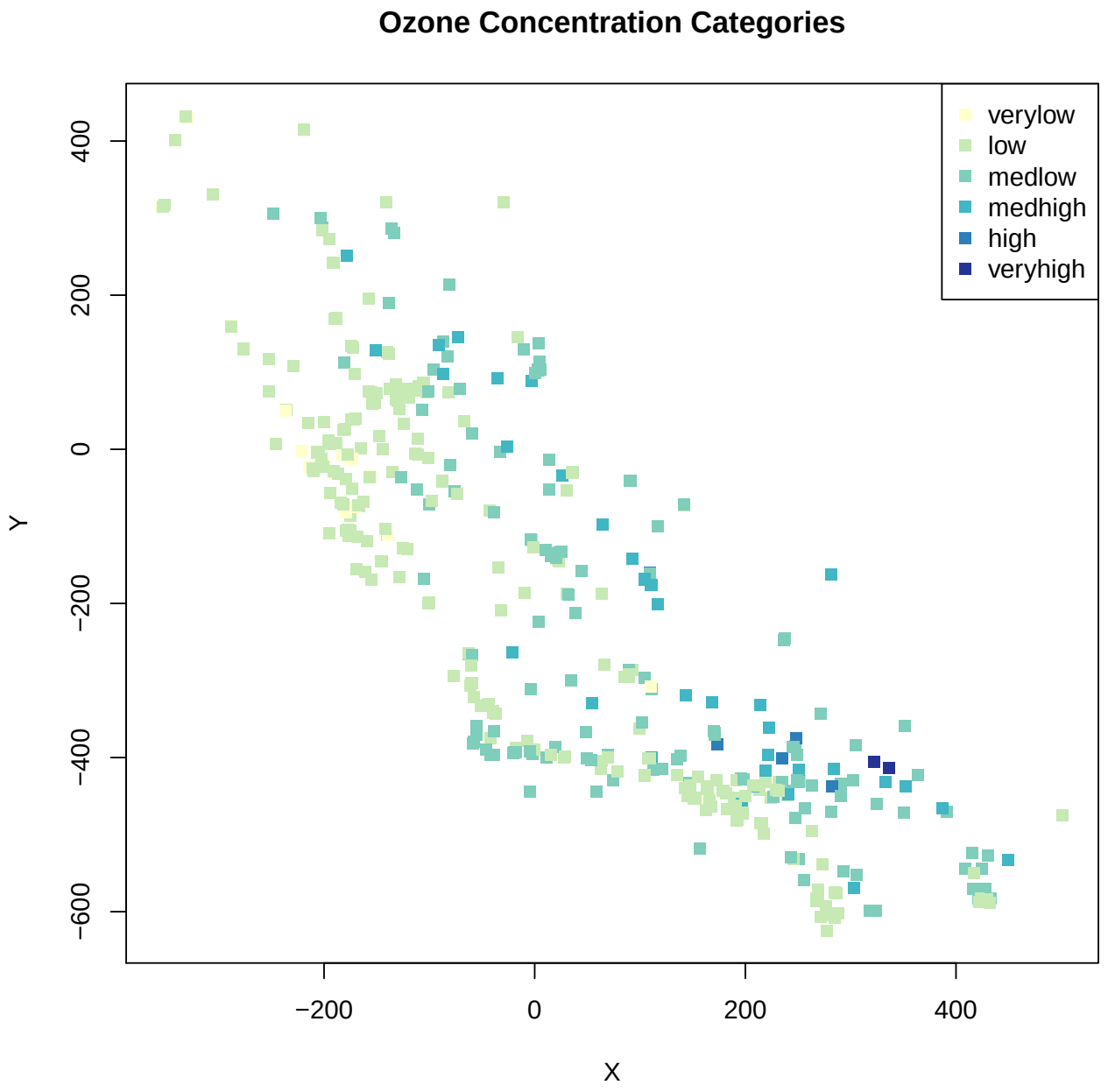



```
# dev.off()
image(MTr$entropy, 40, max.dist = 50, col = rev(heat.colors(500)), nlevels = 5, breaks = 0:500/500)
```

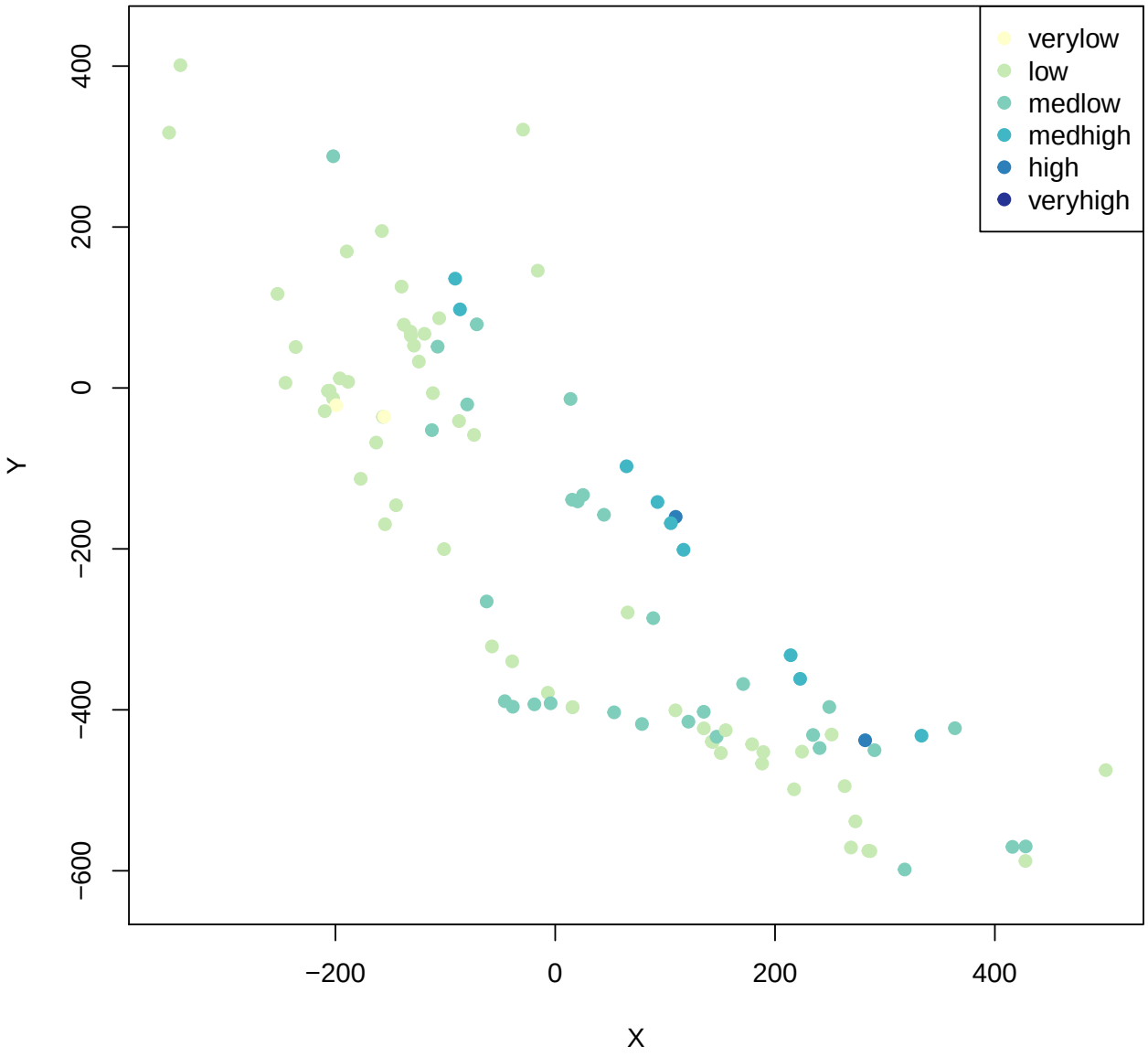



```
# now, the simulation

# plot the map of observations again
X <- aq_ta$x
Y <- aq_ta$y

par(mfrow = c(1, 1), mar = c(5, 4, 4, 2))
plot(X, Y, col = brwCol[aq_ta$categories], pch = 15, main = "Ozone Concentration Categories")
```

```
legend("topright", legend = levels(aq_ta$categories), col = brwCol, pch = 15)
```

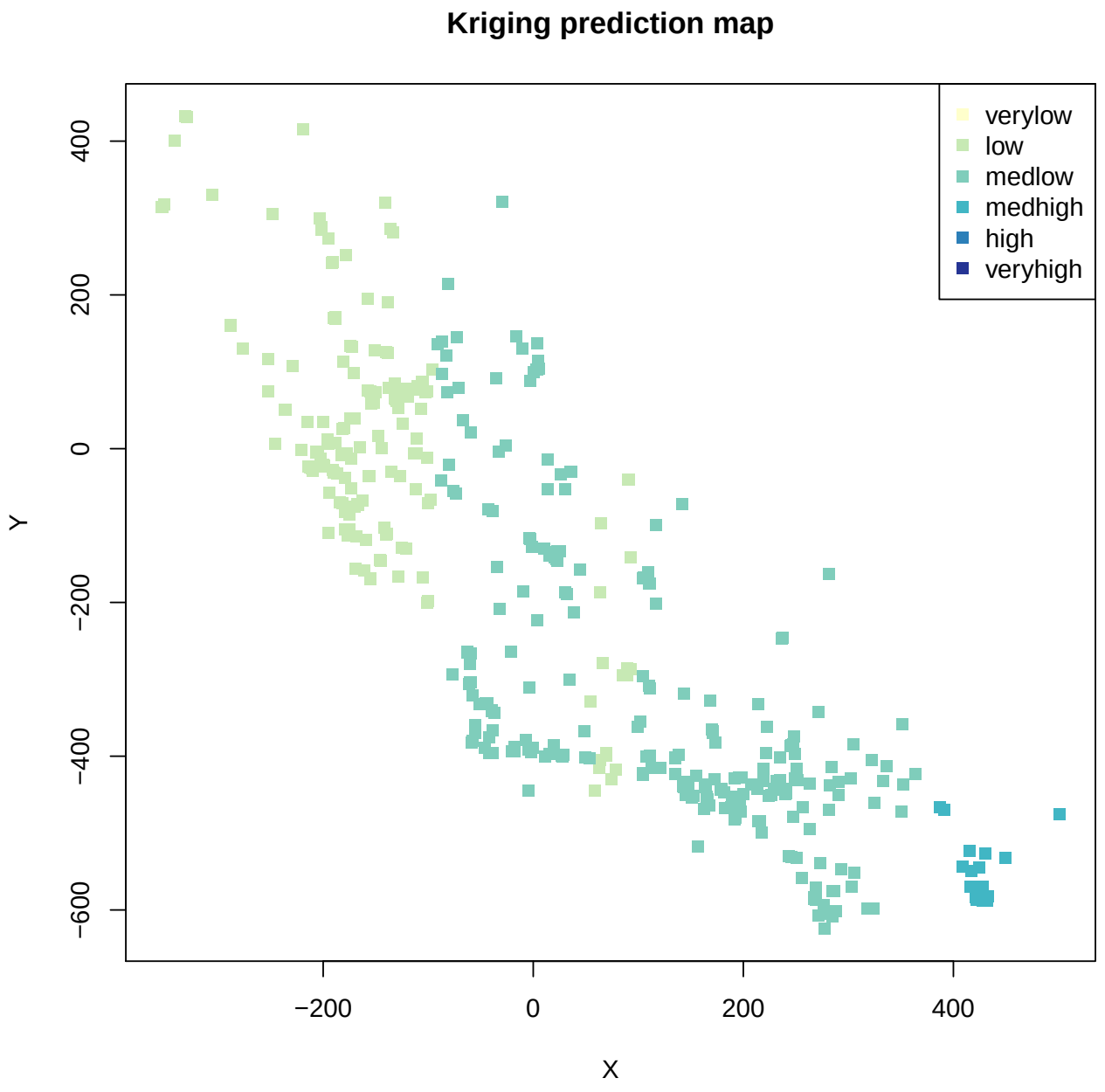



```
# sample observations
set.seed(20161209)
smp <- sample(length(aq_ta):1, 100)

# plot the sample
plot(X, Y, type = "n", main = "Sample of 100 observations", xlab = "X", ylab = "Y")
points(X[smp], Y[smp], pch = 19, col = brwCol[aq_ta$categories[smp]])
legend("topright", legend = levels(aq_ta$categories), col = brwCol, pch = 19)
```

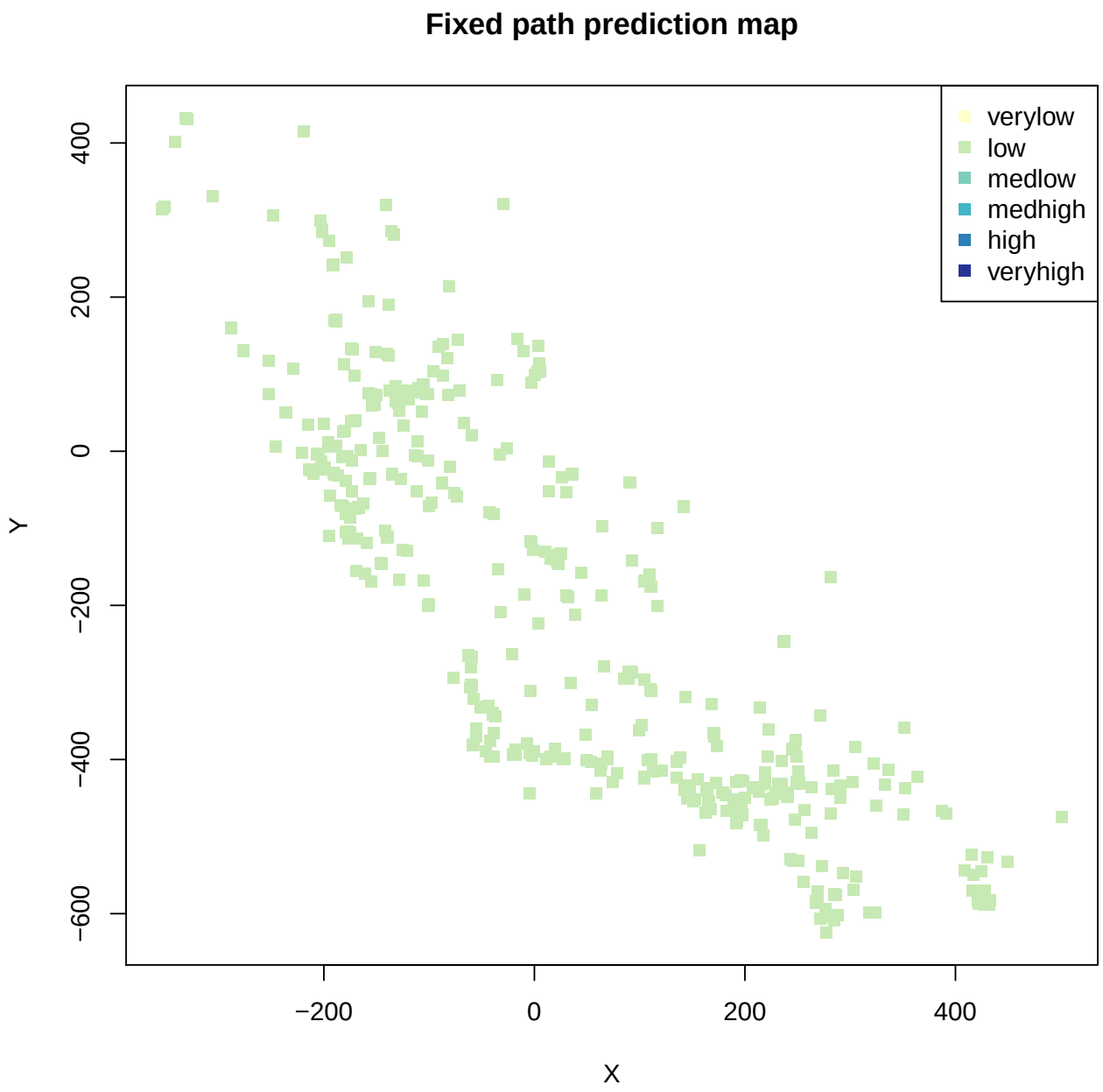



```
# make a grid
grid <- expand.grid(X, Y)

# the kriging algorithm will approximate the conditional probabilities by considering the twelve neares
# all points in the simulation grid.
iks <- sim_ik(MTr$average, data = aq_ta$categories[smp], coords = aq_ta@coords[smp, ], grid, knn = 12)

# Both fixed and random path simulation methods are performed by considering those nearest points along
# directions within a radius of length ten.
fpth <- sim_path(MTr$average, data = aq_ta$categories[smp], coords = aq_ta@coords[smp, ], grid, radius =
rpth <- sim_path(MTr$average, data = aq_ta$categories[smp], coords = aq_ta@coords[smp, ], grid, radius =

# The multinomial categorical simulation method will approximate the prediction probabilities by consid
# points.
```

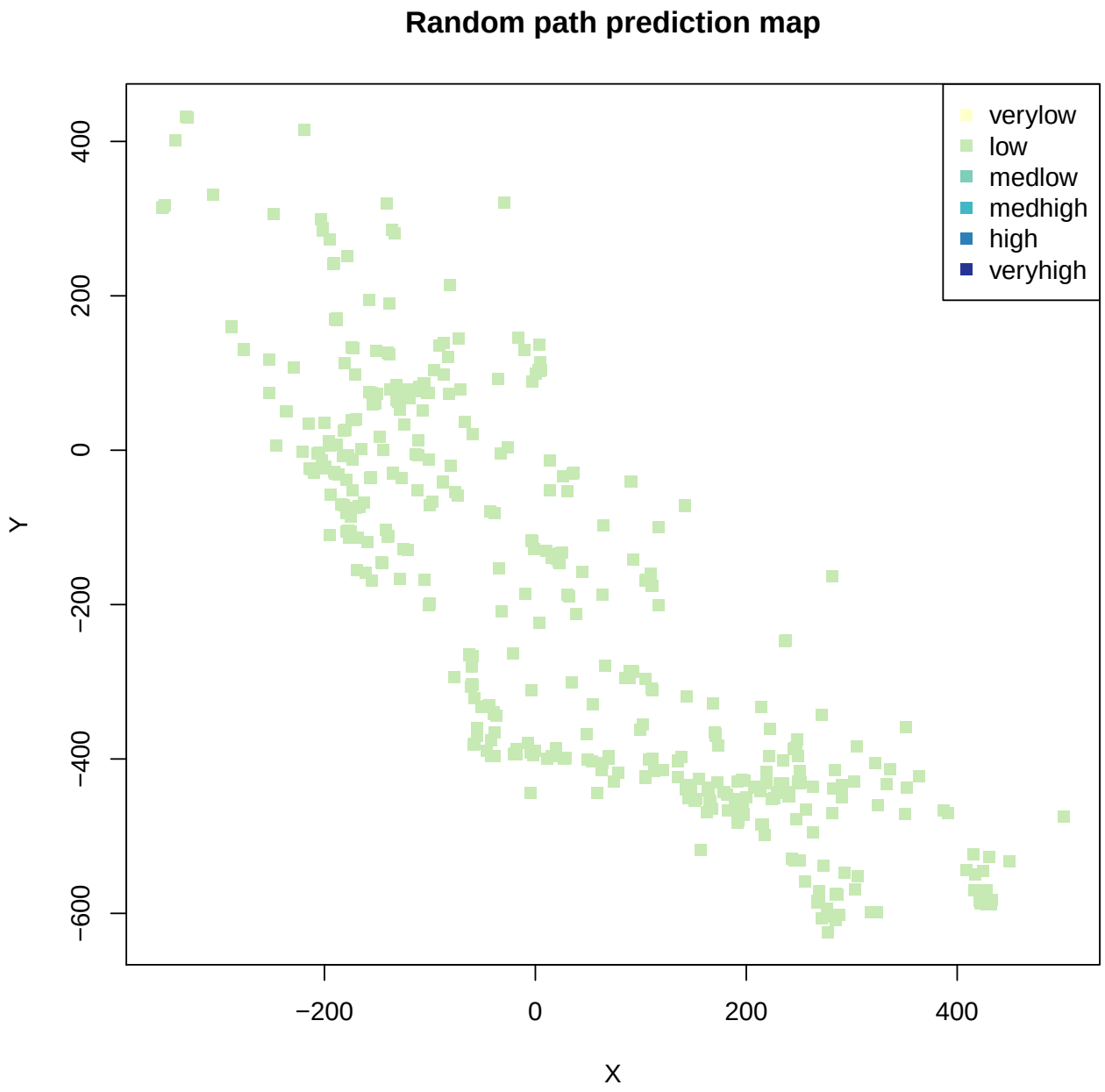
```
mcs <- sim_mcs(MTr$average, data = aq_ta$categories[smp], coords = aq_ta@coords[smp, ], grid)

posCol <- as.integer(iks$Prediction)
plot(X, Y, pch = 15, col = brwCol[posCol], main = "Kriging prediction map")
legend("topright", legend = levels(aq_ta$categories), col = brwCol, pch = 15)
```



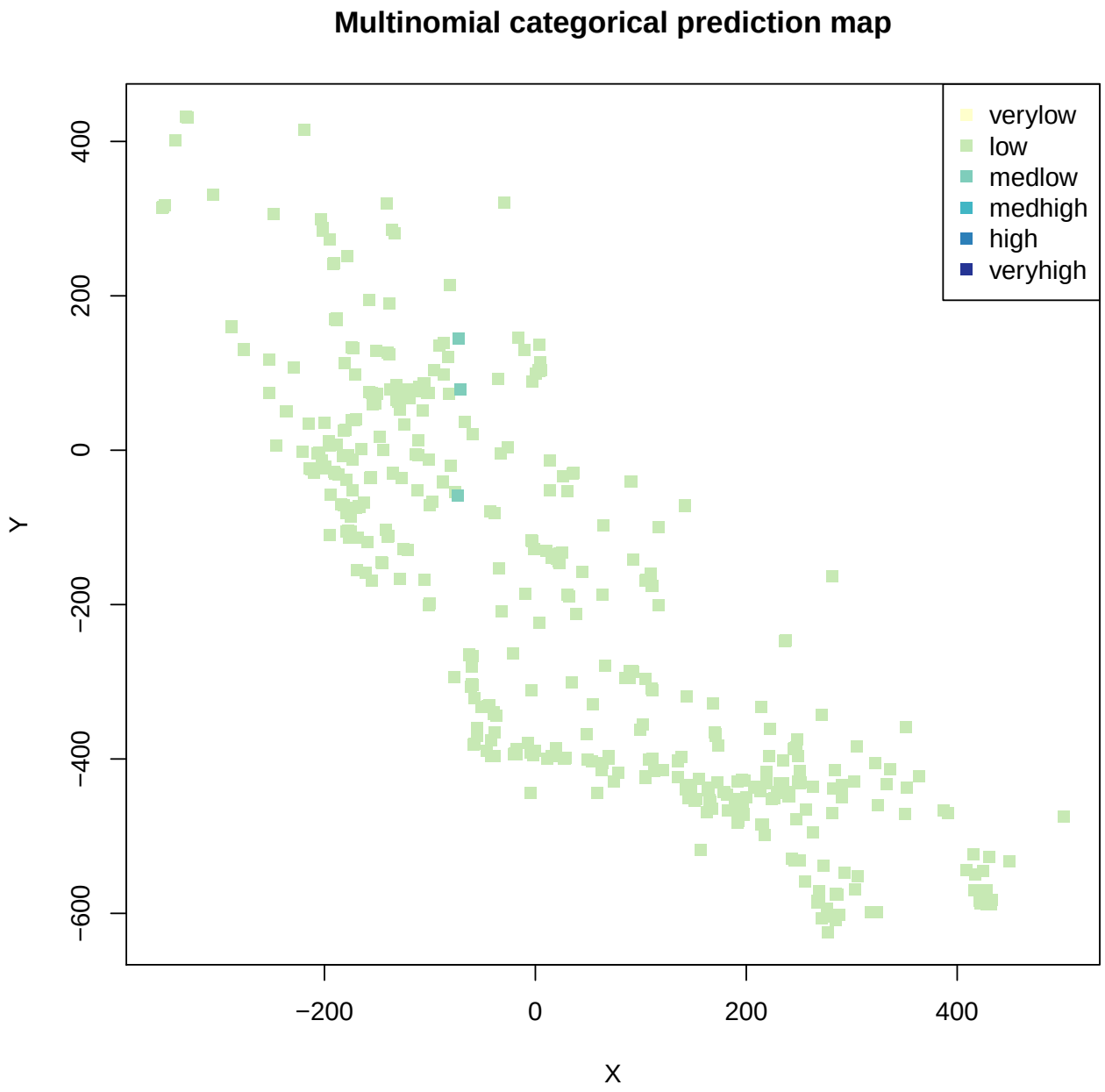

```
posCol <- as.integer(fpth$Prediction)
plot(X, Y, pch = 15, col = brwCol[posCol], main = "Fixed path prediction map")
legend("topright", legend = levels(aq_ta$categories), col = brwCol, pch = 15)
```




```
posCol <- as.integer(rpth$Prediction)
plot(X, Y, pch = 15, col = brwCol[posCol], main = "Random path prediction map")
legend("topright", legend = levels(aq_ta$categories), col = brwCol, pch = 15)
```




```
posCol <- as.integer(mcs$Prediction)
plot(X, Y, pch = 15, col = brwCol[posCol], main = "Multinomial categorical prediction map")
legend("topright", legend = levels(aq_ta$categories), col = brwCol, pch = 15)
```




```
# to compute the number of matches, we calculate the contingency table
ikTb <- table(aq_ta$categories[smp], iks$Prediction[smp])
fpTb <- table(aq_ta$categories[smp], fpth$Prediction[smp])
rpTb <- table(aq_ta$categories[smp], rpth$Prediction[smp])
mcTb <- table(aq_ta$categories[smp], mcs$Prediction[smp])

# relative frequencies of matches
ikPr <- sum(diag(ikTb))/length(aq_ta$categories)
fpPr <- sum(diag(fpTb))/length(aq_ta$categories)
rpPr <- sum(diag(rpTb))/length(aq_ta$categories)
mcPr <- sum(diag(mcTb))/length(aq_ta$categories)

# print in a table
probcov <- c(ikPr, fpPr, rpPr, mcPr)
```

```r
names(probcov) <- "Probability of coverage"
methods <- c("Kriging", "Fixed path", "Random path", "Multinomial")
comparison_methods <- data.frame(methods, probcov)
comparison_methods
##       methods   probcov
## 1     Kriging 0.1150442
## 2  Fixed path 0.1261062
## 3 Random path 0.1261062
## 4 Multinomial 0.1261062

# in order to establish which is the best predictor, one should perform these simulations more than onc
# another 100 observations must be randomly selected.
```