1 Data import

First, we'll import the stations and meteo data from .csv files, and give temperature a decent name:

```
> stations = read.csv("stations.csv")
> lapply(stations, class)

$X01.stationkennung
[1] "factor"

$X02.station.name
[1] "factor"

$X03.messnetz
[1] "factor"

$X04.koord..x
[1] "integer"

$X05.koord..y
[1] "integer"

$X06.hoehe.ueber.meer..m.
[1] "integer"

$X07.min..datum
[1] "factor"

$X08.max..datum
[1] "logical"

$X09.kanton
[1] "factor"

$X10.klimat.region.nr
[1] "integer"

$X11.landkarte
[1] "logical"

> slf_ams = read.csv("slf_ams.csv")
> lapply(slf_ams, class)

$X01.stationsabk.rzung
[1] "factor"
```

```
$X02.standortnummer
[1] "integer"

$X03.datum.und.zeit
[1] "factor"

$X04.windgeschwindigkeit..m.s.
[1] "numeric"

$X05.windrichtung..grad.
[1] "integer"

$X06.gr.sste.sekundenb.e..m.s.
[1] "numeric"

$X07.lufttemperatur...c.
[1] "numeric"

$X08.rel..luftfeuchtigkeit....
[1] "integer"

$X09.reflekt..strahlung..w.m2.
[1] "integer"

$X10.schneeh.he...cm.
[1] "integer"

$X11.schneeoberfl.chetemp....c.
[1] "numeric"
```

```
> slf_ams$temp = as.numeric(slf_ams$X07.lufttemperatur...c.)
```

2 Sanity checks

Next, we'll check if the station names

```
> levels(stations$X01.stationkennung)
```

```
[1] "DAV-3" "DAV-4" "DAV-5" "KLO-3" "PAR-2" "SLF-2" "WFJ-1" "WFJ-2"
```

agree with the slf_ams stations, defined by two fields:

```
> unique(paste(slf_ams$X01.stationsabk.rzung, slf_ams$X02.standortnummer,
+     sep = "-"))
```

```
[1] "DAV-3" "DAV-4" "DAV-5" "KLO-3" "PAR-2" "SLF-2" "WFJ-1" "WFJ-2"
```

Then, we create a common field, called station:

```
> stations$station = stations$X01.stationkennung
> slf_ams$station = paste(slf_ams$X01.stationsabk.rzung, slf_ams$X02.standortnummer,
+     sep = "-")
```

3 Time, monthly means

Now we'll work on the time field, which is now a factor that looks like this:

```
> slf_ams$X03.datum.und.zeit[1]
```

```
[1] 2002.10.01 00:00
104973 Levels: 2002.10.01 00:00 2002.10.01 00:30 ... 2006.10.01 00:00
```

and we need to manually set the format of the fields (see ?strptime):

```
> slf_ams$time = as.POSIXct(strptime(as.character(slf_ams$X03.datum.und.zeit),
+     "%Y.%m.%d %H:%M"))
```

Make the selection for february and august data:

```
> feb = format.POSIXct(slf_ams$time, "%m") == "02"
> summary(feb)
```

```
   Mode   FALSE    TRUE
logical  448706   37463
```

```
> feb[1:10]
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
> aug = format.POSIXct(slf_ams$time, "%m") == "08"
```

Make the selection for the 14:00 data; if we onlys select 14:00 we loose 2 out of 8 stations!

```
> at1400 = format.POSIXct(slf_ams$time, "%H") == "14" & format.POSIXct(slf_ams$time,
+     "%M") == "00"
> at1340 = format.POSIXct(slf_ams$time, "%H") == "13" & format.POSIXct(slf_ams$time,
+     "%M") == "40"
> afternoon = at1400 | at1340
```

Select the tables (data.frames) on [rows, columns]:

```
> temp_feb = na.omit(slf_ams[feb & afternoon, c("temp", "station")])
> temp_aug = na.omit(slf_ams[aug & afternoon, c("temp", "station")])
```

... and compute station means:

```
> T.feb = lapply(split(temp_feb$temp, as.factor(temp_feb$station)),
+     mean)
> T.aug = lapply(split(temp_aug$temp, as.factor(temp_aug$station)),
+     mean)
```

Here, `split` splits the first argument according to groups in the second, and `lapply` applies something (function in second argument) to each element of a list (1st arg.)

The function unlist creates a simple vector:

```
> out = data.frame(T.aug = unlist(T.aug), T.feb = unlist(T.feb))
> out

          T.aug     T.feb
DAV-3  8.677419 -6.755357
DAV-4  9.556098 -6.666964
DAV-5 10.742157 -5.366667
KLO-3  9.579032 -5.471429
PAR-2  9.453659 -6.125893
SLF-2 15.146341 -1.681651
WFJ-1  7.157258 -9.573451
WFJ-2  7.616129 -7.438938
```

4 Adding coordinates; matching tables

Now we will match table `stations` with `out` in order to get the coordinates and altitude of the temperature means:

```
> m = match(row.names(out), as.character(stations$X01.stationkennung))
> m

[1] 5 3 4 6 7 8 2 1
```

Check that we match right; `x[m]` puts x in order m; `all.equal` checks equality of two things

```
> all.equal(as.character(stations$X01.stationkennung[m]), row.names(out))

[1] TRUE
```

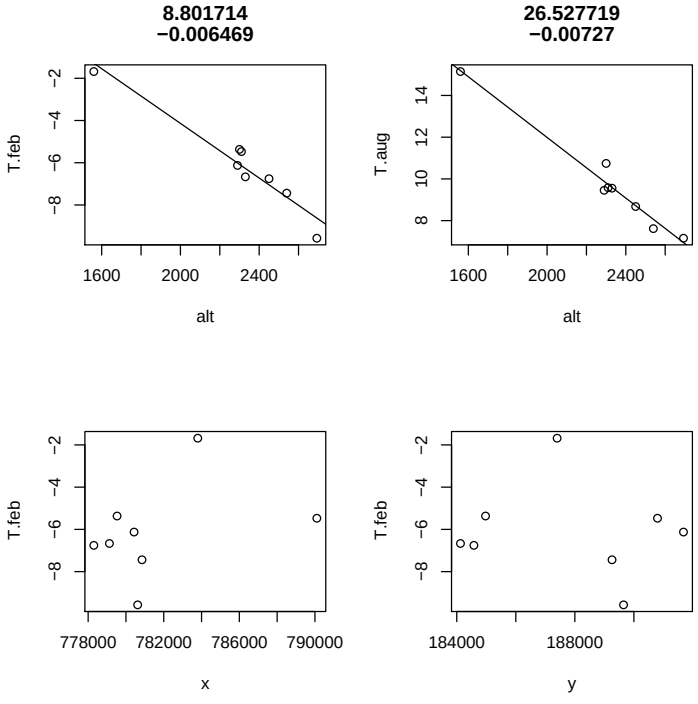
```
> out$x = stations$X04.koord..x[m]
> out$y = stations$X05.koord..y[m]
> out$alt = stations$X06.hoehe.ueber.meer..m.[m]
> out

          T.aug     T.feb      x      y  alt
DAV-3  8.677419 -6.755357 778300 184580 2450
DAV-4  9.556098 -6.666964 779125 184125 2330
```

```
DAV-5 10.742157 -5.366667 779530 184975 2300
KLO-3  9.579032 -5.471429 790100 190800 2310
PAR-2  9.453659 -6.125893 780430 191680 2290
SLF-2 15.146341 -1.681651 783800 187400 1560
WFJ-1  7.157258 -9.573451 780620 189650 2693
WFJ-2  7.616129 -7.438938 780850 189260 2540
```

5 Regression, regression diagnostics

We can plot some regression diagnostics:

```
> par(mfrow = c(2, 2))
> f1 = T.feb ~ alt
> plot(f1, out)
> abline(lm(f1, out))
> title(round(coefficients(lm(f1, out)), 6))
> f2 = T.aug ~ alt
> plot(f2, out)
> abline(lm(f2, out))
> title(round(coefficients(lm(f2, out)), 6))
> plot(T.feb ~ x, out)
> plot(T.feb ~ y, out)
```



6 Interpolating altitude, using idw

Next, we import the altitude points. Package `rgdal` deals with raster and vector import; it loads package sp.

```
> library(rgdal)

Geospatial Data Abstraction Library extensions to R successfully loaded
Loaded runtime: GDAL 1.4.1.0, released 2007/04/09

> bm1197p = readOGR("../GISAufbaukurs_WS0708/data/1197-1198/DHM25/basis/point",
+     "bm1197p")

OGR data source with driver: ESRI Shapefile
Source: "../GISAufbaukurs_WS0708/data/1197-1198/DHM25/basis/point", layer: "bm1197p"
with  800  rows and  4  columns

> bm1198p = readOGR("../GISAufbaukurs_WS0708/data/1197-1198/DHM25/basis/point",
+     "bm1198p")

OGR data source with driver: ESRI Shapefile
Source: "../GISAufbaukurs_WS0708/data/1197-1198/DHM25/basis/point", layer: "bm1198p"
with  784  rows and  4  columns

> summary(bm1197p)

Object of class SpatialPointsDataFrame
Coordinates:
               min      max
coords.x1 777510.9 794962.5
coords.x2 182009.4 193996.9
coords.x3   1179.0   3085.0
Is projected: NA
proj4string : [NA]
Number of points: 800
Data attributes:
    OBJECTID              OBJECTVAL   OBJECTORIG   YEAROFCHAN
 Min.   :8641614   Hoehenkote  :799   LK25:800   Min.   :1985
 1st Qu.:8641814   Seebodenkote:  1              1st Qu.:1985
 Median :8642014                                 Median :1985
 Mean   :8642014                                 Mean   :1985
 3rd Qu.:8642213                                 3rd Qu.:1985
 Max.   :8642413                                 Max.   :1985

> summary(bm1198p)

Object of class SpatialPointsDataFrame
Coordinates:
```

```
               min      max
coords.x1 795017.1 812479.6
coords.x2 182009.4 193915.6
coords.x3   1298.0   3411.0
Is projected: NA
proj4string : [NA]
Number of points: 784
Data attributes:
    OBJECTID            OBJECTVAL   OBJECTORIG   YEAROFCHAN
 Min.   :8642414   Hoehenkote:784   LK25:784   Min.   :1985
 1st Qu.:8642610                               1st Qu.:1985
 Median :8642806                               Median :1985
 Mean   :8642806                               Mean   :1985
 3rd Qu.:8643001                               3rd Qu.:1985
 Max.   :8643197                               Max.   :1985
```

The first two commands set the sets bpy.colors() colour ramp; then the two point data files are merged (rbind: row bind), and converted to a spatial data set:

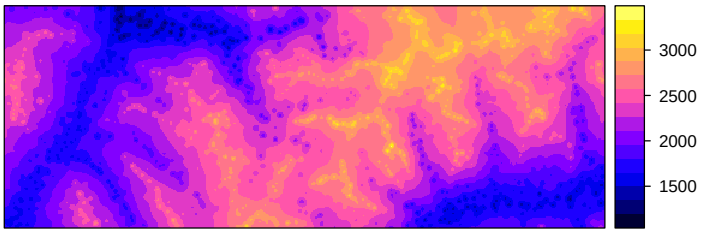
```
> library(lattice)
> trellis.par.set(sp.theme())
> bmp = rbind(bm1197p, bm1198p)
> bmp = as.data.frame(bmp)
> names(bmp) = c(names(bmp)[1:4], "x", "y", "alt")
> summary(bmp)

    OBJECTID              OBJECTVAL    OBJECTORIG    YEAROFCHAN
 Min.   :8641614   Hoehenkote  :1583   LK25:1584   Min.   :1985
 1st Qu.:8642010   Seebodenkote:   1               1st Qu.:1985
 Median :8642406                                   Median :1985
 Mean   :8642406                                   Mean   :1985
 3rd Qu.:8642801                                   3rd Qu.:1985
 Max.   :8643197                                   Max.   :1985
       x                y               alt
 Min.   :777511   Min.   :182009   Min.   :1179
 1st Qu.:785796   1st Qu.:184869   1st Qu.:1884
 Median :794789   Median :187925   Median :2314
 Mean   :794951   Mean   :187871   Mean   :2269
 3rd Qu.:804052   3rd Qu.:190775   3rd Qu.:2642
 Max.   :812480   Max.   :193997   Max.   :3411

> class(bmp)

[1] "data.frame"

> coordinates(bmp) = ~x + y
> class(bmp)
```

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

```
> spplot(bmp["alt"])
> grd = makegrid(bmp, n = 1e+05)
> names(grd) = c("x", "y")
> coordinates(grd) = ~x + y
> grd = as(grd, "SpatialPixels")
> library(gstat)
> grd.alt = idw(alt ~ 1, bmp, grd)
```

```
[inverse distance weighted interpolation]
```

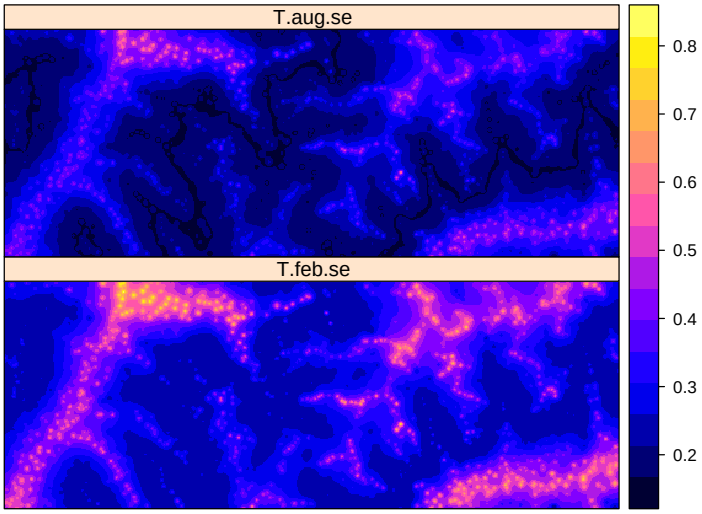
```
> print(spplot(grd.alt["var1.pred"]))
```



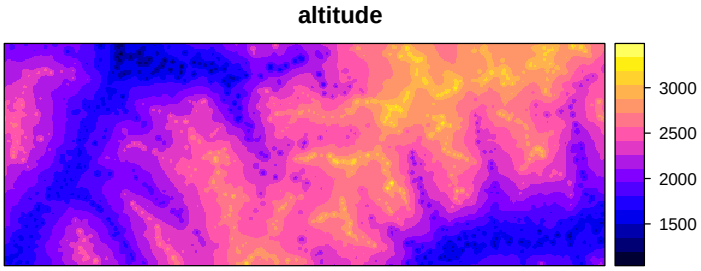
7 Spatial prediction with a regression model

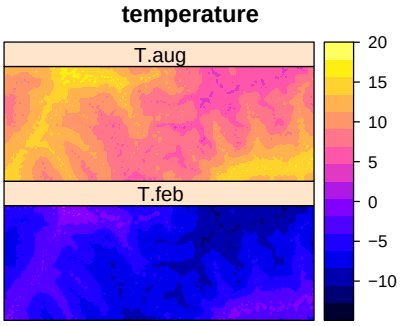
```
> grd.alt$alt = grd.alt$var1.pred
> grd.alt$T.feb = predict(lm(f1, out), grd.alt)
> grd.alt$T.aug = predict(lm(f2, out), grd.alt)
> print(spplot(grd.alt[c("T.feb", "T.aug")]))
```

```
> grd.alt$T.feb.se = predict(lm(f1, out), grd.alt, se.fit = T)$se.fit
> grd.alt$T.aug.se = predict(lm(f2, out), grd.alt, se.fit = T)$se.fit
> print(spplot(grd.alt[c("T.feb.se", "T.aug.se")]))
```

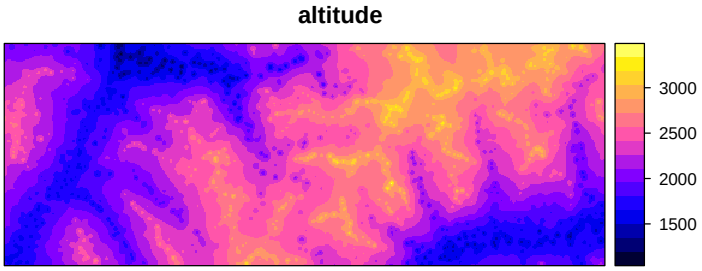


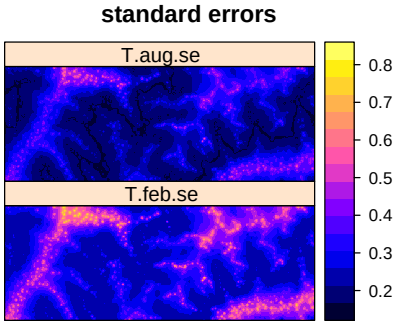

```
> print(spplot(grd.alt[c("alt")], main = "altitude"), split = c(1,
+     1, 1, 2), more = T)
> print(spplot(grd.alt[c("T.feb", "T.aug")], main = "temperature"),
+     split = c(1, 2, 1, 2), more = F)
```






```
> print(spplot(grd.alt[c("alt")], main = "altitude"), split = c(1,
+     1, 1, 2), more = T)
> print(spplot(grd.alt[c("T.feb.se", "T.aug.se")], main = "standard errors"),
+     split = c(1, 2, 1, 2), more = F)
```





8 Ordinary kriging of altitude

```
> library(gstat)
> v = variogram(alt ~ 1, bmp)
> v
```

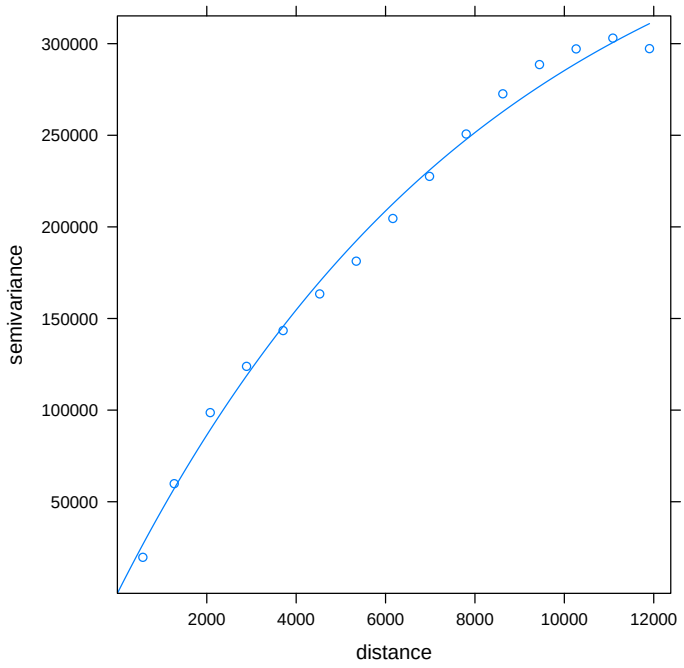
```
      np       dist     gamma dir.hor dir.ver   id
1   5839   569.9613  19670.18       0       0 var1
2  17454  1271.5887  59863.63       0       0 var1
3  27096  2077.2693  98639.08       0       0 var1
4  35526  2889.7851 123895.60       0       0 var1
5  42545  3707.5985 143416.36       0       0 var1
6  47600  4526.5666 163425.41       0       0 var1
7  52173  5344.1203 181295.12       0       0 var1
8  55361  6163.5497 204598.89       0       0 var1
9  56802  6983.8903 227578.54       0       0 var1
10 58288  7804.0773 250701.92       0       0 var1
11 57726  8623.4404 272643.68       0       0 var1
12 55889  9443.8364 288598.30       0       0 var1
13 53647 10263.7473 297153.43       0       0 var1
14 50750 11085.3437 303038.14       0       0 var1
```

```
15 46832 11904.9439 297291.43       0       0 var1
```

```
> plot(v)
> v.fit = fit.variogram(v, vgm(1, "Exp", 3000))
> v.fit
```

```
  model    psill    range
1   Exp 413491.0 8538.357
```

```
> print(plot(v, v.fit))
```



When executed, the following will break with an obscure error message:

```
> grd.alt.kr = krige(alt ~ 1, bmp, grd, v.fit, nmax = 30)
```

which is due to a duplicate observation. We can find the duplicate measurement(s) that cause(s) this error:

```
> zerodist(bmp)
```

```
     [,1] [,2]
[1,] 1193 1198
```

and check if the attribute is duplicate as well:

```
> bmp[c(1193, 1198), ]

          coordinates OBJECTID  OBJECTVAL OBJECTORIG YEAROFCHAN  alt
1193 (796161, 187155)  8642806 Hoehenkote       LK25       1985 2322
1198 (796161, 187155)  8642811 Hoehenkote       LK25       1985 2322
```

Now, we can krige while de-selecting one of the duplicates:

```
> grd.alt.kr = krige(alt ~ 1, bmp[-1193, ], grd, v.fit, nmax = 30)

[using ordinary kriging]

> print(spplot(grd.alt.kr[1]))

> print(spplot(grd.alt.kr[1], cuts = 25, main = "kriged surface"),
+     split = c(1, 1, 1, 2), more = TRUE)
> grd.alt.kr$se = sqrt(grd.alt.kr$var1.var)
> print(spplot(grd.alt.kr["se"], cuts = 25, main = "kriging standard error surface"),
+     split = c(1, 2, 1, 2), more = FALSE)
```