# 11

# Land surface temperature (HRtemp)

## 11.1 Introduction

In this exercise we use one year of measurements of daily mean temperature in Croatia; kindly provided by Melita Perčec–Tadić from the Croatian Meteorological and Hydrological Service[1]. Croatia is a relatively small country but it comprises several different climate regions, which is result from its specific position on the Adriatic sea and fairly diverse topography ranging from plains on the east, through a hilly central part to the mountains separating the continental from the maritime part of the country.

Weather systems originating or crossing over Croatian territory are strongly influenced by this topography, thus the influence they have on weather and climate is highly dependent on the region. The temperature measurements are automatically collected at 123 meteorological stations (Fig. 11.1). The spatial distribution of the stations is not ideal (Zaninović et al., 2008): there is a certain undersampling at higher elevations and in areas with lower population density (for practical reasons, areas of higher population density have been given a priority). Hence, one could expect that mapping accuracy will be lower at higher elevations and in highlands. In addition, some well-known smaller scale effects cannot be represented successfully, e.g. the Zagreb urban heat island that is, according to measurement, 0.5–1.5°C warmer from the surrounding countryside.

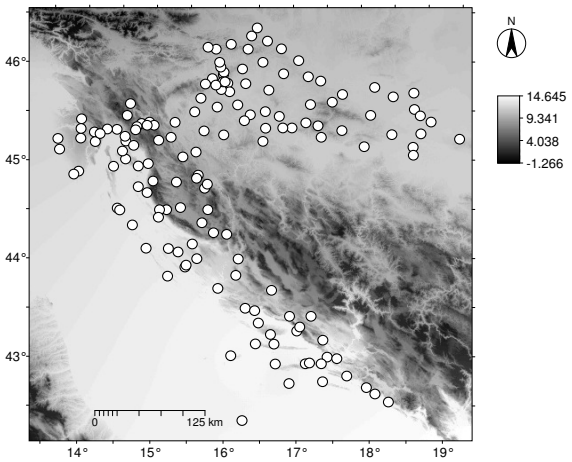


Fig. 11.1: Location of climatic stations in Croatia and long-term monthly temperature for April from the Climatic Atlas of Croatia (Zaninović et al., 2008).

We will model temperature as a function of elevation, distance from the sea, latitude, longitude, time of the year and MODIS daily LST images:

$$\begin{aligned} LST(\mathbf{s}_0, t_0) = {} & b_0 + b_1 \cdot DEM(\mathbf{s}_0) + b_2 \cdot LAT(\mathbf{s}_0) + b_3 \cdot LON(\mathbf{s}_0) + b_4 \cdot DISTC(\mathbf{s}_0) \\ & + b_5 \cdot \cos\left( [t_0 - \phi] \cdot \frac{\pi}{180} \right) + b_6 \cdot LST_{\texttt{MODIS}}(\mathbf{s}_0, t_0); \qquad \Delta t = 1 \text{ day} \end{aligned} \tag{11.1.1}$$

where *DEM* is the elevation map, *LAT* is the map showing distance from the equator, *LON* is the longitude, *DISTC* is the distance from the coast line, $\cos(t)$ is a generic function to account for seasonal variation of values, $\phi$ is the phase angle[2], and $LST_{\texttt{MODIS}}$ is the land surface temperature estimated by the MODIS satellite.

[1] `http://meteo.hr`

[2] A time delay from the coldest day.

*DEM*, *LAT*, *DISTC* are temporally-constant predictors; $LST_{MODIS}$ are temporally variable predictors i.e. time-series of remotely sensed images. More details about how were the MODIS LST images were obtained can be found in section 4.2.

The residuals from such a spatio-temporal regression model can also be analyzed for (spatio-temporal) auto-correlation and used to run 3D interpolation (see §2.5). Once we have fitted the space-time variogram, we can then run **spatio-temporal regression-kriging**[3] to estimate the values at 3D locations. In practice, we only wish to produce maps for a given time interval ($t_0$=constant), i.e. to produce 2D-slices of values in time (see Fig. 2.10; §2.5). For a more gentle introduction to spatio-temporal interpolation see some classical papers by e.g. Huerta et al. (2004), Jost et al. (2005) and Pebesma et al. (2007). A geostatistical exercises with stochastic simulation of rainfall data using remote sensing images can be followed in Teo and Grimes (2007). Schuurmans et al. (2007) propose an automated framework, similar to the one described in this chapter, for prediction of the rainfall fields using spatio-temporal data and Kriging with External Drift.

## 11.2 Data download and preprocessing

The time-series of remotely sensed images, data from meteorological stations and auxiliary maps have been previously prepared by author (§4.2). First, open a new R session and change the working directory to where all the data sets are located (e.g. `C:/croatia/`). Open the R script (`HRtemp.R`) and load the necessary libraries:

```
> library(maptools)
> library(gstat)
> library(rgdal)
> library(lattice)
```

The ground measurements of temperatures can be obtained from the book's homepage. There are two zip files: (1) `HRtemp2006.zip` — contains a digital elevation model, distance from the coast line and temperature measurements from the meteorological stations, (2) `LST2006HR.zip` — contains 92 geotiff's of reprojected MODIS LST images. We need to download and unzip them locally:

```
# Download MODIS LST images:
> download.file("http://spatial-analyst.net/book/system/files/LST2006HR.zip",
+      destfile=paste(getwd(), "LST2006HR.zip", sep="/"))

  trying URL 'http://spatial-analyst.net/book/system/files/LST2006HR.zip'
  Content type 'application/zip' length 20655622 bytes (19.7 Mb)
  opened URL
  downloaded 19.7 Mb


> unzip(zipfile="LST2006HR.zip", exdir=getwd())
> unlink("LST2006HR.zip")
# Download auxiliary maps and LST measurements:
> download.file("http://spatial-analyst.net/book/system/files/HRtemp2006.zip",
+      destfile=paste(getwd(), "HRtemp2006.zip", sep="/"))

  trying URL 'http://spatial-analyst.net/book/system/files/HRtemp2006.zip'
  Content type 'application/zip' length 682970 bytes (666 Kb)
  opened URL
  downloaded 666 Kb


> unzip(zipfile="HRtemp2006.zip", exdir=getwd())
```

and you will find the following data sets:

- `HRdem.asc` — Digital Elevation Model projected in the UTM (zone 33) system;
- `HRdsea.asc` — buffer to coast line in km;

[3] This is called a "*Space-time metric model*", because time dimension is modeled as space dimension (Huerta et al., 2004).

- IDSTA.shp — location of meteorological stations in geographical coordinates;
- HRtemp2006.txt — mean daily temperatures measured at 123 locations for 365 days (the whole year 2006);
- LST2006_**_**.LST_Day_1km.tif — 8-day estimates of daily LST;
- LST2006_**_**.LST_Night_1km.tif — 8-day estimates of night time LST;

We start by importing the temperatures from the HRtemp2006.txt file:

```
> HRtemp2006 <- read.delim("HRtemp2006.txt")
> str(HRtemp2006) # Mean daily temperatures;

  'data.frame':   44895 obs. of  3 variables:
   $ IDT_AK: Factor w/ 123 levels "GL001","GL002",..: 1 1 1 1 1 1 1 1 1 1 ...
   $ DATE  : Factor w/ 365 levels "2006-1-1","2006-1-10",..: 1 12 23 26 27 28 29...
   $ MDTEMP: num  1.6 0.7 1.5 0.3 -0.1 1 0.3 -1.9 -5.4 -3.6 ...
```

This shows that there are 44,895 measurements of mean daily temperature in total. These are only daily mean values, meteorologists typically work with even finer support size (e.g. hourly values). We need to format the imported dates, from string format to the date-time class and then to a numerical format, so that we can use them in further quantitative analysis:

```
> HRtemp2006$cday <- floor(unclass(as.POSIXct(HRtemp2006$DATE))/86400)
```

where POSIXct is the date-time class. Now the days are expressed as cumulative days since 1970-01-01, i.e. as numeric values. For example, a date 2006-01-30, corresponds to:

```
> floor(unclass(as.POSIXct("2006-01-30"))/86400)[[1]]

  [1] 13177


# inverse transformation:
# as.POSIXct(13177*86400, origin="1970-01-01")
```

Next, we can import the latitude/longitude coordinates of the 152 meteorological stations and convert them to the target coordinate system[4]:

```
> IDSTA <- readShapePoints("IDSTA.shp", proj4string=CRS("+proj=longlat +datum=WGS84"))
> IDSTA.utm <- spTransform(IDSTA, CRS("+proj=utm +zone=33 +ellps=WGS84
+         +datum=WGS84 +units=m +no_defs"))
> locs <- as.data.frame(IDSTA.utm)
> names(locs) <- c("IDT_AK", "X", "Y")
> str(locs)

  'data.frame':   152 obs. of  3 variables:
   $ IDT_AK: Factor w/ 152 levels "GL001","GL002",..: 1 2 3 4 5 6 7 8 9 10 ...
   $ X     : num  670760 643073 673778 752344 767729 ...
   $ Y     : num  5083464 5086417 5052001 4726567 4717878 ...


# stations without measurements:
dif.IDSTA <- merge(locs["IDT_AK"], data.frame(IDT_AK=levels(HRtemp2006$IDT_AK),
+     sel=rep(1, length(levels(HRtemp2006$IDT_AK)))), by.x="IDT_AK", all.x=TRUE)
```

and then the raster maps:

[4]Note that we import coordinates of the stations separately because the stations are fixed, so that we only need to know the ID of a station. Otherwise would be inefficient to attach coordinates to each space-time measurements.

```
# Import grids:
> grids <- readGDAL("HRdem.asc")
> names(grids@data)[1] <- "HRdem"
> grids$HRdsea <- readGDAL("HRdsea.asc")$band1
> proj4string(grids) <- IDSTA.utm@proj4string
# create dummy grids (Lat/Lon):
> grids.ll <- spTransform(grids[1], CRS("+proj=longlat +datum=WGS84"))
> grids$Lat <- grids.ll@coords[,2]
> grids$Lon <- grids.ll@coords[,1]
> str(grids@data)

  'data.frame':   238630 obs. of  4 variables:
   $ HRdem : int  1599 1426 1440 1764 1917 1912 1707 1550 1518 1516 ...
   $ HRdsea: num  93 89.6 89.8 93.6 95 ...
   $ Lat   : num  46.5 46.5 46.5 46.5 46.5 ...
   $ Lon   : num  13.2 13.2 13.2 13.2 13.2 ...
```

We will import both nighttime and daytime values, and then derive the average daily values of LST as an average between the two[5]. From the data set description for the MOD11A2 MODIS product we can notice that the original values are in degree Kelvin, which we need to transform to degree Celsius; all values <7500 are NA values; the scaling ratio is 0.02. So in summary, we run:

```
> LST.listday <- dir(pattern=glob2rx("LST2006_**_**.LST_Day_1km.tif"))
> LST.listnight <- dir(pattern=glob2rx("LST2006_**_**.LST_Night_1km.tif"))
> for(i in 1:length(LST.listday)){
>   LSTname <- strsplit(LST.listday[i], ".LST_")[[1]][1]
>   tmp1 <- readGDAL(LST.listday[i])$band1
>   tmp2 <- readGDAL(LST.listnight[i])$band1
# convert to Celsius:
>   tmp1 <- ifelse(tmp1<=7500, NA, tmp1*0.02-273.15)
>   tmp2 <- ifelse(tmp2<=7500, NA, tmp2*0.02-273.15)
>   grids@data[,LSTname] <- (tmp1+tmp2)/2
# simple average -- this ignores that day/night ratio is variable!
> }
```

If you visualize some LST images for various dates (use SAGA GIS), you will notice that there are many NA pixels (especially for winter months). In average, there will always be 10–30% missing pixels in the MODIS images, which is a serious limitation. Also notice that the images can be fairly noisy and with many strange patterns — line or polygon features, jumps in values — which are obviously artifacts. You need to know that these images have been created by patching together images from a period of ±4 days, this way the amount of clouds can be reduced to minimum. Depending on the local meteorological conditions, amount of clouds in an 8–day LST image can still be high (up to 100%). On the other hand, the advantage of using MODIS LST images is that they account for small differences in temperature that are due to different land cover, moisture content, and human-connected activities. Such features cannot be modeled with the constant physical parameters such as elevation, latitude, longitude and distance from the coast line.

## 11.3 Regression modeling

We first need to prepare the regression matrix by overlaying the meteorological stations and imported grids:

```
> IDSTA.ov <- overlay(grids, IDSTA.utm)
> locs <- cbind(IDSTA.ov@data[c("HRdem", "HRdsea", "Lat", "Lon")], locs)
> str(locs)

  'data.frame':   152 obs. of  7 variables:
   $ HRdem : int  161 134 202 31 205 563 80 96 116 228 ...
   $ HRdsea: num  198.5 181.7 192.9 0 1.5 ...
   $ Lat   : num  45.9 45.9 45.6 42.7 42.6 ...
```

[5]Here we will ignore that the length of daytime and nighttime differ for different days.

```
  $ Lon   : num  17.2 16.8 17.2 18.1 18.3 ...
  $ IDT_AK: Factor w/ 152 levels "GL001","GL002",..: 1 2 3 4 5 6 7 8 9 10 ...
  $ X     : num  670760 643073 673778 752344 767729 ...
  $ Y     : num  5083464 5086417 5052001 4726567 4717878 ...
```

which is the initial regression matrix with temporally constant predictors. Temperatures measured at the meteorological stations are missing. We also need to copy the coordinates of stations to the original table. The two tables can be merged by using:

```
> HRtemp2006locs <- merge(HRtemp2006, locs, by.x="IDT_AK")
```

which will basically copy values of constant predictors for all dates. Next we also need to copy the values of MODIS estimated LST at meteorological stations. This is not as trivial because MODIS LST images are not available for all dates. They also need to be sorted in a way that is suitable for analysis. First, let us see which days of the year are available as images:

```
> LSTdate <- rep(NA, length(LST.listday))
> for(i in 1:length(LST.listday)){
>   LSTdate[i] <- gsub("_", "-", strsplit(strsplit(LST.listday[i],
+              ".LST_")[[1]][1], "LST")[[1]][2])
> }
# cumulative days since 2006-01-01:
> LSTcdate <- round((unclass(as.POSIXct(LSTdate)) -
+           unclass(as.POSIXct("2006-01-01")))/86400, 0)
# add one extra day:
> LSTcdate <- c(LSTcdate, 365)
> LSTcdate[1:5]

[1]  0  8 16 24 32
```

next, we need to sort the values in a data frame of the same size as the `HRtemp2006locs`:

```
# create an empty data frame:
> MODIStemp <- expand.grid(IDT_AK=levels(HRtemp2006$IDT_AK),
+      DATE=levels(HRtemp2006$DATE), stringsAsFactors=TRUE)
> MODIStemp$MODIS.LST <- rep(NA, length(MODIStemp[1]))
# copy MODIS LST values per date:
> MODIStemp$MODIS.LST[1:(123*4)] <- rep(IDSTA.ov@data[!is.na(dif.IDSTA$sel),
+      strsplit(LST.listday[i], ".LST_")[[1]][1]], 4)
# all other days:
> for(i in 2:length(LST.listday)){
>   LSTname <- strsplit(LST.listday[i], ".LST_")[[1]][1]
# position/date:
>   d.days <- round((LSTcdate[i+1]-LSTcdate[i-1])/2, 0)
>   d.begin <- round((LSTcdate[i]-d.days/2)*123+1, 0)
>   d.end <- round((LSTcdate[i]+d.days/2)*123+1, 0)
# copy the values:
>   MODIStemp$MODIS.LST[d.begin:d.end] <- rep(IDSTA.ov@data[!is.na(dif.IDSTA$sel),
+      LSTname], d.days)
> }
# the last days:
> MODIStemp$MODIS.LST[(d.end+1):length(MODIStemp$MODIS.LST)] <-
+      rep(IDSTA.ov@data[!is.na(dif.IDSTA$sel),
+      strsplit(LST.listday[i], ".LST_")[[1]][1]], 2)
```

so that we can finally copy all MODIS LST values:

```
> HRtemp2006locs$MODIS.LST <- MODIStemp$MODIS.LST[order(MODIStemp$IDT_AK)]
> str(HRtemp2006locs)
```

```
'data.frame':   44895 obs. of  11 variables:
 $ IDT_AK   : Factor w/ 123 levels "GL001","GL002",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ DATE     : Factor w/ 365 levels "2006-1-1","2006-1-10",..: 1 12 23 26 27 28...
 $ MDTEMP   : num  1.6 0.7 1.5 0.3 -0.1 1 0.3 -1.9 -5.4 -3.6 ...
 $ cday     : num  13148 13149 13150 13151 13152 ...
 $ HRdem    : int  161 161 161 161 161 161 161 161 161 161 ...
 $ HRdsea   : num  198 198 198 198 198 ...
 $ Lat      : num  45.9 45.9 45.9 45.9 45.9 ...
 $ Lon      : num  17.2 17.2 17.2 17.2 17.2 ...
 $ X        : num  670760 670760 670760 670760 670760 ...
 $ Y        : num  5083464 5083464 5083464 5083464 5083464 ...
 $ MODIS.LST: num  -1.17 -1.17 -1.17 -1.17 -5.92 ...
```

note that the values of MODIS LST are now available as a single column in the original regression matrix. Before we can create a 3D point data set, it is a useful thing to scale $t$-coordinate, so it has approximately similar range[6] as the $XY$ coordinates:

```
# scale the values:
> tscale <- (((grids@bbox[1,"max"]-grids@bbox[1,"min"])+(grids@bbox[2,"max"]
+    -grids@bbox[2,"min"]))/2)/(max(HRtemp2006locs$cday)-min(HRtemp2006locs$cday))
> HRtemp2006locs$cdays <- tscale * HRtemp2006locs$cday
# 3D points:
> coordinates(HRtemp2006locs) <- c("X", "Y", "cdays")
> proj4string(HRtemp2006locs) <- CRS(proj4string(grids))
# copy values:
> HRtemp2006locs$cdays <- HRtemp2006locs@coords[,"cdays"]
```

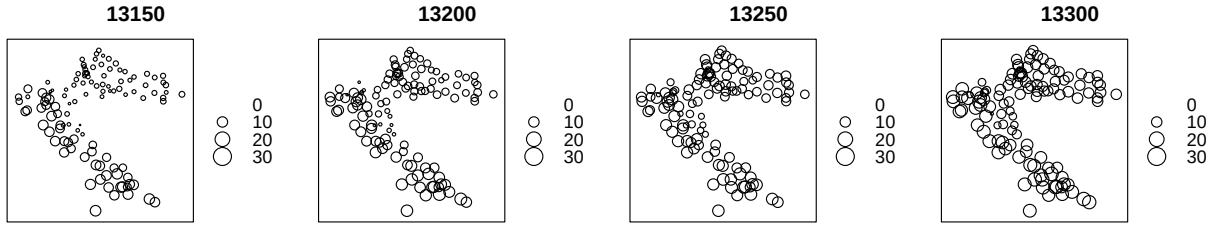


Fig. 11.2: Spatial pattern of measured mean-daily temperatures for 2006-01-02 (13150), 2006-02-21 (13200), 2006-04-12 (13250), 2006-06-01 (13300).

Now that we have attached coordinates to the temperature measurements and created a 3D point object, we can visualize them by using the `bubble` method available in the sp package (Fig. 11.2):

```
> bubble(subset(HRtemp2006locs, HRtemp2006locs$cday==13150&!is.na(HRtemp2006locs$MDTEMP),
+     select="MDTEMP"), fill=F, col="black", maxsize=2,
+     key.entries=c(0,10,20,30), main="13150")
```

We can also make subset of the data and observe how the values change through time at a specific station:

```
# pick three meteorological stations:
> GL001 <- subset(HRtemp2006locs@data, IDT_AK=="GL001", select=c("MDTEMP", "cday"))
> KL003 <- subset(HRtemp2006locs@data, IDT_AK=="KL003", select=c("MDTEMP", "cday"))
> KL094 <- subset(HRtemp2006locs@data, IDT_AK=="KL094", select=c("MDTEMP", "cday"))
> par(mfrow=c(1,3))
> scatter.smooth(GL001$cday, GL001$MDTEMP, xlab="Cumulative days",
+     ylab="Mean daily temperature (\260C)", ylim=c(-12,28), col="grey")
> scatter.smooth(KL003$cday, KL003$MDTEMP, xlab="Cumulative days",
+     ylab="Mean daily temperature (\260C)", ylim=c(-12,28), col="grey")
> scatter.smooth(KL094$cday, KL094$MDTEMP, xlab="Cumulative days",
+     ylab="Mean daily temperature (\260C)", ylim=c(-12,28), col="grey")
```

[6] This is not really a requirement for the analysis, but it makes visualization of 3D space and variograms easier.

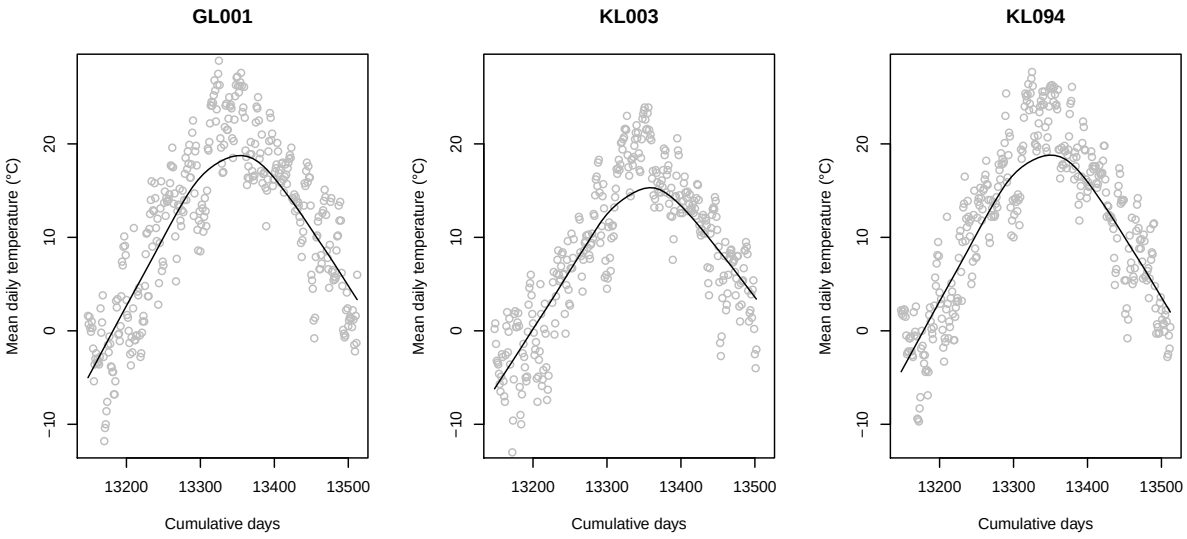


Fig. 11.3: Temporal dynamics of mean-daily temperatures at selected meteorological stations.

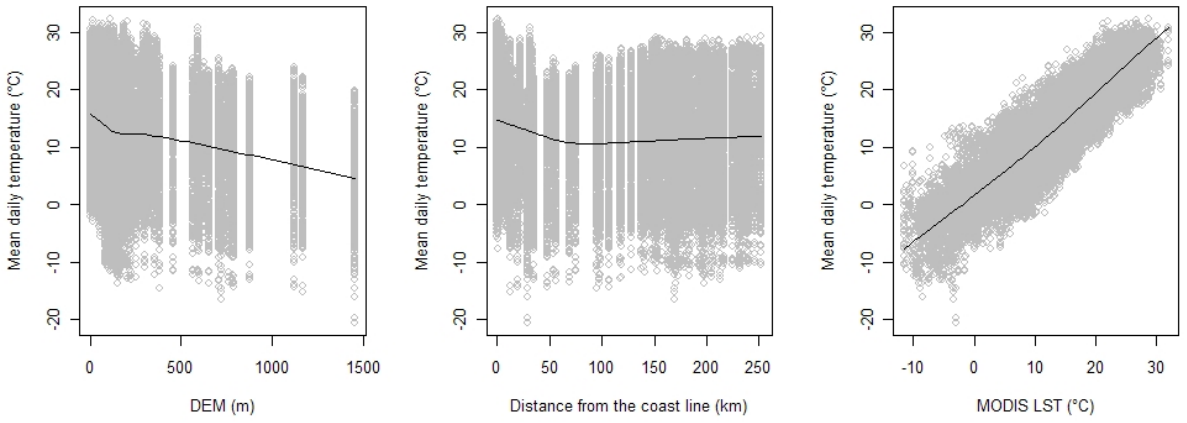


Fig. 11.4: Scatter plots showing the general relationship between daily temperature (`MDTEMP`), elevation, distance from the coast line and MODIS LST images.

which shows that the values change more systematically in time domain, than in the space domain. It will also be interesting to observe individual relationships between `MDTEMP`, `HRdem`, `HRdsea` and `MODIS.LST` (Fig. 11.4). This shows, as expected, that the temperature drops with elevation, and with distance from the coast line[7]. Note also that MODIS LST seem to be a fairly accurate (unbiased) predictor of the actual measured temperature: the relationship is highly linear and the scatter around the regression line is constant. Nevertheless, Fig. 11.4 also shows that there is a significant scatter around the regression lines, which means that also the residuals will be significant.

We can now fit a linear model using the Eq.(11.1.1):

```
> theta <- min(HRtemp2006locs$cday)
> lm.HRtemp <- lm(MDTEMP ~ HRdem+HRdsea+Lat+Lon+cos((cday-theta)*pi/180)+MODIS.LST,
+       data=HRtemp2006locs)
```

[7]Based on this data, it seems that the influence of the sea climate is up to the distance of about 80 km.

```
> summary(lm.HRtemp)$adj.r.squared

  [1] 0.8423278


# plot(lm.HRtemp)
```

which shows that all predictors are highly significant; the model explains 84% of variability in the MDTEMP values; the derived residuals are normally distributed around the 0 value, with only 3 influential values (outliers).

## 11.4 Space-time variogram estimation

Now we can also try to fit the 3D variogram for residuals. Note that gstat supports 3D interpolation, but it does not actually support interactive fitting of 3D variograms. In fact, to my knowledge, interactive modeling of space-time autocorrelation is still very limited in R, but also in commercial packages such as Isatis or ArcGIS.

We can start with plotting the points in a 3D space, which is possible by using the cloud method available in the lattice package. This will produce a plot shown in Fig. 11.5, which gives us a good idea about the sampling design specific to this data set. The data set consist of large number of space-time points. Furthermore, for the purpose of this exercise, we can first randomly subset the original data set to 10% of its size to speed up plotting and fitting of the variograms:

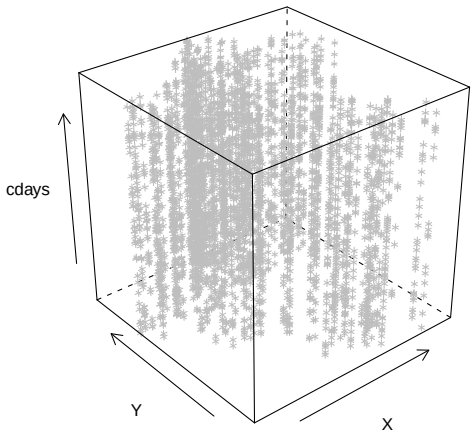


Fig. 11.5: Cloud plot showing location of meteorological stations in the space-time cube.

```
# remove missing values:
> HRtemp2006.f <- HRtemp2006locs[-lm.HRtemp$na.action,]
# copy the residuals:
> HRtemp2006.f$rMDTEMP2006 <- lm.HRtemp$residuals
# sub-sample:
> HRtemp2006.sel <- HRtemp2006.f[
+    runif(length(HRtemp2006.sel$rMDTEMP2006))<0.1,]
# str(HRtemp2006.sel)
# plot the 3D points:
> coords <- as.data.frame(HRtemp2006.sel@coords)
> cloud(cdays ~ X*Y, coords, col="grey")
```

In addition, assuming that the variogram will be anisotropic, a good idea is to plot a variogram map (Fig. 11.6, left):

```
> varmap.plt <- plot(variogram(rMDTEMP2006 ~ 1, HRtemp2006.sel, map=TRUE,
+    cutoff=sqrt(areaSpatialGrid(grids))/2, width=30*grids@grid@cellsize[1]),
+    col.regions=grey(rev(seq(0,1,0.025))))
> rv.MDTEMP2006 <- variogram(rMDTEMP2006 ~ 1, HRtemp2006.sel, alpha=c(45,135))
> rvgm.MDTEMP2006 <- fit.variogram(rv.MDTEMP2006,
+    vgm(psill=var(HRtemp2006.sel$rMDTEMP2006),
+    "Exp", sqrt(areaSpatialGrid(grids))/4, nugget=0, anis=c(p=45,s=0.5)))
> vgm.plt <- plot(rv.MDTEMP2006, rvgm.MDTEMP2006, plot.nu=FALSE, cex=2, pch="+",
+    col="black")
> print(varmap.plt, split=c(1,1,2,1), more=T)
> print(vgm.plt, split=c(2,1,2,1), more=F)
> rvgm.MDTEMP2006
```

```
  model    psill    range ang1 anis1
1   Nug 6.094418     0.00    0   1.0
2   Exp 4.639634 15392.17   45   0.5
```

which shows that a strong anisotropy exists (the azimuth of the longer axis is somewhere at 135°), although it is not so distinct. Recall that the main mountain chain on the Balkans (see Fig. 11.1) spreads approximately in the same direction.

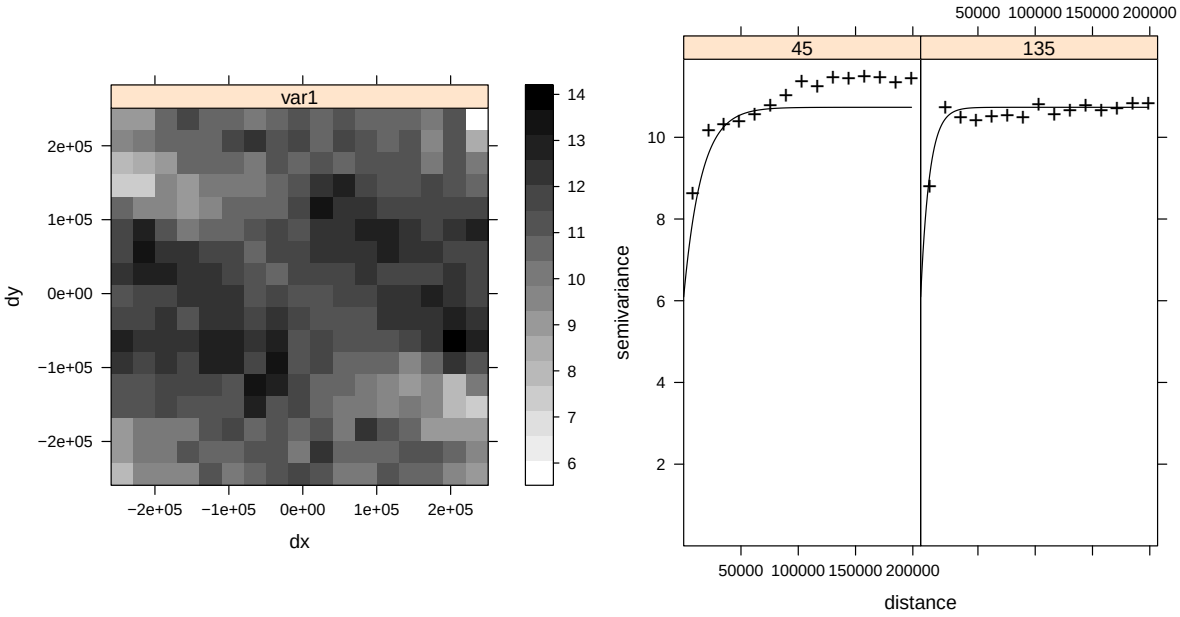


Fig. 11.6: Variogram map (left) and fitted anisotropic variogram model (right) for MDTEMP residuals.

The plotted 3D variograms look the same as in Fig. 9.1, although we know that this is not a 2D but a 3D data set. Visual exploration of space-time (3D or 4D) variogram models in R is at the moment limited to 2D spaces, although the situation might change with the new space-time (stpp[8]) package. Note also from Fig. 11.3 that is obvious that nugget variation in time direction will be much higher greater in the space domain.

## 11.5 Spatio-temporal interpolation

Making prediction in the space-time domain (3D) is not as easy as making 2D predictions (the previous exercises). It will take some time to prepare the prediction locations, get the values of all constant and temporal predictors and then visualize the final results. As mentioned previously, we do not intend to make predictions for the whole space-time cube, but only for fixed time-intervals, i.e. time slices (see also Fig. 2.10). Detailed steps are now explained down-below.

### 11.5.1 A single 3D location

We can start by defining the geostatistical model (for residuals):

```
> g.MDTEMP <- gstat(id=c("rMDTEMP2006"), formula=rMDTEMP2006 ~ 1,
+       data=HRtemp2006.f, nmax=40, model=rvgm.MDTEMP2006)
```

We can now first test the algorithm by using a single 3D point. For example, we ask ourselves what is the mean daily temperature for 1st of August 2006, at location X=575671 E, Y=5083528 N? To prepare the new point location we can use:

[8] http://stpp.r-forge.r-project.org

```
> newloc.t <- "2006-08-01"
> newloc.ct <- floor(unclass(as.POSIXct(newloc.t))/86400)[[1]]
> newloc.x <- 575671
> newloc.y <- 5083528
> newloc.xyt <- as.data.frame(matrix(c(newloc.x, newloc.y, tscale*newloc.ct),
+       ncol=3, dimnames = list(c("p1"), c("X","Y","cdays"))))
> coordinates(newloc.xyt) <- c("X","Y","cdays")
> proj4string(newloc.xyt) <- CRS(proj4string(grids))
# 3D prediction location:
> newloc.xyt

  SpatialPoints:
            X       Y     cdays
  [1,] 575671 5083528 17929560
  Coordinate Reference System (CRS)
  arguments: +proj=utm +zone=33
  +ellps=WGS84 +datum=WGS84 +units=m
  +no_defs +towgs84=0,0,0
```

Next, we need to get the values of the auxiliary predictors at this location. We can do this by overlaying the new point with the imported gridded maps:

```
> newloc.xy <- as.data.frame(newloc.xyt)
> coordinates(newloc.xy) <- c("X","Y")
> proj4string(newloc.xy) <- CRS(proj4string(grids))
> ov.newloc.xy <- overlay(grids, newloc.xy)
# add the "time"-location:
> ov.newloc.xy$cday <- newloc.ct
> str(ov.newloc.xy@data)

  'data.frame':   1 obs. of  52 variables:
   $ HRdem        : int 805
   $ HRdsea       : num 165
   $ Lat          : num 45.9
   $ Lon          : num 16
   $ LST2006_01_01: num NA
   $ LST2006_01_09: num -5.23
  ...
   $ LST2006_12_27: num 0.37
   $ cday         : num 13360
```

Notice that the value of `MODIS.LST` is missing. The overlay operation works only in 2D, hence it does not know what the value of `MODIS.LST` is for the given date. We need to first estimate what would be the closest MODIS image for `2006-08-01`, and then copy those values:

```
> cdate <- round((unclass(as.POSIXct(newloc.t)) -
+          unclass(as.POSIXct("2006-01-01")))/86400, 0)[1]
> cdate

  [1] 212


> LSTname <- strsplit(LST.listday[which.min(abs(cdate-LSTcdate))], ".LST_")[[1]][1]
> LSTname

  [1] "LST2006_07_28"


> ov.newloc.xy$MODIS.LST <- ov.newloc.xy@data[,LSTname]
> ov.newloc.xy$MODIS.LST

  [1] 21.29
```

which shows that the '*closest*' MODIS image is from 2006-07-28 and MODIS estimated temperature for that date is 21.29°C. Now the new location is complete, we can make predict the mean daily temperature at this location, and using the model estimated in the previous section:

```
> locMDTEMP.reg <- predict(lm.HRtemp, ov.newloc.xy)
# the trend part:
> locMDTEMP.reg

    35497
 18.76291


# OK of residuals;
> locMDTEMP <- predict.gstat(g.MDTEMP, newloc.xyt, beta=1, BLUE=FALSE)

  [using ordinary kriging]


> locMDTEMP.reg + locMDTEMP$rMDTEMP2006.pred

    35497
 18.98239
```

which is somewhat lower than we expected for this time of year. Just to check how close the result is to the temperature actually measured at the closest location:

```
# locate the closest measured temperature:
> closest.pnt <- which.min(dist(rbind(newloc.xy@coords,
+      IDSTA.utm@coords))[1:length(IDSTA.utm@coords[,1])])
> closest.IDSTA <- as.character(IDSTA.utm$IDSTA[closest.pnt])
> closest.IDSTA

  [1] "GL023"


> subset(HRtemp2006locs, HRtemp2006locs$cday==newloc.ct&
+   HRtemp2006locs$IDT_AK==closest.IDSTA, select="MDTEMP")

                     coordinates MDTEMP
 8243 (575118, 5084260, 17929600)   17.2
```

which shows that the predicted temperature is somewhat higher than the one measured at the same day, at the closest station. The values are higher mainly because the LST image shows higher values for that period. Surprisingly, the residuals are positive, even though measured values are above predicted and even though the prediction point is fairly close to the measurement location (distance is only 913 m). Take into account that station GL023 is at the top of a mountain, so that it is realistic to always expect a lower temperature even at so short distance.

### 11.5.2 Time-slices

We can now generate predictions for a list of new locations. Because there is still no support for 3D grids in R, we can instead make 3D regular points and then predict at those locations. In fact, we will define only slices of the space-time cube for which we will make predictions. Another important issue is that we will put operations in a loop to speed up the processing. Imagine, there are 365 days, and if we would want to interpolate the values for MDTEMP — this would take a lot of scripting. To avoid this problem, we create a R loop that will interpolate as many maps as we like. For the purpose of this exercise, we derive only time-slices for which we also have MODIS images available:

```
# available MODIS images:
> slices <- LSTdate
# new locations:
> grids.xy <- as(grids[c("HRdem", "HRdsea", "Lat", "Lon")], "SpatialPixelsDataFrame")
> for(i in 1:length(slices)) {
>   newlocs.xyt <- grids.xy@data
>   newlocs.xyt$X <- grids.xy@coords[,"x"]
>   newlocs.xyt$Y <- grids.xy@coords[,"y"]
>   slice <- floor(unclass(as.POSIXct(slices[i]))/86400)[[1]]
>   newlocs.xyt$cday <- rep(slice, length(newlocs.xyt[1]))
>   newlocs.xyt$cdays <- tscale * newlocs.xyt$cday
>   LSTname <- strsplit(LST.listday[i], ".LST_")[[1]][1]
>   newlocs.xyt$MODIS.LST <- grids@data[grids.xy@grid.index,LSTname]
>   coordinates(newlocs.xyt) <- c("X","Y","cdays")
>   proj4string(newlocs.xyt) <- CRS(proj4string(grids))
>   MDTEMP.ok <- predict.gstat(g.MDTEMP, newlocs.xyt, beta=1, BLUE=FALSE)
>   MDTEMP.reg <- predict(lm.HRtemp, newlocs.xyt)
>   grids@data[,paste(LSTname,".RK",sep="")] <- MDTEMP.reg+MDTEMP.ok$rMDTEMP2006.pred
> }
```

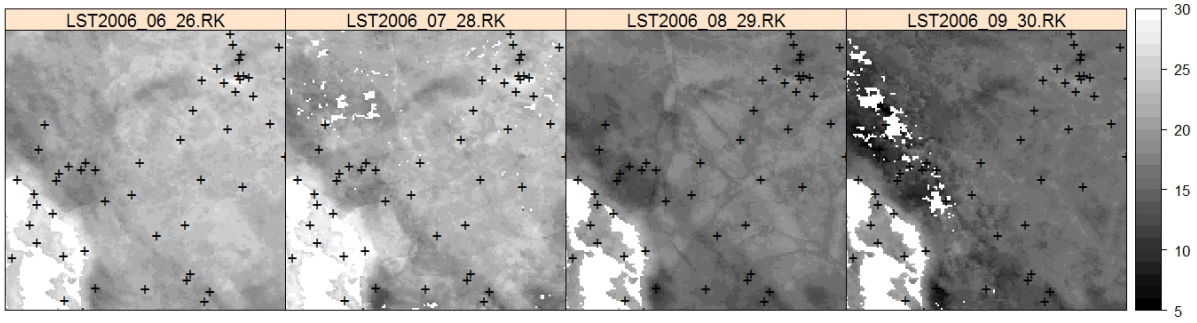


Fig. 11.7: Mean daily temperatures (°C) predicted using spatio-temporal regression-kriging (see titles for dates). Missing pixels are due to clouds in the MODIS LST images.

This operation is relatively time and memory consuming[9] so try also to limit the number of slices to <20. After the process is finished, a useful thing to do is to visualize the predicted maps together with the measured point values by using e.g. (Fig. 11.7):

```
> pr.list <- c("LST2006_06_26.RK", "LST2006_07_28.RK", "LST2006_08_29.RK", "LST2006_09_30.RK")
> spplot(grids[pr.list], col.regions=grey(seq(0,1,0.025)), at=seq(5,30,1),
+      xlim=c(450000,600000), ylim=c(4950000,5100000), sp.layout=list("sp.points",
+      IDSTA.utm, pch="+", cex=1.5, col="black"))
```

You will soon discover that these predictions are mainly controlled by the MODIS LST images (also clearly visible from Fig. 11.4); a large portion of `NA` areas is visible in the output maps. These could be fixed by iteratively filtering[10] the original MODIS images before these are used as predictors.

### 11.5.3 Export to KML: dynamic maps

We have produced a series of maps of daily temperatures. We want now to export these maps to Google Earth and visualize them as a time series. Note that Google Earth support spatio-temporal data that can be browsed by using a *Timeline* bar (Fig. 11.8). In principle, transformation of 2D data to spatio-temporal data is rather

[9] A way to speed up the processing would be to limit the search radius (see p.94), but this would also lead to artifacts in the maps.
[10] See for example Addink and Stein (1999).

simple — we only need to add a field called <TimeSpan> and then define the begin and end time to which a map refers to. If Google Earth sees that this field has been defined, it will automatically browse it with a time slider. We first need to resample all maps to geographic grid (see also section 5.6.2 for more details):

```
# resampling of maps to geographic coordinates:
> for(i in seq(5,35,2)) {
>   LSTname <- strsplit(LST.listday[i], ".LST_")[[1]][1]
>   write.asciigrid(grids[paste(LSTname, ".RK", sep="")],
+      paste(LSTname, "_RK.asc", sep=""), na.value=-999)
>   rsaga.esri.to.sgrd(in.grids=paste(LSTname, "_RK.asc", sep=""),
+      out.sgrd=paste(LSTname, "_RK.sgrd", sep=""), in.path=getwd())
 # bilinear resample:
>   rsaga.geoprocessor(lib="pj_proj4", 2, param=list(SOURCE_PROJ=paste('"',
+      proj4string(grids), '"', sep=""), TARGET_PROJ="\"+proj=longlat
+      +datum=WGS84\"", SOURCE=paste(LSTname, "_RK.sgrd", sep=""),
+      TARGET=paste(LSTname, "_RK_ll.sgrd", sep=""), TARGET_TYPE=0,
+      INTERPOLATION=1))
 # write back to ASCII:
>   rsaga.sgrd.to.esri(in.sgrds=paste(LSTname, "_RK_ll.sgrd", sep=""),
+      out.grids=paste(LSTname, "_RK.asc", sep=""), prec=1, out.path=getwd())
> }
```

and then read the maps back into R:

```
> MDTEMP.list <- dir(pattern=glob2rx("LST2006_**_**_RK.asc"))
> grids.geo <- readGDAL(MDTEMP.list[1])
> proj4string(grids.geo) <- CRS("+proj=longlat +datum=WGS84")
> names(grids.geo) <- strsplit(MDTEMP.list[1], ".asc")[[1]][1]
> for(i in 2:length(MDTEMP.list)){
>    LSTname <- strsplit(MDTEMP.list[i], ".asc")[[1]][1]
>    grids.geo@data[,LSTname] <- readGDAL(MDTEMP.list[i])$band1
> }
```

We can now prepare a GE_SpatialGrid object using the original sp SpatialGridDataFrame:

```
> grids.kml <- GE_SpatialGrid(grids.geo)
```

We want to use the same legend for all time-slices and add it as a screen overlay, which means that we need to determine suitable limits for the legend, e.g. 98% range of values:

```
> MDTEMPxlim <- quantile(HRtemp2006.f$MDTEMP, probs=c(0.01,0.99))
> MDTEMPxlim

    1%  99%
 -6.0 28.4
```

We export all maps as PNGs using a fixed legend:

```
# export all maps as PNG:
> for(i in 1:length(MDTEMP.list)) {
>   LSTname <- strsplit(MDTEMP.list[i], ".asc")[[1]][1]
>   png(file=paste(LSTname, ".png", sep=""), width=grids.kml$width,
+     height=grids.kml$height, bg="transparent")
>   par(mar=c(0,0,0,0), xaxs="i", yaxs="i")
>   image(as.image.SpatialGridDataFrame(grids.geo[LSTname]), col=bpy.colors(),
+     zlim=(MDTEMPxlim), xlim=grids.kml$xlim, ylim=grids.kml$ylim)
>   dev.off()
> }
```

To export the legend PNG, we use:

```
# prepare the legend:
> png(file="legend.png", width=grids.kml$width/4, height=grids.kml$height/3, bg="white")
> par(mar=c(0,0.1,0,0.1), yaxs="i")
> image(grids, names(grids[5]), col="white")
> source("http://spatial-analyst.net/scripts/legend_image.R")
> legend_image(c(grids@bbox[1,1],
+     grids@bbox[1,2]-(grids@bbox[1,2]-grids@bbox[1,1])/4), c(grids@bbox[2,1],
+     grids@bbox[2,2]), seq(MDTEMPxlim[[1]],MDTEMPxlim[[2]],1), vertical=TRUE,
+     col=bpy.colors(round(MDTEMPxlim[[2]]-MDTEMPxlim[[1]]/1,0)), offset.leg=.2,
+     cex=1.5)
> dev.off()
```

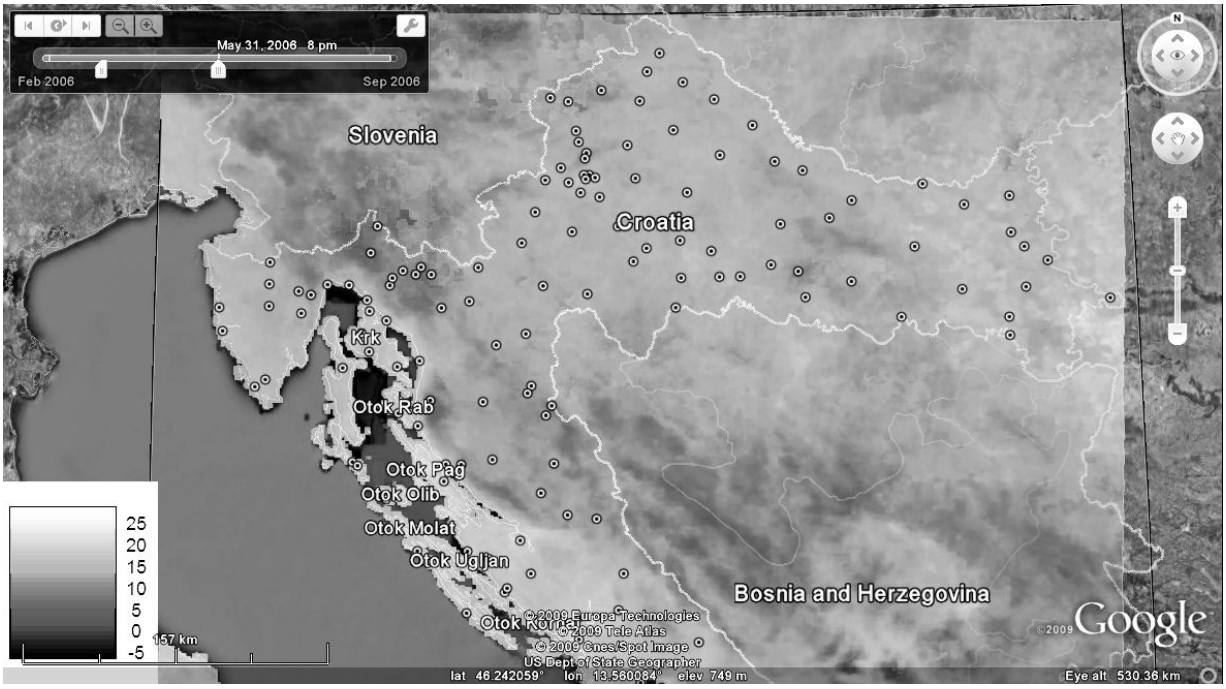


Fig. 11.8: Interpolation of temperatures visualized in Google Earth as time-series of maps. Unlike many standard GIS, Google Earth allows visual exploration of spatio-temporal data. Note from the time-bar that you can also edit the temporal support and produce *smoothed* images in time dimension.

The package maptools does not support export of time-series of maps to Google Earth. This means that we need to write the KML file ourselves. This is more simple than you anticipate, because we can again use loops (see below). In principle, we only need to respect some common headers and structure used in KML, everything else we can easily control. Note also that the KML file structure is easy to read and can be easily edited, even manually:

```
> filename <- file("MDTEMP2006.kml")
> write('<?xml version="1.0" encoding="UTF-8"?>', filename)
> write('<kml xmlns="http://earth.google.com/kml/2.2">', filename, append=TRUE)
> write('<Folder>', filename, append=TRUE)
> write('        <name>Mean Daily TEMP</name>', filename, append=TRUE)
> write('        <open>1</open>', filename, append=TRUE)
> for(i in 1:length(MDTEMP.list)) {
>   LSTname <- strsplit(MDTEMP.list[i], ".asc")[[1]][1]
# KML is compatible with POSIXct formats:
>   slice <- as.POSIXct(gsub("_", "-", strsplit(strsplit(MDTEMP.list[i],
+           ".asc")[[1]][1], "LST")[[1]][2]))
>   write('     <GroundOverlay>', filename, append=TRUE)
>   write(paste('   <name>', LSTname, '</name>',sep=""), filename, append=TRUE)
>   write('    <TimeSpan>', filename, append=TRUE)
```

```
>   write(paste('    <begin>',slice,'</begin>',sep=""), filename,
+           append=TRUE)
>   write(paste('    <end>',slice+1,'</end>',sep=""), filename, append=TRUE)
>   write('    </TimeSpan>', filename, append=TRUE)
>   write('    <color>99ffffff</color>', filename, append=TRUE)
>   write('     <Icon>', filename, append=TRUE)
>   write(paste('    <href>', getwd(), '/', LSTname, '.png</href>',sep=""),
+         filename, append=TRUE)
>   write('    <viewBoundScale>0.75</viewBoundScale>', filename,
+         append=TRUE)
>   write('    </Icon>', filename, append=TRUE)
>   write('     <altitude>50</altitude>', filename, append=TRUE)
>   write('     <altitudeMode>relativeToGround</altitudeMode>', filename, append=TRUE)
>   write('    <LatLonBox>', filename, append=TRUE)
>   write(paste('  <north>',grids.kml$ylim[[2]],'</north>',sep=""),
+         filename, append=TRUE)
>   write(paste('  <south>',grids.kml$ylim[[1]],'</south>',sep=""),
+        filename, append=TRUE)
>   write(paste('   <east>',grids.kml$xlim[[2]],'</east>',sep=""),
+         filename, append=TRUE)
>   write(paste('   <west>',grids.kml$xlim[[1]],'</west>',sep=""),
+         filename, append=TRUE)
>   write('   </LatLonBox>', filename, append=TRUE)
>   write('  </GroundOverlay>', filename, append=TRUE)
> }
> write('<ScreenOverlay>', filename, append=TRUE)
> write(paste('  <name>from',MDTEMPxlim[[1]],' to
+          ',MDTEMPxlim[[2]],'</name>',sep=""), filename, append=TRUE)
> write('   <Icon>', filename, append=TRUE)
> write(paste('    <href>',getwd(),'/legend.png</href>',sep=""), filename, append=TRUE)
> write('    </Icon>', filename, append=TRUE)
> write('    <overlayXY x="0" y="0" xunits="fraction" yunits="fraction"/>',
+       filename, append=TRUE)
> write('    <screenXY x="0" y="0" xunits="fraction" yunits="fraction"/>',
+       filename, append=TRUE)
> write('    <rotationXY x="0" y="0" xunits="fraction" yunits="fraction"/>',
+       filename, append=TRUE)
> write('    <size x="0" y="0" xunits="fraction" yunits="fraction"/>',
+       filename, append=TRUE)
> write('  </ScreenOverlay>', filename, append=TRUE)
> write('</Folder>', filename, append=TRUE)
> write('</kml>', filename, append=TRUE)
> close(filename)
```

The final output of data export is shown in Fig. 11.8. You can try to modify the original script and produce even more time-slices. In theory, we would normally want to interpolate temperatures for all 365 days and then export all these as images to Google Earth, but this is not maybe a good idea to do considering the size of the maps and the computational effort.

## 11.6 Summary points

The results of this case study demonstrate that regression-kriging models can be used also with spatio-temporal records, both of predictors and of target variables. The output pattern of the interpolated temperatures for this data set is mainly controlled with the MODIS LST images. Surprisingly, much of the variation in values can be explained by using just date. On the other hand, local variability of temperatures in the time dimension is more noisy (hence the significant nugget in Fig. 11.6) than in the geographical space. Although elevation and distance from the sea are less significant predictors of temperature than e.g. dates, it is even more important to include information on landscape to model changes in temperatures because this model is based on a physical law.

This exercises also shows that spatio-temporal prediction models are both more complex and more computationally intensive than plain spatial techniques. One significant issue not really addressed in this exercise is the problem of space-time anisotropy and space-time interactions. One should always address the issue that time units principally differ from space units, and separately quantify how fast autocorrelation decreases in time, and in space — the differences will often be large (Pebesma et al., 2007). Think of rainfall that might occur abruptly over short temporal periods, but then continuously over wide geographical areas. In this exercise we have only considered modeling the geometric anisotropy; we have completely ignored products of space-time and different scale in time dimension. An alternative would be to model the **zonal anisotropy**, where (also) the variance (sill) depends on direction.

Estimation of spatio-temporal variograms will often be cumbersome because we need to fit space-time models, for which we might not have enough space-time observations (Jost et al., 2005). Not to mention that many authors do not believe that temporal variograms are the same in both directions (past vs future). In fact, many argue whether there is any logical explanation to use observations from today to predict values from yesterday — conceptually speaking, the future does not influence the past! Nevertheless, if you compare this approach with the plain 2D techniques used to interpolate daily temperatures (Jarvis and Stuart, 2001; Schuurmans et al., 2007), you will notice that the space-time model (Eq.11.1.1) will be able to explain more variation in the temperature measurements than e.g. simple ordinary kriging using only temporally fixed measurements. Hence it is an investment worth the computational effort.

**Self-study exercises:**

(1.) What is the 95% range of values for `MDTEMP`? (HINT: use the `quantile` method and probabilities at 0.025 and 0.975.)

(2.) Are the differences between daily values of temperature also very different at other stations? (HINT: try looking at least two more stations.)

(3.) Try to fit a linear model with temperature only. Is the model much different? What would happen if we would try to model temperature as a function of time only?

(4.) How would you decrease the nugget variation in Fig. 11.6? (Consider at least two strategies and then try to prove them.)

(5.) Does it make sense to make predictions of temperature using measurements from a day or more days after (the future measurements)? How would you limit the search algorithm to one direction in time only?

(6.) What is the mean daily temperature for 1st of June 2006, at location `lon`=15.5 E, `lat`=45.0 N? (HINT: you will need to convert the time to numeric variable and then prepare a one-point spatial data layer. See section 11.5.1.)

(7.) At which locations is standard deviation of daily temperatures over the year the highest? (HINT: predict `MDTEMP` values for at least 20 slices over the whole year; then derive the mean value of temperature at each grid node and the standard deviation.)

(8.) Are the maps produced using the method explained in section 11.5.2 valid for the entire mapped region or only in the vicinity of the points? (how much are we allowed to extrapolate in geographical space? HINT: look at the variogram of residuals in Fig. 11.6.)

(9.) Obtain temperature measurements for a similar area, download the MODIS LST images (following the instructions in §4.2), and repeat this exercise with your own case study.

**Further reading:**

★ Jarvis, C. H., Stuart, N., 2001. A comparison among strategies for interpolating maximum and minimum daily air temperatures. Part II: The interaction between number of guiding variables and the type of interpolation method. Journal of Applied Meteorology 40: 1075–1084.

★ Jost, G., Heuvelink, G. B. M., Papritz, A. 2005. Analysing the space-time distributions of soil water storage of a forest ecosystem using spatio-temporal kriging. Geoderma 128 (3), 258–273.

★ Huerta, G., Sansó, B., Stroud, J.R. 2004. A spatiotemporal model for Mexico City ozone levels. Journal Of The Royal Statistical Society Series C, 53 (2): 231–248.

★ Pebesma, E. J., de Jong, K., Briggs, D. J., 2007. Visualising uncertain spatial and spatio-temporal data under different scenarios: an air quality example. International Journal of Geographical Information Science 21 (5): 515–527.

★ Schuurmans, J., Bierkens, M., Pebesma, E., Uijlenhoet, R., 2007. Automatic prediction of high-resolution daily rainfall fields for multiple extents: The potential of operational radar. Journal of Hydrometeorology 8: 1204–1224.