Tutorial: Regional mapping of climate variables from point samples

Ordinary Least Squares trend
Generalized Least Squares trend
Regression Kriging
Kriging with External Drift
Generalized Additive Models trend
Data-driven methods: Regresion trees, Random Forests, Cubist
Thin-plate splines
Ordinary kriging, inverse-distance, Thiessen polygons

D G Rossiter
Cornell University, Section of Soil & Crop Sciences
ISRIC–World Soil Information

May 11, 2024

Contents

循序渐进
"Advance step-by-step; slow but steady wins the race"

1 Introduction

This exercise presents various methods for regional mapping of climate variables from station information. The methods all relate to the **universal model of spatial distribution** of a variable:

$$Z(\mathrm{s}) = Z^*(\mathrm{s}) + \varepsilon(\mathrm{s}) + \varepsilon'(\mathrm{s}) \tag{1}$$

(s) : a location in space, designated by a **vector** of coördinates;

Z(s) : **true** (unknown) value of some property at the location;

- when modelled, expressed as **most likely** value and some **uncertainty**, or as a **probability distribution**

Z^*(s) : **deterministic** component, due to some known or modelled **non-stochastic** process which operates over the entire region;

ε(s) : locally **spatially-autocorrelated stochastic** component;

ε'(s) : pure ("white") **noise**, no structure.

In regional mapping, the deterministic component Z^*(s) is called a **trend**, and models of these are called **trend surfaces**.

The focus is on prediction by Regression Kriging with Generalized Least Squares fitting of linear models (RK-GLS). This technique accounts for spatial correlation among the residuals ε(s) of the trend surface Z^*(s), fitting both together.

We contrast RK-GLS with predictions made by:

1. OLS trend surface;
2. GLS trend surface;
3. Kriging with External Drift (KED);
4. Data-driven (machine-learning) methods: Regression Trees, Random Forests, Cubist;
5. Generalized Additive Models (GAM);
6. Thin-plate splines;
7. Local interpolators: Ordinary Kriging (OK), Inverse Distance Weighting (IDW), Thiessen polygons.

This exercise also gives some practice in importing, manipulating, and verifying a large dataset, and also gives some practice with the `ggplot2`, `nlme`, `sf`, `gstat`, `rpart`, `randomForest`, `ranger`, `Cubist`, `raster`, `plotKML` and `fields` R packages.

The exercise is organized as a set of **discussions**, **tasks**, **R code** to complete the tasks, self-study **questions** (with answers) to test your understanding, and a few **challenges**. §15 is an exercise to apply these techniques to a different study area and/or a different climate variable.

> **Note:** The source for this document is a text file that includes ordinary LaTeX source and "chunks" of R source code, using the Noweb[1] syntax. The formatted R source code, R text output, and R graphs in this document were automatically generated and incorporated into a LaTeX source file by running the Noweb source document through R, using the `knitr` package [30]. The LaTeX source was then compiled by LaTeX into the PDF you are reading. The source code is provided in file `MappingRegionalClimate.R`.

2 Example dataset

In this tutorial we use **agricultural climate** as an example of a spatial points dataset. The Northeast Regional Climate Center[2] has kindly provided a set of point ESRI shapefiles with various variables related to agricultural climate measured from 1971-2000. This dataset was devloped by the Unifed Climate Access Network (UVAN) network, and covers the entire United States of America and its dependencies. It consists of several variables relevant for agricultural climate: mean monthly and annual precipitation, mean monthly and annual temperature, annual freeze free period base 32°F and 28°F[3], annual extreme minimum temperature, and monthly and annual growing degree-days , base 50°F (relevant for C4 crops) and base 40°F (relevant for C3 crops)[4].

In this tutorial we use as an example *growing degree-days, base 50°F*[5] (GDD50).

> **Note:** Temperatures T are expressed in °F in the USA, Bahamas, Belize, the Cayman Islands, and Palau. The conversion is 1°C = (°F - 32) × (5/9).

These are defined for one day as [15]:

$$\text{GDD50} = \max\left([(T_{\text{max}} + T_{\text{min}})/2] - T_{\text{base}}, 0\right) \tag{2}$$

where $T_{\text{base}} = 50$.

For example, a day with maximum 86°F and minimum 60°F would account for $146/2 - 50 = 23$ GDD50. These are summed over some time period; we will use the sum over the year, i.e., Annual GDD50. These heat units are well-correlated with seasonal maize growth and are used to select maize varieties on the basis of the number of GDD50 required to reach maturity.

The aim of all the methods in this tutorial is to **predict** GDD50 at an arbitrary location, using the records from the known stations, possibly also using covariables that are available over the whole region: latitude, longitude and elevation above sea level, as well as local spatial structure. While predicting, we also can examine the predictive models to **understand** the causes for Annual GDD50 spatial variation.

[1] http://www.cs.tufts.edu/~nr/noweb/
[2] http://www.nrcc.cornell.edu
[3] -2.2 and 0°C
[4] 4.44 and 10°C, respectively
[5] 10°C

In §13 we compare models and predictions of this variable with models and predictions for others from the same dataset.

For the purposes of this tutorial, we have prepared the dataset of GDD50, along with State boundaries and a set of environmental covariables associated with agricultural climate. If you are interested in how this was prepared, for example if you want to apply this method for other agricultural climate variables or in other regions of the USA, the details are in the related tutorial, "Tutorial: Regional mapping of climate variables from point samples: Data preparation".

3 Data exploration

Task 1 : Load the "Simple Features" spatial geometry package to be used in this and subsequent sections. •

```
require(sf)
```

Task 2 : Load the GDD50 dataset and supporting files. •

The `load` function is used to load R objects from an `RData` format file. The `save` function stores the object name(s) along with the object(s), so that after `load`, there are new objects in the workspace, with these names. Setting the optional `verbose` argument to `load` shows the objects that were loaded from the `RData` format file.

```
load(file="./StationsDEM_covariates.RData", verbose=TRUE)
```

```
## Loading objects:
##   ne
##   ne.m
##   ne.df
##   state.ne
##   state.ne.m
##   dem.ne.m.sf
##   dem.ne.m.df
##   dem.ne4.m
##   ne.crs
```

3.1 Feature-space summary

Task 3 : Summarize the coördinates elevations and annual GDD50, so we know the range of values we will use in the analysis. •

```
summary(ne.df[,c("E", "N", "ANN_GDD50","ELEVATION_")])
```

```
##        E                  N               ANN_GDD50      ELEVATION_
##  Min.   :-375745   Min.   :-393923   Min.   : 795   Min.   :   5.0
##  1st Qu.:-156768   1st Qu.:-208100   1st Qu.:2100   1st Qu.: 330.0
##  Median :  47130   Median :-105357   Median :2463   Median : 711.0
##  Mean   :   3231   Mean   : -77213   Mean   :2518   Mean   : 799.7
##  3rd Qu.: 151648   3rd Qu.:  36720   3rd Qu.:2930   3rd Qu.:1160.0
##  Max.   : 318960   Max.   : 276614   Max.   :4021   Max.   :3950.0
```

The stations go from almost at sea level up to more than 4 000' (1 220 m.a.s.l.). The maximum annual GDD50 is more than five times the minimum. So in both we have a good range to find statistical relations.

3.2 Station locations

Task 4 : Display the locations of the weather stations. •

We use base graphics for this. The coördinates of the `sf` object are a two-dimensional array, so this is a scatterplot, also called "x-y plot", with the axes being the coördinates. The generic `plot` method, when given a two-dimensional matrix, calls the internal function `plot.xy`. Colours are from the default palette[6], according to the level number of the factor. These are in alphabetic sort order, so `NJ` has colour 1 (black), `NY` has colour 2 (red), etc. We also use the `labels` argument to the `text` function to display the first four characters of the station name.

We then can add the state boundaries to help visualize the study area.

The boundary lines are extracted from the polygons with the `st_cast` method, that "casts" (a computer science term) one data type into another, in this case from a `MULTIPOLYGON` to a `"MULTILINESTRING"`, and then reduced to only the geometry, i.e., removing the attributes such as State ID and name, with the `st_geometry` method.

Here the `plot` function has an extra argument, `add`, specifying that we are adding to a plot, not starting a new one. The `col` argument specifies the line colour, and the `lwd` "line width" argument specifies how much the default line width should be expanded or contracted. The result is Fig. 1.

```
unique(st_geometry_type(state.ne.m))

## [1] MULTIPOLYGON
## 18 Levels: GEOMETRY POINT LINESTRING POLYGON ... TRIANGLE

state.ne.m.boundary <- st_geometry(st_cast(state.ne.m, "MULTILINESTRING"))
unique(st_geometry_type(state.ne.m.boundary))

## [1] MULTILINESTRING
## 18 Levels: GEOMETRY POINT LINESTRING POLYGON ... TRIANGLE

plot(st_coordinates(ne.m), asp=1, xlab="E", ylab="N", pch=20)
text(st_coordinates(ne.m),
     labels=substr(ne.m$STATION_NA, 1, 4), cex=0.5,
     pos=2, col=as.numeric(ne.m$STATE))
plot(state.ne.m.boundary, add=TRUE, col="darkgray", lwd=2, sf_max.plot = 1)
grid()
```

3.3 Postplots

We now display the locations, but also using the point symbol represent the data value. This is called a **postplot**, because it "posts" (puts into geographic position) the data values.

Task 5 : Display a postplot of the annual growing degree days above 50° F

[6] See the help for the `palette` function

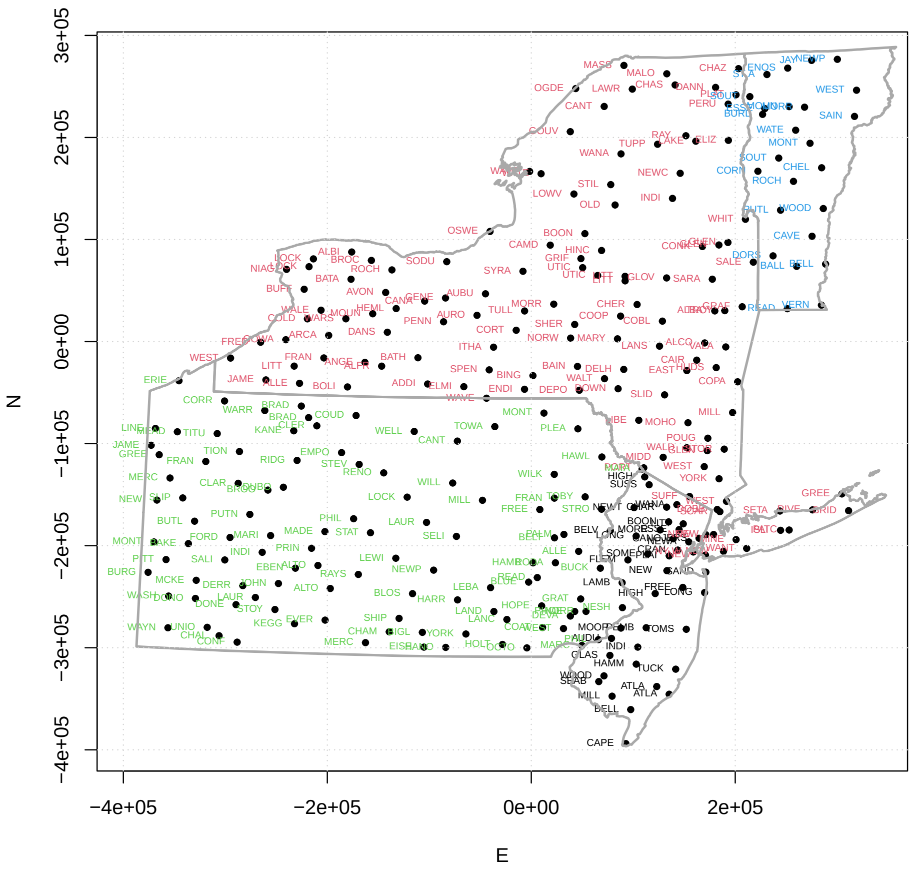


Figure 1: Location of weather stations

(attribute `ANN_GDD50`). •

For this we use the "Grammar of Graphics" [27] as implemented in the `ggplot2` package [25, 26]. This is part of the so-called "tidyverse"[7] set of packages from Hadley Wickham. The tidyverse web site has a comprehensive introduction to `ggplot2`[8].

```
require(ggplot2)
```

The `ggplot2` concept is that a graphic is *initialized* with `ggplot` and then elements are *added* to the graphic, each separated by a `+` operator, which in this context means "add to the graph", not arithmetic addition.

In this example we first open the plot with the `ggplot` function, specifying the source of the data for the plot with the `data` argument, and then:

1. specify how the graph should be set up on the page with the `aes` function: the `x` axis from the E coördinate and the `y` axis from the N coördinate;
2. add points with the the `geom_point` "geometry" function;
3. add axis labels with the `xlab` and `ylab` functions;
4. add a fixed-scale coördinate system for the graphic, with `coord_fixed`.

The points have an "aesthetic" (how they are displayed), specified with the `aes` function, and the name of the data frame where the names in the aesthetic can be found. We make the `size` of the points proportional by degree days, and the `colour` of the points by elevation, so we can visually assess if there is any relation both with coördinates and with elevation. We specify a printing character with the `shape` argument to `geom_point`.

```
ggplot(data=ne.df) +
    aes(x=E, y=N) +
    geom_point(aes(size=ANN_GDD50,
                   colour=ELEVATION_),
               shape=20) +
        xlab("E") +  ylab("N") + coord_fixed()
```

The result is Fig. 2.

Q1 : *Does there appear to be any regional trend with North or East? with elevation?* *Jump to A1* •

The `sf` object can be directly plotted, without reducing it to a data frame, using the `geom_sf` "class `sf` geometry" method from the `ggplot2` package.

```
ggplot() +
  geom_sf(data = ne.m,
          aes(size=ANN_GDD50,
              colour=ELEVATION_), shape = 20) +
  labs(x = "Longitude", y = "Latitude",
       size = "GDD base 50F",
       colour = "Elevation, feet a.s.l.") +
  geom_sf(data = state.ne.m.boundary, col = "darkgray", size = 2)
```

[7] https://www.tidyverse.org
[8] https://ggplot2.tidyverse.org

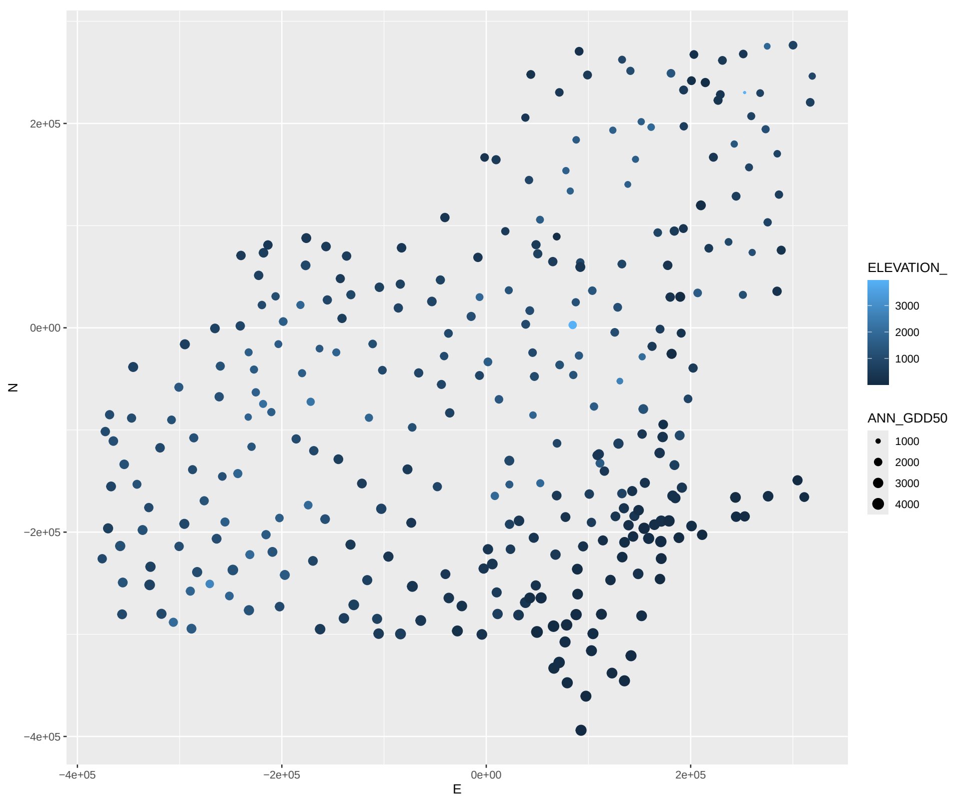


Figure 2: Annual Growing Degree days, 50F

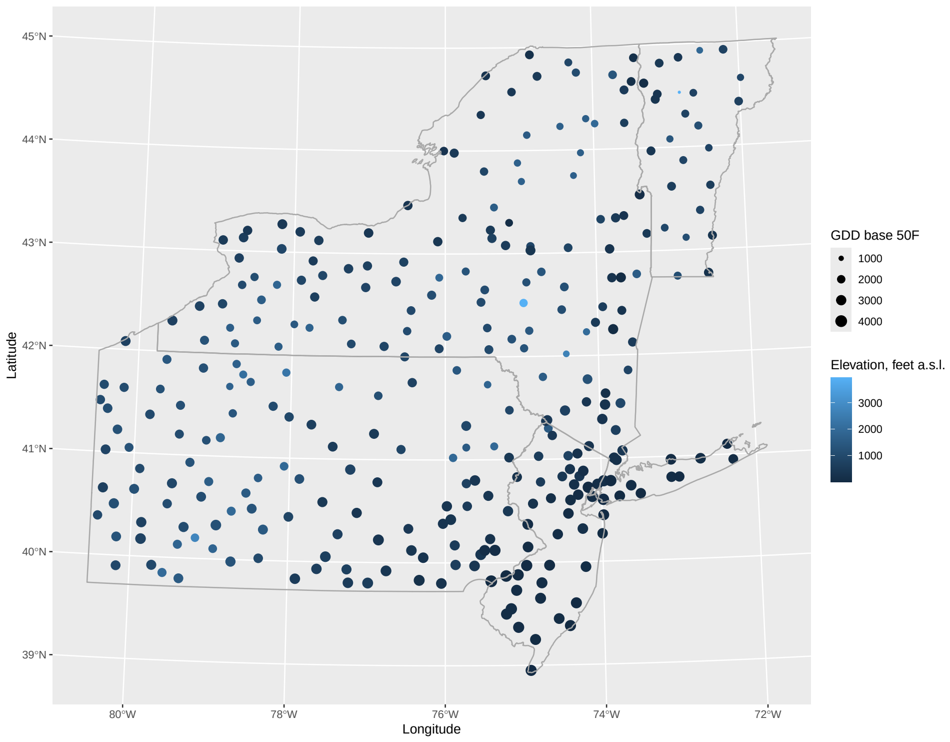


Figure 3: Annual Growing Degree days, base 50F

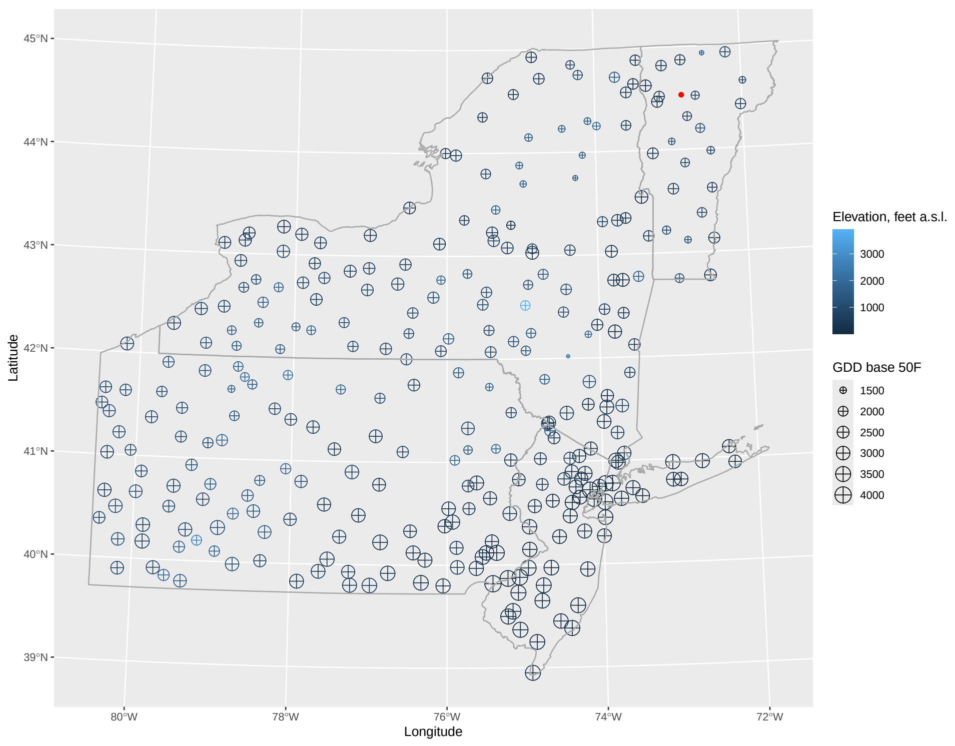


Figure 4: Annual Growing Degree days, base 50F

Note the one very small GDD50, which causes the remainer of the points to look quite similar. We can remove this for display purposes, and then show it separately as a red point.

```
(which.lowest.gdd <- which.min(ne.m$ANN_GDD50))

## [1] 293

ggplot() +
  geom_sf(data = ne.m[-which.lowest.gdd, ],
          aes(size=ANN_GDD50,
              colour = ELEVATION_), shape = 10) +
  geom_sf(data = ne.m[which.lowest.gdd, ], colour = "red") +
  labs(x = "Longitude", y = "Latitude",
       size = "GDD base 50F",
       colour = "Elevation, feet a.s.l.") +
  geom_sf(data = state.ne.m.boundary, col = "darkgray", size = 2)
```

Still quite a wide range, but now easier to visualize.

3.4 * Viewing in geographic context

To check station locations, and to relate them to geographic features (land use, water bodies, cities …) an easy method is to export the points as a KML file for display in Google Earth. Features in Google Earth must be in geographic coördinates (long/lat), on the WGS84 ellipsoid. As we saw

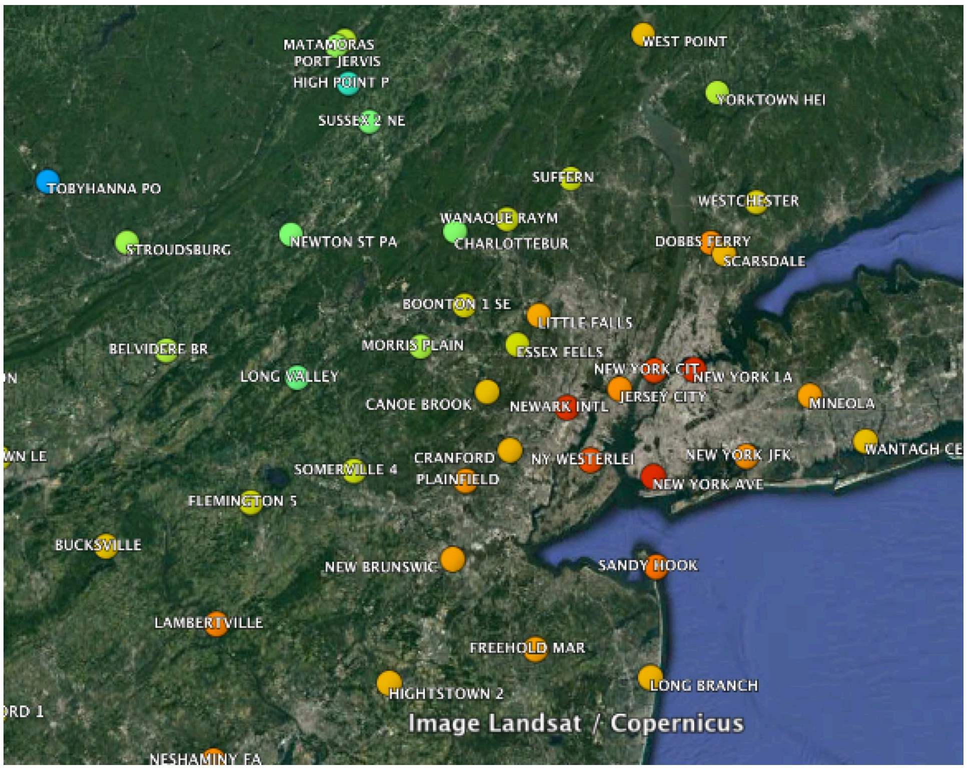


Figure 5: Stations shown on Google Earth

above, these station records do have geographic coördinates, but on the NAD27 datum, which uses the Clarke 1866 ellipsoid. So we need to do a datum transformation from one representation of the Earth's shape (and hence the location on the surface of geographic coördinates) to another. For this we use the `st_transform` method of the `sf` package.

We use three functions from the `plotKML` package: `kml_open`, `kml_layer`, and `kml_close`. The colour ramp is from the target variable, here annual growing-degree days, and the points are labelled with the station name.

```
ne.wgs84 <- st_transform(ne.m, st_crs(4326))
            # this is the EPSG code for global WGS84 long/lat
require(plotKML)
shape = "http://maps.google.com/mapfiles/kml/pal2/icon18.png"
station.names = substr(ne.wgs84$STATION_NA,1,12)
kml_open(file.name='ne.stations.kml')

## KML file opened for writing...

kml_layer(ne.wgs84, colour=ANN_GDD50,
   colour_scale=SAGA_pal[[1]], shape=shape,
   points_names=station.names)

## Writing to KML...

kml_close('ne.stations.kml')

## Closing  ne.stations.kml
```

Open the resulting KML file in Google Earth to see the station locations in their geographic context; see Figure 5.

4 Trend surface: a linear model solved by Ordinary Least Squares

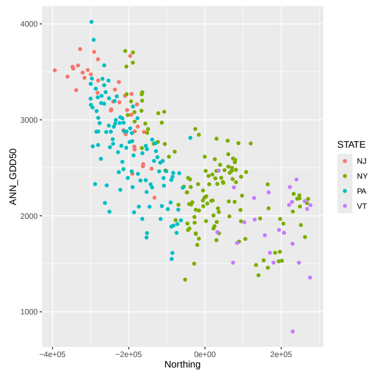
An obvious approach to prediction is to develop a model of GDD50 based on one or more of the covariables in the dataframe: (1) Easting, (2) Northing coördinates and (3) elevation. We know that in general higher elevations and more northerly latitudes are cooler; in this Atlantic region maybe the more easterly longitudes are warmer. We saw these relations spatially with the 2.5D postplots of the previous section, here we look at the relations in feature space for each possible covariable separately.

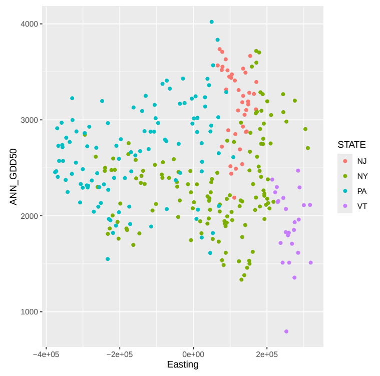
4.1 Exploring the relation between predictors and predictand

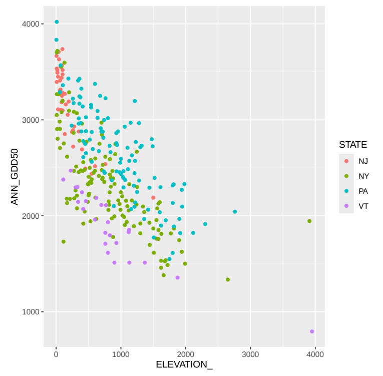
Task 6 : Display the relation between `ANN_GDD50` and covariates elevation and coordinates; colour the scatterplot points by State. •

```
p1 <- ggplot() +
    geom_point(
        aes(x=ELEVATION_, y=ANN_GDD50, colour=STATE),
            data=ne.df
    )
p2 <- ggplot() +
    geom_point(
        aes(x=E, y=ANN_GDD50, colour=STATE),
            data=ne.df
    ) + xlab("Easting")
p3 <- ggplot() +
    geom_point(
        aes(x=N, y=ANN_GDD50, colour=STATE),
            data=ne.df
    ) + xlab("Northing")
```

```
print(p3)
print(p2)
print(p1)
```







We now examine the scatterplots to see if these relations are useful predictors of GDD50, and if so, does the relation appear to be linear, or if a transformation to linearity is needed.

Q2 : *Describe the relations between GDD50 and the three possible predictors.* *Jump to A2*

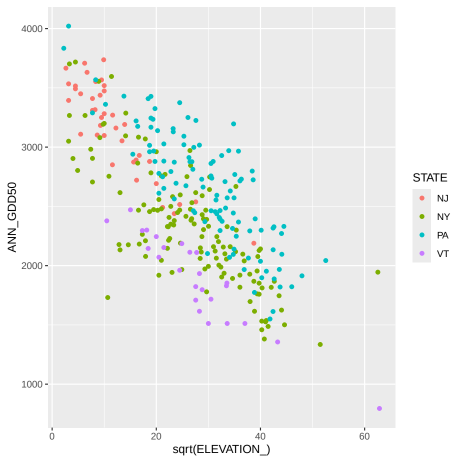


Figure 6: GDD50 vs. square root of elevation

•

The relation of GDD50 with elevation looks parabolic, so try a square-root transform.

Task 7 : Re-display the relation between GDD50 and elevation, but with the elevation square-root transformed to correspond to the inverse parabolic shape perceived in the original graph. •

```
ggplot() +
    geom_point(
        aes(x=sqrt(ELEVATION_), y=ANN_GDD50, colour=STATE),
            data=ne.df
    )
```

A square-root transformation of elevation looks quite linear.

> **Note:** This does not agree with theory, in that the adiabatic lapse rate (due to thinner air) of temperature with elevation is linear.

Task 8 : Compute the linear correlations between each predictor and GDD50. •

The `cor` function computes bivariate correlations; the default method is Pearson product-moment (linear) correlation.

```
with(ne.df, cor(ANN_GDD50, N))

## [1] -0.728741

with(ne.df, cor(ANN_GDD50, E))

## [1] -0.009425215

with(ne.df, cor(ANN_GDD50, sqrt(ELEVATION_)))

## [1] -0.7468479
```

Q3 : *Which linear correlations are strongest? What does the sign (±) of the correlation indicate?* *Jump to A3* •

4.2 OLS fit to the linear model

We first fit a linear model with the strongest single factor.

Task 9 : Fit a linear model, using ordinary least squares (OLS), of annual GDD predicted by the square root of elevation. Display the model summary. •

The workhorse `lm` "linear modelling" function fits the model.

```
m.ols.elev <- lm(ANN_GDD50 ~ sqrt(ELEVATION_), data=ne.df)
summary(m.ols.elev)

##
## Call:
## lm(formula = ANN_GDD50 ~ sqrt(ELEVATION_), data = ne.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1346.24  -278.03     2.65   258.74  1012.87
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      3472.287     53.520   64.88   <2e-16 ***
## sqrt(ELEVATION_)  -37.000      1.893  -19.55   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 382.3 on 303 degrees of freedom
## Multiple R-squared:  0.5578,Adjusted R-squared:  0.5563
## F-statistic: 382.2 on 1 and 303 DF,  p-value: < 2.2e-16
```

Q4 : *How much of the variability of GDD50 over the four states is explained by this model?* *Jump to A4* •

This is not so successful, so we try to use both predictors that showed a good correlation with GDD50.

Task 10 : Fit a linear model, using ordinary least squares (OLS), of annual GDD predicted by the additive effects of Northing and square root of elevation. Display the model summary. •

Again we use the `lm` "linear modelling" function to fit the model:

```
m.ols <- lm(ANN_GDD50 ~ sqrt(ELEVATION_) + N, data=ne.df)
summary(m.ols)

##
## Call:
## lm(formula = ANN_GDD50 ~ sqrt(ELEVATION_) + N, data = ne.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -899.15 -156.45   -9.78  153.12  660.08
```

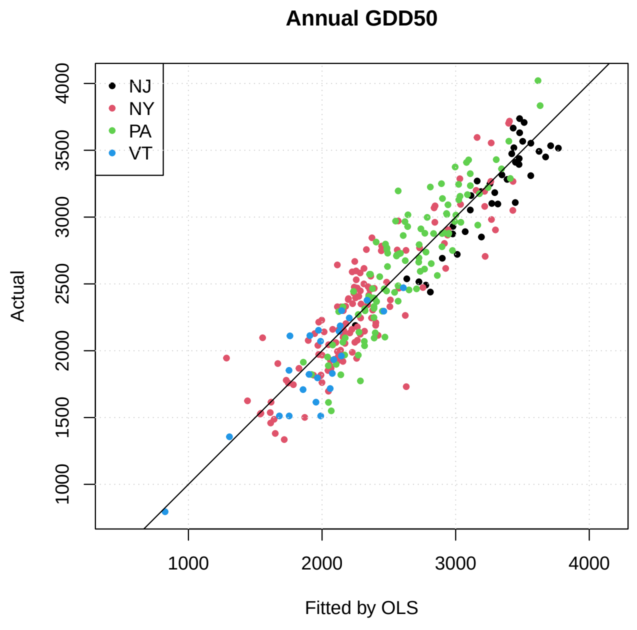


Figure 7: Actual vs. predicted, OLS model

```
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       3.119e+03  3.409e+01   91.49   <2e-16 ***
## sqrt(ELEVATION_) -2.924e+01  1.138e+00  -25.69   <2e-16 ***
## N                -1.981e-03  8.051e-05  -24.60   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 220.9 on 302 degrees of freedom
## Multiple R-squared:  0.8528,Adjusted R-squared:  0.8518
## F-statistic: 874.9 on 2 and 302 DF,  p-value: < 2.2e-16
```

Now the model explains much more (85.2%) of the variation in GDD50; we conclude that the North coördinate is an important predictor.

Task 11 : Plot the actual vs. fits as a scatterplot, adding a 1:1 line. •

```
plot(ne.m$ANN_GDD50 ~ fitted(m.ols),
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by OLS", ylab="Actual",
     main="Annual GDD50")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid()
abline(0,1)
```

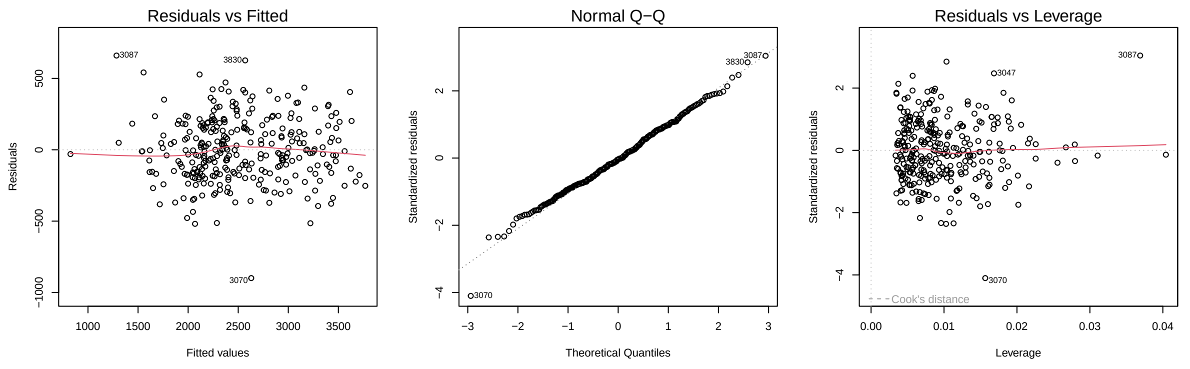
Q5 : *How well does the model fit the observations? Are any observations poorly-fit?* *Jump to A5* •

OLS modelling has strong assumptions, some of which we now examine.

- Residuals are normally-distributed;
- No relation between the **fitted value** and the **distribution of the residuals**, either their mean or their spread;
- No observations with high **leverage** (i.e., influence on the regression coefficients) have large residuals. This is measured with Cook's Distance, which gives the effect of deleting an observation on the regression coefficients.

Task 12 : Examine the feature-space model diagnostics for these three requirements. •

```
par(mfrow=c(1,3))
plot(m.ols, which=c(1,2,5))
par(mfrow=c(1,1))
```



Q6 : *Is there any relation between fits and residuals? Is the variability of the residuals approximately the same across the range of fits? Are the residuals normally-distributed? Are any high-leverage points inconsistent with the others?* *Jump to A6* •

We will look at the suspect observations in detail in the next §4.3.

We've now built a model, and we can use it to predict at any point for which we know the values of the predictors, i.e., the Northing and elevation.

4.3 Data cleaning

But first, we look at some of the most poorly-fitted points, to determine if the poor fit is caused by an error in the database, or by an incorrect model.

Task 13 : Find and display the points with the maximum and minimum model residuals. •

The `which.min` and `which.max` functions find the index (position) in a vector of the minimum and maximum value, respectively.

```
(ix <- c(which.max(residuals(m.ols)), which.min(residuals(m.ols))))

## 3087 3070
##  113   96

ne.df[ix,]

##      STATION_ID STATE    STATION_NA LATITUDE_D LONGITUDE_ ELEVATION_
## 3087     305113    NY MARYLAND 6 SW      42.52     -74.97       3911
## 3070     303889    NY HINCKLEY 2 SW      43.30     -75.15        114
##      OID_ COOP_ID STATE_1      STN_NAME LAT_DD LONG_DD ELEV_FT
## 3087 3085  305113      NY MARYLAND 6 SW     43     -75    3911
## 3070 3068  303889      NY HINCKLEY 2 SW     43     -75     114
##      JAN_GDD50 FEB_GDD50 MAR_GDD50 APR_GDD50 MAY_GDD50 JUN_GDD50
## 3087         0         0         0        23       177       387
## 3070         0         0         0        22       155       349
##      JUL_GDD50 AUG_GDD50 SEP_GDD50 OCT_GDD50 NOV_GDD50 DEC_GDD50
## 3087       542       496       251        66         2         1
## 3070       505       443       210        46         1         0
##      ANN_GDD50        E         N dist.lakes dist.coast      mrvbf
## 3087      1945 84568.04  2726.668  153690.43   201440.5 1.97863948
## 3070      1731 68938.15 89269.546   89837.08   283505.4 0.04802479
##          tri3    pop15  pop2pt5
## 3087 53.52679 1.370249 1.478395
## 3070 47.67730 1.558281 1.491255

ne.df[ix,c("LATITUDE_D", "LONGITUDE_")]

##      LATITUDE_D LONGITUDE_
## 3087      42.52     -74.97
## 3070      43.30     -75.15
```

The under-prediction (station 3087) is approximately six miles SW of Maryland, NY in Otsego County. This appears to be a mistake in the database; the elevation is listed as 3911' (feet), which is much higher than any terrain in the area.

> **Note:** If you wish, find the approximate location on Google Earth by entering the geographic coördinates; note these are on NAD27 so will not match the WGS84 of Google Earth exactly. You can also find this on Google Maps and display the terrain.

Note that 3911' = 1192 m; the strong suspicion is that this height which should be in feet was assumed to be in meters and then multiplied by (3.28084 feet m^{-2}), when in fact it was in feet. So, the incorrectly high elevation lead to an incorrectly low predicted value. After some research we find[9] revised records for this station[10]:

```
                                    BEGINS   ENDS      LATITUDE  LONGITUDE
NUM    DIV ST COUNTY COOP STATION   YEARMODY YEARMODY   D  M  S    D  M  S  ELEV
305113 02 NY OTSEGO MARYLAND 6 SW 19831026 19881001  42 31 00 -074 58 00  1192
305113 02 NY OTSEGO MARYLAND 6 SW 19881001 20080702  42 31 00 -074 58 00  1192
```

Here the elevation is given as 1192', which matches well with the terrain; 6 miles SW from the village of Maryland is along Schenvus Creek , whose

[9] https://wrcc.dri.edu/Monitoring/Stations/station_inventory_show.php?snet=coop&sstate=NY

[10] In fact this station was moved in July 2008, after our time period 1971-2000, so there is another record for it at a slightly different location

bluffs are at about 1200'. Further this matches the assumed mistake. So we feel justified in correcting this record accordingly.

The over-prediction is two miles SW of Hinckley, NY in Oneida County, near Barneveld. Again this appears to be a mistake, the elevation listed as 114' but is about 1200'; so the model predicts as if it is at a much lower elevation, i.e., the prediction is too high. It seems here there was just a missing digit; the database record is:

```
                                  BEGINS   ENDS      LATITUDE  LONGITUDE
NUM    DIV ST COUNTY COOP STATION YEARMODY YEARMODY   D  M  S    D  M  S  ELEV
303889 06 NY ONEIDA HINCKLEY 2 SW 19871014 19930301  43 18 00 -075 09 00  1141
```

We can correct these two points with their elevations from this (it appears correct) database. We make sure to make a record of these changes and our reasons for making them, in case other analysts want to check our work.

Task 14 : Correct the elevation attribute of these two points in the dataset. •

We correct the `ELEVATION_` field, which we are using for modelling. But we see that the dataset also has a `ELEV_FT` field, which duplicates this information – it is unclear why. At any rate, we do not want an inconsistent dataset, so we correct both elevation fields to the same value.

```
ne.m[ix[1],"ELEVATION_"] <- ne.m[ix[1],"ELEV_FT"] <- 1192
ne.df[ix[1],"ELEVATION_"] <- ne.df[ix[1],"ELEV_FT"] <- 1192
ne.m[ix[2],"ELEVATION_"] <- ne.m[ix[2],"ELEV_FT"] <- 1141
ne.df[ix[2],"ELEVATION_"] <- ne.df[ix[2],"ELEV_FT"] <- 1141
```

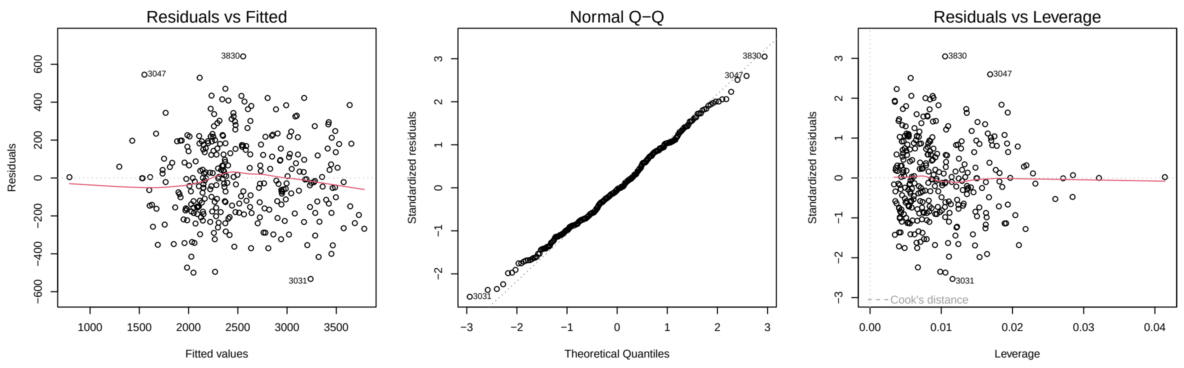
Task 15 : Re-fit the linear model from the corrected database. •

```
m.ols <- lm(ANN_GDD50 ~ sqrt(ELEVATION_) + N, data=ne.df)
summary(m.ols)

##
## Call:
## lm(formula = ANN_GDD50 ~ sqrt(ELEVATION_) + N, data = ne.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -532.76 -153.15   -7.84  155.44  641.76
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      3.151e+03  3.321e+01   94.87   <2e-16 ***
## sqrt(ELEVATION_) -3.040e+01  1.113e+00  -27.33   <2e-16 ***
## N               -1.952e-03  7.730e-05  -25.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 211.6 on 302 degrees of freedom
## Multiple R-squared:  0.865,Adjusted R-squared:  0.8641
## F-statistic: 967.3 on 2 and 302 DF,  p-value: < 2.2e-16
```

The model coefficients have changed and the R^2 has increased about 1.5%, as a result of correcting just these two (out of a total 305) records.

What about the regression diagnostics?

```r
par(mfrow=c(1,3))
plot(m.ols, which=c(1,2,5))
par(mfrow=c(1,1))
```



These are much better. Notice how the points with high absolute residuals now have much lower leverage, and the normal Q-Q plot no longer has points well off the expected 1:1 line.

Redo the actual vs. fitted plot:

```r
plot(ne.m$ANN_GDD50 ~ fitted(m.ols),
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by OLS (corrected data)",
     ylab="Actual",
     main="Annual GDD50")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid()
abline(0,1)
```

Task 16 : Summarize the OLS linear model residuals. •

```r
summary(residuals(m.ols))

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -532.764 -153.151   -7.838    0.000  155.435  641.758
```

To put these residuals in perspective, the difference between an early and medium maturity maize variety is about 300 GDD50. The inter-quartile range (IQR) is about ±150, which is about half of this on each side of perfect fit; i.e., the IQR is about the difference between maize maturities..

There are still some high residuals that could be checked for accuracy.

Challenge: Check the records for the largest residuals in the corrected model; if there is good evidence to do so, adjust the database accordingly and re-fit the model.

Challenge: Add Easting to the additive linear model and see if the model coefficient is significantly different from zero. You can try a pure trend surface (without elevation), a trend surface with elevation, and a trend surface with an elevation interaction – i.e., the effect of elevation is not the same across the region. Interpret the "best" model. Does this have a physical interpre-

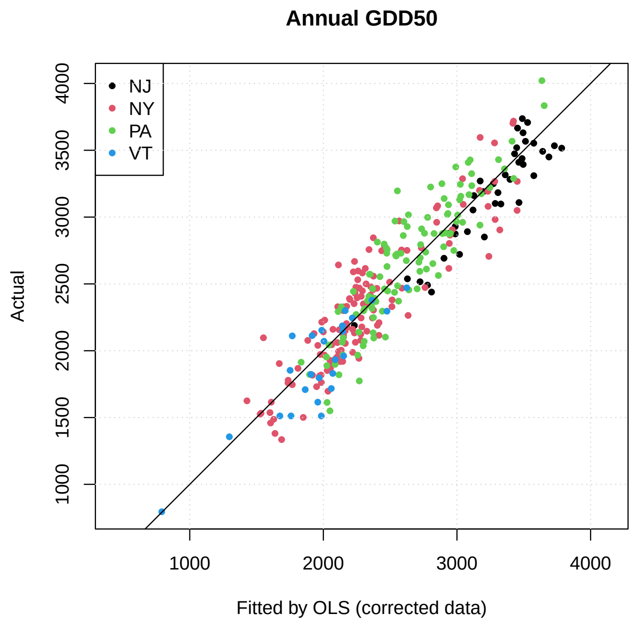


Figure 8: Actual vs. predicted, OLS model with corrected observations

tation, in the same way we can relate GDD50 to Northing and elevation? Compare with ANOVA and/or AIC; confirm good linear model diagnostics. Identify the stations with the largest positive and negative residuals; try to explain why.

4.4 Spatial correlation of OLS model residuals

Are the model residuals spatially correlated? If so, that violates the assumption of independent residuals that is necessary for the OLS fit to be optimum.

Task 17 : Add the linear model residuals to the `sf` object, so we can display them spatially. •

```
ne.m$ols.resid <- residuals(m.ols)
```

A so-called "bubble" plot shows the absolute value of a variable by circle size (the "bubble"), and the sign (±) by colour.

Task 18 : Display a bubble plot of the model residuals. •

There is no built-in bubble plot function for `sf` point geometries, so we build a small **user-defined function** for this. It will also be used later in the tutorial.

user-defined function

Note: This code uses the ability of R to build a command string using the `paste` function, parse it into R internal format with the `parse` functions, and then evaluate it in the current environment with the `eval` function.

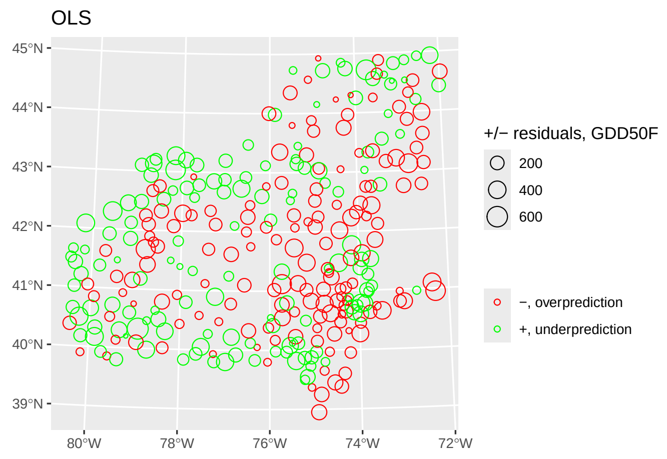
Arguments:

.oint.obj.name :	Name of the point object, not the object itself
.field.name :	Name of the data column (“field”) in the point object to be plotted
.field.label :	A text label for the field
.title :	Plot title, default none.

```
bubble.sf <- function(.point.obj.name, .field.name, .field.label, .title="") {
    # make a plus/minus indicator
    eval(parse(
        text = paste0("pm <- factor(",
                      .point.obj.name, "$",
                      .field.name, "> 0)")
    ))
    # rename them
    levels(pm) <- c("-, overprediction", "+, underprediction")
    # plot
    eval(parse(
      text = paste0("ggplot(",
                    .point.obj.name,
                    ") + geom_sf(aes(colour = pm, size = abs(",
                    .field.name,
                    ")), shape = 1) + labs(size = paste('+/-', .field.label),
                    colour = '', title = '",
                    .title, "') +
                    scale_colour_manual(values = c('red', 'green'))")
    ))
}
```

Now use this function on the OLS residuals:

```
bubble.sf("ne.m", "ols.resid", "residuals, GDD50F", "OLS")
```



Q7 : *Is there evidence for spatial correlation of the OLS residuals? Describe some locations where this is most evident. If you know the study area, speculate as to why certain areas are over- or under-predicted by the linear* *trend surface.* *Jump to A7* •

We now quantify the spatial correlation with variogram analysis.

4.4.1 The empirical variogram

The empirical variogram shows the average **semivariance** as a function of **separation** between point pairs.

The semivariance between any two observations is defined as:

$$\gamma(\mathrm{s}_i, \mathrm{s}_j) = \frac{1}{2}[z(\mathrm{s}_i) - z(\mathrm{s}_j)]^2 \tag{3}$$

where s is a geographic point and $z(\mathrm{s})$ is its **attribute value**, in this case the residual `ANN_GDD50` from the OLS model. Each pair of points is separated by a vector h, generally computed as the **Euclidean distance** between the points:

$$\mathrm{h} = ||\mathrm{x}_i, \mathrm{x}_j|| = (\sum_{k=1}^{n}(s_{i,k} - s_{j,k})^2)^{\frac{1}{2}} \tag{4}$$

where n is the number of dimensions (in this example, 2). There are $(n \cdot (n-1))/2$ point-pairs that can be compared this way; in our example this is $(305 \cdot 304)/2 = 46\ 360$; clearly we need some way to summarize this.

The model of spatial dependence assumes **2nd-order stationarity**, i.e., the semivariance does not depend on the absolute location. Therefore an empirical variogram **averages** the individual semivariances in some **separation range** called a "bin":

$$\gamma(\mathrm{h}) = \frac{1}{2N_\mathrm{h}}\sum_{i=1}^{N_\mathrm{h}}[z(\mathrm{s}_i) - z(\mathrm{s}_i + \mathrm{h})]^2 \tag{5}$$

where h is a lag vector, i.e., a range of separations.

The analyst chooses the bin widths: wide enough to have enough point-pairs ($> \approx 150$) for reliable estimation, narrow enough to reveal the fine structure of spatial dependence.

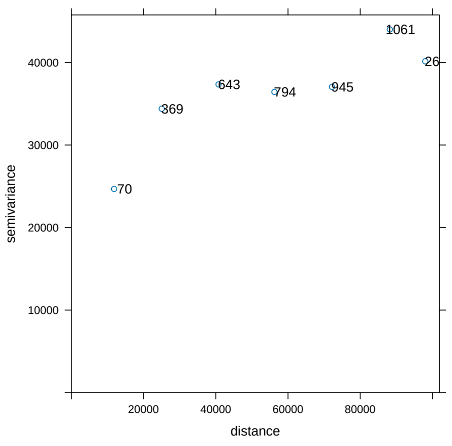
Task 19 : Compute and display a variogram of the OLS residuals. Use a short cutoff, estimated from the bubble plot. •

The variogram can be computed with the `variogram` function of the `gstat` package. Load this package:

```r
require(gstat)
```

Then compute and display the variogram. We use a cutoff of 100 km and a bin size of 16 km, to have enough points in the closest bin, and to avoid very local effects.

```
v.r.ols <- variogram(ols.resid ~ 1, locations=ne.m,
                     cutoff=100000, width=16000)
plot(v.r.ols, pl=T)
```



Note that the units of measure of the semivariances are (ANN_GDD50)^2.

Q8 : *What is the range of the spatial correlation? That is, the maximum separation distance at which there is lower semivariance than the maximum (total sill)?* *Jump to A8* •

4.4.2 Fitting an authorized variogram model

We require a continuous function of semivariance vs. separation, so that for any separation we can compute the semivariance to be used in the kriging equations. This must be an **authorized** variogram model that ensures that the kriging system will be postive semi-definite and thus invertible.

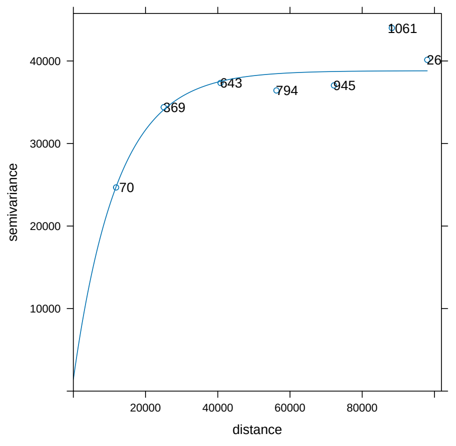
Task 20 : Fit an exponential variogram model to this empirical variogram. •

The `fit.variogram` function of the `gstat` package uses a weighted least-squares (WLS) fit of a model form to the empirical variogram. The default weighting is proportional to the number of point-pairs per bin, and inverse-square proportional to the separation. So more weight is given to the short-range component of the model, and to bins with more reliable estimates of the semivariance.

An exponential variogram model is often used with residuals, as it is the simplest model form and corresponds to a diffusion process, which seems to fit our concept of air movement.

```
(vmf.r.ols <- fit.variogram(v.r.ols, vgm(15000, "Exp", 20000, 20000)))

##   model     psill    range
## 1   Nug  1435.842     0.00
## 2   Exp 37382.603 12106.96
```

```
plot(v.r.ols, pl=T, model=vmf.r.ols)
```



The effective range of the fitted model is 36.3 km, that is, pairs of points within this range partially duplicate each other's information.

This successful model fit shows that *there is spatial correlation among the residuals*, so the OLS fit of §4 is *not optimal*.

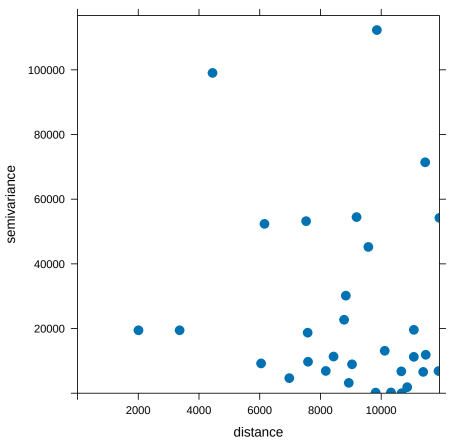
4.5 * Close-range anomalies

An interesting detour is to investigate anomalies which may show some features of micro-climate, using the **variogram cloud**. The results of this *optional* section are not used later in the exercise.

Task 21 : Examine the short-range behaviour with the variogram cloud (all point-pairs): •

The `cloud` argument to the `variogram` function produces a variogram *cloud*, i.e., a variogram that shows each point-pair's semivariance vs. separation, rather than summarizing in (somewhat arbitrary) bins. This allow us to identify pairs of points with unusually high or low semivariances for their separation.

```
vc <- variogram(ols.resid ~ 1, locations=ne.m, cutoff=12000, cloud=T)
plot(vc, pch=20, cex=2)
```



We see some close point-pairs with fairly large semivariances (differences in their model residuals), especially the point-pair separated by about 4 000 m but with a semivariance of about 100 000 GDD50^2. Let's see which point-pairs are closest, and their individual semivariances.

The `as.data.frame` method applied to a variogram cloud creates two new fields, `left` and `right`, giving the identities of the two points in each pair.

> **Note:** The order of the points is arbitrary, since the distance between them does not depend on which point is the origin and which the destination.

We then look for the highest semivariances at shortest distances ($<$ 8 km).

We use the `order` function is used to sort the indices, here in order of increasing separation.

```
vc.df <- as.data.frame(vc)
vc.close <- subset(vc.df, vc.df$dist < 8000)
## sort by separation, look for anomalies.
vc.close[order(vc.close$dist),c("dist","gamma","left","right")]

##        dist     gamma left right
## 26 2007.455 19456.999  234   136
## 20 3362.049 19461.756  143    77
## 11 4446.408 99043.801  107   106
## 31 6044.983  9207.781  289   283
## 28 6156.757 52361.745  250   197
## 3  6973.391  4663.255   21    11
## 13 7527.628 53205.364  123    19
## 9  7578.549 18717.268   39    33
## 14 7590.756  9725.831  125   123
```

This list shows all the point-pairs separated by $<$ 8km, see field `dist`. They have semivariances ranging from about 9000 to 55000 GDD50^2, except for the pair of points 106 and 107, which have a very large semivariance, 9.9044×10^4. This shows that the two have quite different residuals from the linear model fit These two are separated by only about 4.5 km but are quite dissimilar. This is quite an anomaly, let's see the details for this two stations:

```
print(ne.m[c(106,107),c("STATE","STATION_NA","LATITUDE_D","LONGITUDE_",
                         "ELEVATION_", "ANN_GDD50","ols.resid")])

## Simple feature collection with 2 features and 7 fields
```

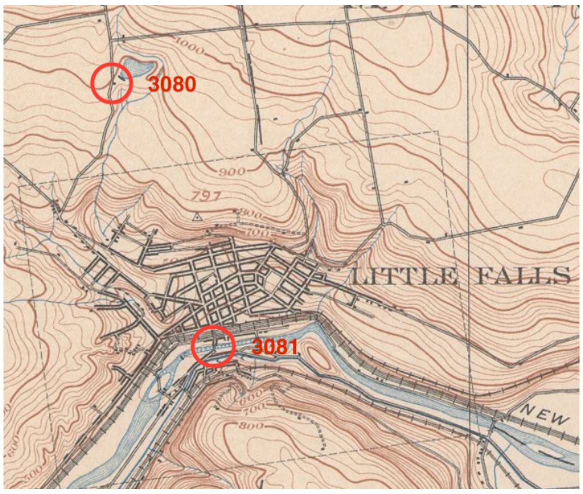


Figure 9: Little Falls NY weather station locations. Source: USGS 15 Minute Series, Little Falls NY quadrangle, 1900. Available from http://nationalmap.gov/historical/

```
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 91979.84 ymin: 59519.73 xmax: 92037.89 ymax: 63965.75
## Projected CRS: +proj=aea +lat_0=42.5 +lat_1=39
##          +lat_2=44 +lon_0=-76 +ellps=WGS84 +units=m
##      STATE            STATION_NA LATITUDE_D LONGITUDE_ ELEVATION_
## 3080    NY LITTLE FALLS CITY RSVR      43.07     -74.87        900
## 3081    NY   LITTLE FALLS MILL ST      43.03     -74.87        360
##      ANN_GDD50 ols.resid                  geometry
## 3080      1994 -119.8751 POINT (91979.84 63965.75)
## 3081      2783  325.1952 POINT (92037.89 59519.73)
```

These two stations are both near Little Falls (NY), one (3081) on Mill Street along the Mohawk River and one (3080) on Reservoir Road north of the village; see Figure 9.

Checking the topographic map, the elevations are correct; the discrepancy is because the Mill Street station is in a narrow river valley that is much warmer than predicted by the model, hence the large positive residual (actual - predicted). The Reservoir Road station is somewhat over-predicted.

So, the data seems to be correct; the lesson is that close-by stations can have quite different micro-climates, and we have no factor in the model to account for this. Local interpolators such as kriging will also fail in this situation.

5 Trend surface: a linear model fit by Generalized Least Squares

To solve the problem of spatially-correlated OLS residuals, we turn to generalized least squares (GLS).

5.1 * GLS – theory

The key difference here is that in the linear model fit by OLS, the residuals ε are assumed to be *independently* and *identically* distributed with the same variance σ^2:

$$y = X\beta + \varepsilon, \ \varepsilon \sim \mathcal{N}(0, \sigma^2 I) \tag{6}$$

Whereas, now the residuals are themselves considered to be a random variable η that has a covariance structure:

$$y = X\beta + \eta, \ \eta \sim \mathcal{N}(0, V) \tag{7}$$

where V is a positive-definite variance-covariance matrix of the model residuals. The covariances in this matrix (off-diagonals) are typically based on the distance between observations, using some model of spatial correlation.

Lark and Cullis [13, Appendix] point out that the error vectors can now not be assumed to be spherically distributed in feature space around the 0 expected value, but rather that error vectors in some directions are longer than in others. So, the measure of distance (the vector norm) is now a so-called "generalized" distance[11], taking into account the covariance between error vectors:

$$S = (y - X\beta)^T V^{-1} (y - X\beta) \tag{8}$$

The OLS equivalent is simpler:

$$S = (y - X\beta)^T (y - X\beta) \tag{9}$$

Comparing these equations, we see that the GLS formulation of Equation 8 includes the variance-covariance matrix of the residuals $V = \sigma^2 C$, where σ^2 is the variance of the residuals and C is the correlation matrix. This reduces to the OLS formulation of Equation 9 if there is no covariance, i.e., V = I.

Expanding Equation 8, taking the partial derivative with respect to the parameters, setting equal to zero and solving we obtain:

$$\begin{aligned}
\frac{\partial}{\partial\beta} S &= -2X^T V^{-1} y + 2X^T V^{-1} X\beta \\
0 &= -X^T V^{-1} y + X^T V^{-1} X\beta \\
\hat{\beta}_{\text{GLS}} &= (X^T V^{-1} X)^{-1} X^T V^{-1} y
\end{aligned} \tag{10}$$

This reduces to the OLS estimate $\hat{\beta}_{\text{OLS}}$ if there is no covariance, i.e., V = I.

In the case of spatial correlation, we ensure positive-definiteness (i.e., always a real-valued solution) by using an **authorized covariance function** C and assuming that the entries are completely determined by the vector **distance** between points $x_i - x_j$:

$$C_{i,j} = C(x_i - x_j) \tag{11}$$

In this formulation C has a three-parameter vector θ, as does the corresponding variogram model: the range a, the total sill σ^2, and the proportion of total sill due to pure error, not spatial correlation s[12].

[11] This is closely related to the Mahalanobis distance

[12] In variogram terms, this is the nugget variance c_0 as a proportion of the total sill (c_0+c_1).

In modelling terminology, the coefficients β are called **fixed** effects, because their effect on the response variable is fixed once the parameters are known. By contrast the covariance parameters η are called **random** effects, because their effect on the response variable is stochastic, depending on a random variable with these parameters.

Models with the form of Equation 7 are called **mixed** models: some effects are fixed (here, the relation between elevation or Northing and the GDD50) and others are random (here, the error variances) but follow a known structure; these models have many applications and are extensively discussed in Pinheiro and Bates [19]. Here the random effect η represents both the spatial structure of the residuals from the fixed-effects model, and the unexplainable (short-range) noise. This latter corresponds to the noise σ^2 of the linear model of Equation 6.

To solve Equation 10 we first need to compute V, i.e., estimate the variance parameters $\theta = [\sigma^2, s, a]$, use these to compute C with equation 11 and from this V, after which we can use equation 10 to estimate the fixed effects β. But θ is estimated from the residuals of the fixed-effects regression, which has not yet been computed. How can this "chicken-and-egg"[13] computation be solved?

The answer is to use **residual** (sometimes called "restricted") **maximum likelihood** (REML) to maximize the likelihood of the random effects θ independently of the fixed effects β.

Here we fit the fixed effects (regression coefficients) at the same time as we estimate the spatial correlation.

Lark and Cullis [13, Eq. 12] show that the likelihood of the parameters in Equation 6 can be expanded to include the spatial dependence implicit in the variance-covariance matrix V, rather than a single residual variance σ^2. The log-likelihood is then:

$$\ell(\beta, \theta|\mathrm{y}) = c - \frac{1}{2}\log|\mathrm{V}| - \frac{1}{2}(\mathrm{y} - \mathrm{X}\beta)^T \mathrm{V}^{-1}(\mathrm{y} - \mathrm{X}\beta) \tag{12}$$

where c is a constant (and so does not vary with the parameters) and V is built from the variance parameters θ and the distances between the observations. By assuming **second-order stationarity**[14], the structure can be summarized by the covariance parameters $\theta = [\sigma^2, s, a]$, i.e., the total sill, nugget proportion, and range.

However, maximizing this likelihood for the random-effects covariance parameters θ also requires maximizing in terms of the fixed-effects regression parameters β, which in this context are called *nuisance parameters* since at this point we don't care about their values; we will compute them after determining the covariance structure.

Both the covariance and the nuisance parameters β must be estimated, it seems at the same time ("chicken and egg" problem) but in fact the technique

[13] from the question "which came first, the chicken or the egg?"

[14] that is, the covariance structure is the same over the entire field, and only depends on the distance between pairs of points

of REML can be used to first estimate θ without having to know the nuisance parameters. Then we can use these to compute C with equation 11 and from this V, after which we can use equation 10 to estimate the fixed effects β.

The maximum likelihood estimate of θ is thus called “restricted”, because it only estimates the covariance parameters (random effects). Conceptually, REML estimation of the covariance parameters θ is ML estimation of both these and the nuisance parameters β, with the later integrated out [19, §2.2.5]:

$$\ell(\theta|\mathrm{y}) = \int \ell(\beta, \theta|\mathrm{y})\; d\beta \tag{13}$$

Pinheiro and Bates [19, §2.2.5] show how this is achieved, given a likelihood function, by a change of variable to a statistic sufficient for β.

5.2 GLS – practice

The computations are performed with the `gls` function of the `nlme` ‘Nonlinear mixed effects models’ package [1].

Task 22 : Set up and solve a GLS model, using the covariance structure estimated from the variogram of the OLS residuals. •

The linear model formulation is the same as for `lm`. However:

- It has an additional argument `correlation`, which specifies the correlation structure.
- This is built with various correlation models; we use `corExp` for exponential spatial correlation, which is what we fit for the OLS residuals,
  - The `form` names the dimensions, here 2D with the Easting and Northing.
  - We initialize the search for the correlation structure parameters with the `value` argument, a list of the initial values. Here we only specify the range.
  - Our fitted variogram model for the OLS residuals showed a zero nugget, so here the function should not fit a nugget. So we set the `nugget` argument to `FALSE`.[15]

Note: For a list of the predefined model forms see `?corClasses`. Users can also define their own `corStruct` classes.

```
require(nlme)
vmf.r.ols[1:2,]

##   model     psill    range
## 1   Nug  1435.842     0.00
## 2   Exp 37382.603 12106.96

m.gls <- gls(model=ANN_GDD50 ~ sqrt(ELEVATION_) + N,
             data=ne.df,
             correlation=corExp(
```

[15] The nugget could also be fixed at a user-specified value.

```
                  value=c(vmf.r.ols[2,"range"]),
                  form=~E + N,
                  nugget=FALSE))
```

The `gls` function is *not* guaranteed to find a valid correlation structure. First, there may be no spatial correlation of the residuals. Second, we may have specified an inappropriate model form. Third, if the starting values are not close to good fits, the optimization method may not find the correct fit. Therefore it is crucial to check the results of the model fitting to see if they are reasonable.

Task 23 : Display the model summary. •

This shows both the linear model coefficients and the spatial correlation structure.

```
summary(m.gls)

## Generalized least squares fit by REML
##   Model: ANN_GDD50 ~ sqrt(ELEVATION_) + N
##   Data: ne.df
##        AIC      BIC    logLik
##   4120.113 4138.665 -2055.056
##
## Correlation Structure: Exponential spatial correlation
##  Formula: ~E + N
##  Parameter estimate(s):
##    range
## 17456.99
##
## Coefficients:
##                      Value Std.Error   t-value p-value
## (Intercept)      3136.3707  44.03754  71.22039       0
## sqrt(ELEVATION_)  -30.0448   1.41085 -21.29546       0
## N                  -0.0019   0.00011 -16.64114       0
##
##  Correlation:
##                  (Intr) s(ELEV
## sqrt(ELEVATION_) -0.888
## N                 0.319 -0.183
##
## Standardized residuals:
##         Min          Q1         Med          Q3         Max
## -2.36510335 -0.68842210 -0.01831242  0.74010041  3.00789760
##
## Residual standard error: 217.344
## Degrees of freedom: 305 total; 302 residual
```

Task 24 : Display the confidence intervals for the coefficients. •

The `intervals` function of the `nlme` gives approximate confidence intervals of the GLS fit.

```
intervals(m.gls, level=0.95)$coef

##                          lower          est.         upper
## (Intercept)       3.049711e+03  3.136371e+03  3.223030e+03
## sqrt(ELEVATION_) -3.282113e+01 -3.004478e+01 -2.726843e+01
## N                -2.135757e-03 -1.909907e-03 -1.684056e-03
## attr(,"label")
## [1] "Coefficients:"
```

These intervals seem fairly wide, indicating that the model is perhaps not sufficiently specified to capture all the reasons for variation in GDD50 over this area..

Task 25 : Display the correlation structure fit by `gls`. •

```
intervals(m.gls, level=0.95)$corStruct

##          lower     est.   upper
## range 12850.43 17456.99 23714.9
## attr(,"label")
## [1] "Correlation structure:"
```

Q9 : *What is the 95% confidence interval of the range parameter? Does this seem narrow or wide?* *Jump to A9* •

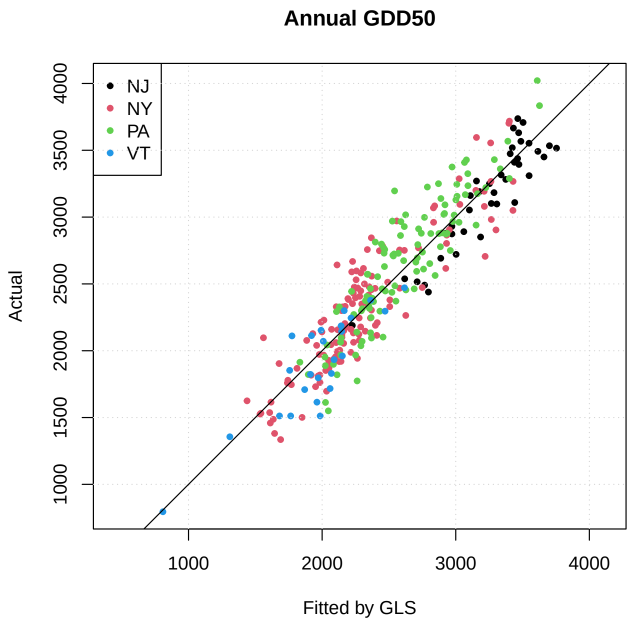
Task 26 : Compare with the correlation structure estimated from the OLS residuals. •

```
intervals(m.gls, level=0.95)$corStruct

##          lower     est.   upper
## range 12850.43 17456.99 23714.9
## attr(,"label")
## [1] "Correlation structure:"

vmf.r.ols[2,"range"]

## [1] 12106.96
```

Q10 : *How closely does the correlation structure fitted by GLS match that estimated from the variogram of the OLS residuals?* *Jump to A10* •

Task 27 : Plot the actual vs. model fits on a 1:1 scatterplot. •

```
plot(ne.m$ANN_GDD50 ~ predict(m.gls),
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by GLS",
     ylab="Actual",
     main="Annual GDD50")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid()
abline(0,1)
```



The fit clusters well around the 1:1 line (good accuracy) but is diffuse (low precision).

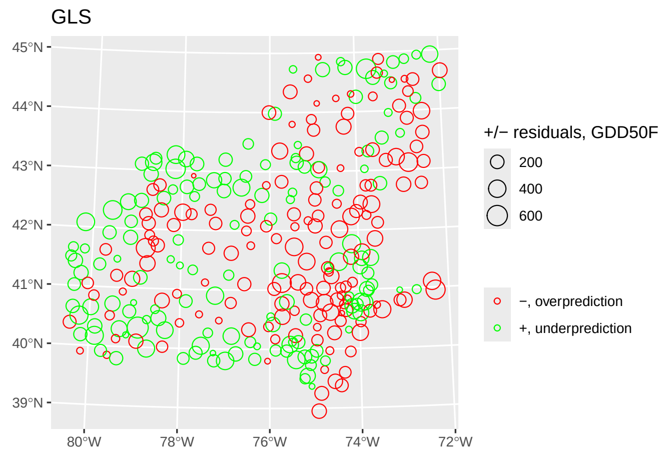
5.3 Spatial correlation of GLS model residuals

As with the OLS model (§4.4), the GLS residuals may show spatial correlation. We examine this with an empirical variogram and then fit a variogram model.

Task 28 : Display a bubble plot of the model residuals. •

We use the function defined in the OLS analysis, but now with a different field.

```
ne.m$gls.resid <- residuals(m.gls)
summary(ne.m$gls.resid)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -514.041 -149.624   -3.980    8.496  160.856  653.749

bubble.sf("ne.m", "gls.resid", "residuals, GDD50F", "GLS")
```



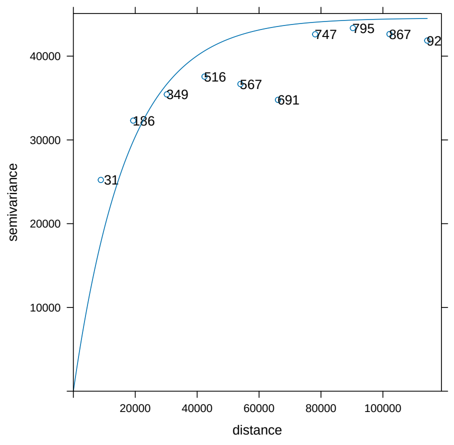
Q11 : *Does GLS remove the spatial correlation? Describe the spatial correlation structure of the GLS residuals.* *Jump to A11* •

These residuals should show the spatial correlation discovered in the REML fit.

Task 29 : Display a variogram of the GLS residuals, with the variogram model estimated as part of the GLS fit. •

This fit gives the range; we estimate the total sill from the overall variance. Knowing these variogram parameters, we construct a variogram model with the `vgm` function: We estimate the total sill as the variance of the residuals. Since there is no nugget variance in this model, we do not have to adjust it downward for the proportional nugget.

```
v.r.gls <- variogram(gls.resid ~ 1, ne.m, cutoff=120000, width=12000)
(vmf.r.gls <- vgm(psill=var(ne.m$gls.resid),
                      model="Exp",
                      range=intervals(m.gls)$corStruct["range","est."],
                      nugget=TRUE))

##   model    psill    range
## 1   Nug     1.00     0.00
## 2   Exp 44558.48 17456.99

plot(v.r.gls, pl=T, model=vmf.r.gls)
```



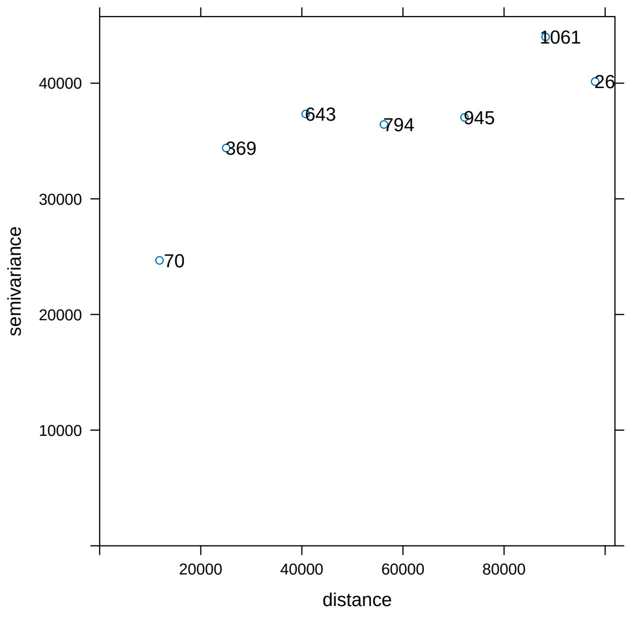
Q12 : *How well does this model fit the empirical variogram of the GLS residuals?* *Jump to A12* •

5.4 * Fitting a GLS correlation structure with a nugget variance

In the previous § we built a model of spatial dependence of the residuals using the `corExp` model for exponential spatial correlation. Because our analysis of the OLS residual variogram in §4.4, we did not fit a nugget, by specifying the `nugget` argument to `FALSE`. In other analyses there may well be a nugget effect to be fit, so here we show how to do that, and the effect it has on the correlation structure fitted by `gls`.

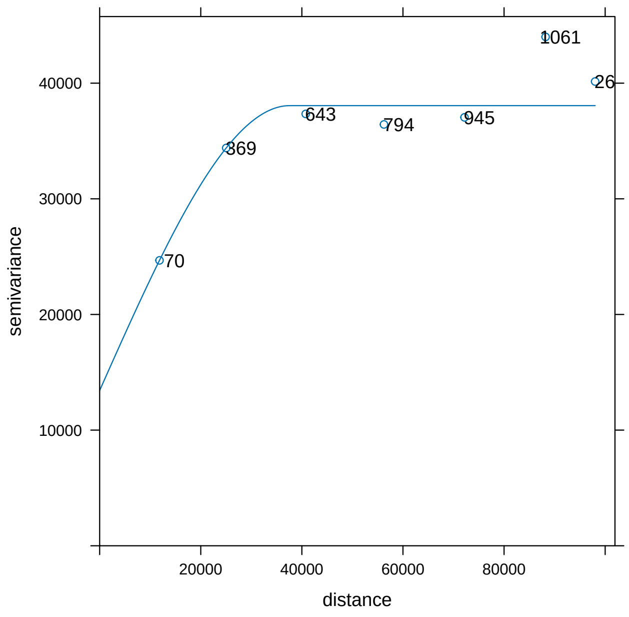
In this example an exponential model has almost no nugget, so we select a spherical model that is nearly linear near the origin. We estimate the starting parameters from the empirical variogram.

```
plot(v.r.ols, pl=T)
```



The total sill appears to be ≈ 35000 (ANN_GDD50)^2, the nugget ≈ 15000 (ANN_GDD50)^2, so, the partial sill is ≈ 20000 (ANN_GDD50)^2; the range (where the spherical model reaches the sill) ≈ 40000 m.

```
(vmf.r.ols.sph <- fit.variogram(v.r.ols,
                                vgm(psill=20000, model="Sph",
                                    range=40000, nugget=15000)))

##   model    psill   range
## 1   Nug 13426.14     0.0
## 2   Sph 24629.99 37585.4

plot(v.r.ols, pl=T, model=vmf.r.ols.sph)
```



This does not fit too badly, and clearly has a substantial nugget. Our initial estimates were close to the fitted values. This must be converted to a *proportional* nugget, i.e., the proportion of the total sill represented by the nugget.

```
(prop.nugget <- vmf.r.ols.sph[1,"psill"]/sum(vmf.r.ols.sph[,"psill"]))

## [1] 0.3527983
```

The nugget is about 35% of the total sill, with this model. We now use this in `gls`, substituting `corSpher` to specify a spherical model of spatial dependence.

To include a nugget variance, we must:

1. specify `nugget` argument as `TRUE`;
2. expand the `value` argument to a two-element list: (1) the starting value of the range, as before; and (2) the starting value of the nugget.

```
m.gls.2 <- gls(model=ANN_GDD50 ~ sqrt(ELEVATION_) + N,
             data=ne.df,
             correlation=corSpher(
                 value=c(vmf.r.ols.sph[2,"range"], prop.nugget),
                 form=~E + N,
                 nugget=TRUE))
```

We compare the fitted correlation structures:

```
intervals(m.gls, level=0.95)$corStruct

##          lower     est.   upper
## range 12850.43 17456.99 23714.9
## attr(,"label")
## [1] "Correlation structure:"

intervals(m.gls.2, level=0.95)$corStruct

##               lower         est.         upper
## range   2.271152e+05 3.014994e+05 4.004629e+05
## nugget 2.565108e-01 4.622866e-01 6.817665e-01
## attr(,"label")
## [1] "Correlation structure:"
```

The nugget proportion has been fit as 0.462, somewhat higher than our original estimate 0.353.

Recall that the range parameter of the exponential model is 1/3 of the effective range, so to compare them:

```
intervals(m.gls)$corStruct["range", 2]*3

## [1] 52370.98

intervals(m.gls.2)$corStruct["range", 2]

## [1] 301499.4
```

The effective range is much longer, about 300 km for the spherical model with nugget, compared to about 50 km for the exponential model with no nugget. This seems unrealistic when we compare it with the residual variogram. The spherical model is thus not appropriate; we showed it here in order to explain how to fit a proportional nugget, if the empirical residual variogram suggests that there is a nugget variance.

We can also compare the regression coefficients:

```
intervals(m.gls, level=0.95)$coef[,2]

##     (Intercept) sqrt(ELEVATION_)                N
##    3.136371e+03    -3.004478e+01    -1.909907e-03

intervals(m.gls.2, level=0.95)$coef[,2]

##     (Intercept) sqrt(ELEVATION_)                N
##    3.209772e+03    -3.241043e+01    -1.658746e-03
```

The changed spatial correlation structure has considerably changed the regression coefficients.

5.5 Comparing OLS and GLS models

How much has accounting for the spatial correlation of the model residuals affected the linear models?

Task 30 : Compare the coefficients of the GLS and OLS models. Compute the relative change. •

```
coefficients(m.gls)

##     (Intercept) sqrt(ELEVATION_)                N
##    3.136371e+03    -3.004478e+01    -1.909907e-03

coefficients(m.ols)

##     (Intercept) sqrt(ELEVATION_)                N
##    3.150881e+03    -3.040452e+01    -1.952152e-03

round(100*(coefficients(m.gls) - coefficients(m.ols))/coefficients(m.ols),2)

##     (Intercept) sqrt(ELEVATION_)                N
##           -0.46            -1.18            -2.16
```

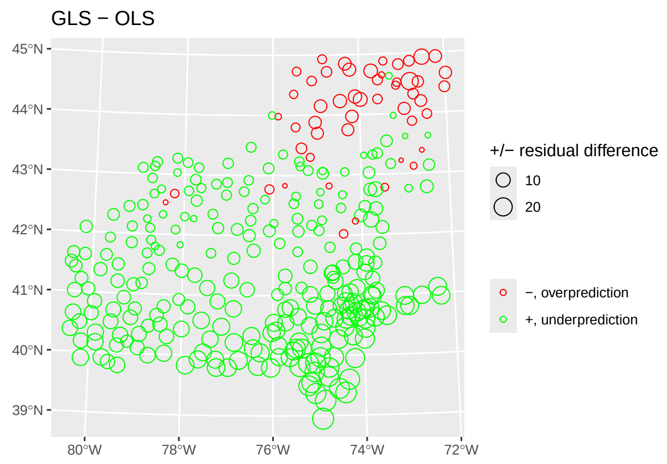
Q13 : *How much have the linear model coefficients changed from the OLS to the GLS fit? What explains the change in coefficients?* Jump to A13 •

We can see this spatially by comparing the residuals.

Task 31 : Compute the difference between the GLS and OLS residuals, add them to the spatial points, and display as a bubble plot. •

We use a different colour scheme to emphasize that this is the *difference* between residuals, *not* the residuals themselves.

```
ne.m$diff.gls.ols.resid <- (ne.m$gls.resid - ne.m$ols.resid)
summary(ne.m$diff.gls.ols.resid)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -17.825   2.088   7.275   8.496  15.631  29.544

bubble.sf("ne.m", "diff.gls.ols.resid", "residual difference", "GLS - OLS")
```



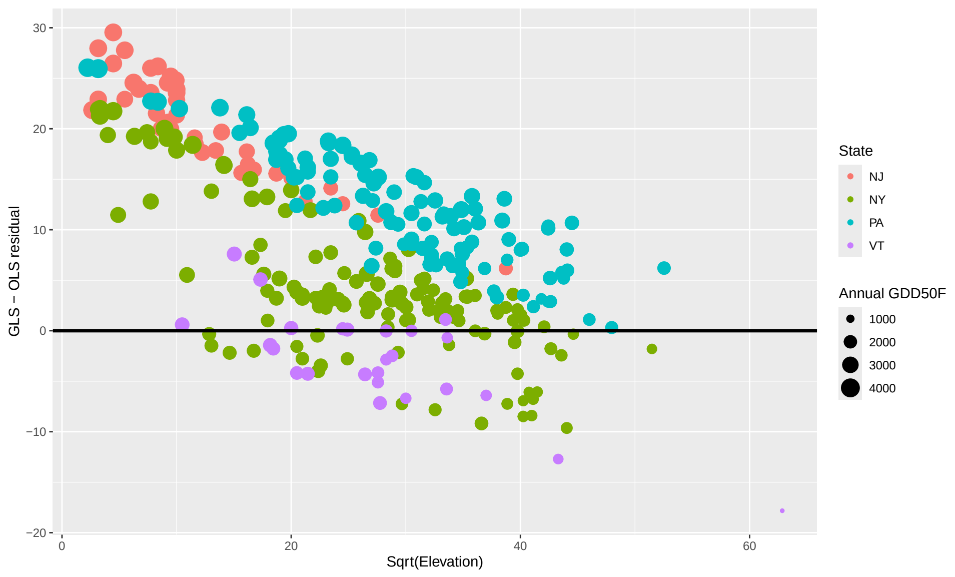
Q14 : *Where are the largest differences between the OLS and GLS residuals?* *Jump to A14*
•

The coefficient for elevation was reduced by a smaller amount, and it is for the square root. To visualize this effect, we can use a scatterplot of the change in residuals vs. this marginal predictor.

Task 32 : Display a scatterplot of change in residuals vs. the square root of elevation. •

We can make the plot more informative by colouring the points by state and making their size proportional to the target variable

```
ggplot() +
    geom_point(aes(x=sqrt(ne.m$ELEVATION_), y=ne.m$diff.gls.ols.resid,
                   size=ne.m$ANN_GDD50, colour=ne.m$STATE)) +
                       xlab("Sqrt(Elevation)") +
                           ylab("GLS - OLS residual") +
                               geom_hline(yintercept=0, linewidth=1.2) +
  labs(colour = "State", size = "Annual GDD50F")
```



This shows clearly that GLS residuals are larger at the lower elevations, and that the largest adjustments tend to be at the largest GDD50. Note that there would be a confounding effect if the two predictors (elevation and Northing) were not almost independent, as they are in this case.

So in combination, a southerly, low-lying station will have the largest positive GLS-OLS residuals, i.e., GLS predicts higher than OLS; a northerly, high-elevation station the largest negative residuals.

Task 33 : Display the station name, elevation, and coördinates of the most positive and negative residuals. •

```
ix <- which.max(ne.m$diff.gls.ols.resid)
ne.df[ix,c("STATION_NA","STATE","ELEVATION_","N","E")]

##         STATION_NA STATE ELEVATION_         N        E
## 2859 CAPE MAY 2 NW    NJ         20 -393922.9 92756.51

ix <- which.min(ne.m$diff.gls.ols.resid)
ne.df[ix,c("STATION_NA","STATE","ELEVATION_","N","E")]

##           STATION_NA STATE ELEVATION_        N        E
## 4716 MOUNT MANSFIELD    VT       3950 230248.4 252835.5
```

The largest negative residual (GLS reduced the prediction the most) is for Mt. Mansfield (VT), the highest elevation station in the dataset, and near the N limit. The largest positive residual (GLS increased the prediction the most) is for Cape May (NJ), just above sea level and the southernmost station.

Conclusion: accounting for spatial correlation in the residuals significantly changed the linear model, resulting in differences up to 30 GDD50.

6 Prediction over the regional grid by OLS and GLS

A natural question is how target variables vary over the entire study area, not just at the observation points – this is the interpolation (or extrapolation) problem. For regional studies we want to predict and visualize over the entire area; i.e., we want to produce a **map** of the target variable. For this we need the predictors used in the models at a set of **grid cells** covering the **whole study area**. Point predictions are then made at the centre of each grid cell.

These have been prepared in the companion tutorial "Regional mapping of climate variables from point samples: Data preparation" and loaded with the points dataset in 3, above.

Task 34 : Predict over the grid with the OLS and GLS models, add the results to the dataframe, and summarize them. •

```
dem.ne.m.df$pred.ols <- predict(m.ols, newdata=dem.ne.m.df)
dem.ne.m.df$pred.gls <- predict(m.gls, newdata=dem.ne.m.df)
summary(dem.ne.m.df[,-(1:3)])

##    dist.lakes       dist.coast           mrvbf
##  Min.   :     0   Min.   :     2.3   Min.   :0.000000
##  1st Qu.: 84666   1st Qu.:126409.8   1st Qu.:0.008421
##  Median :222394   Median :276842.2   Median :0.380940
##  Mean   :228099   Mean   :275295.6   Mean   :0.947745
##  3rd Qu.:346447   3rd Qu.:406957.9   3rd Qu.:1.787373
##  Max.   :648952   Max.   :708701.1   Max.   :3.991617
##       tri3            pop15            pop2pt5         pred.ols
##  Min.   :  0.00   Min.   :0.0000   Min.   :0.000   Min.   : 877
##  1st Qu.: 10.66   1st Qu.:0.9855   1st Qu.:0.231   1st Qu.:2004
##  Median : 29.11   Median :1.4808   Median :1.159   Median :2332
##  Mean   : 39.98   Mean   :1.4046   Mean   :1.114   Mean   :2474
##  3rd Qu.: 57.30   3rd Qu.:1.9091   3rd Qu.:1.648   3rd Qu.:2821
##  Max.   :371.70   Max.   :3.8818   Max.   :4.418   Max.   :3926
##     pred.gls
##  Min.   : 892.9
##  1st Qu.:2003.9
##  Median :2327.1
##  Mean   :2466.6
##  3rd Qu.:2806.5
##  Max.   :3894.9
```

Task 35 : Create a function to display prediction maps using `ggplot`. •

user-defined function

One purpose of a user-defined **function** is to make a consistent approach to a repetitive task. In this tutorial we will make many maps showing the predictions of a mapping method over the study area – here we already have two, OLS and GLS. For consistency we define a function, using the `function` function (!). The function is assigned to a workspace name, and has a set of **arguments** with names that are used within the function.

Here since we know we always want to show the known points on the map (to show the discrepancy between predicted and known at the points) we hard-code that part into the function, and do not make it an argument. Also, we know the map data source will be a field in the `dem.ne.m.df` object, so that is also hard-coded.

The default colour palette (§A) is `YlGnBu`, a "sequential" palette from a "low" to a "high" value. This can be changed by the caller to show other kinds of maps.

We include an option `.plot.limits`, default `NULL`, to specify the limits of the legend scale, to allow side-by-side comparison of maps.

Arguments:

.prediction :	name of the field in `dem.ne.m.df` to plot (character string)
.plot.title :	title for the map (character string)
.legend.title :	title for the legend (character string)
.plot.limits :	range of values for the legend (two-element numeric vector); default "NULL", i.e., taken from the data
.palette :	name of a colour palette (character string); default "YlGnBu"

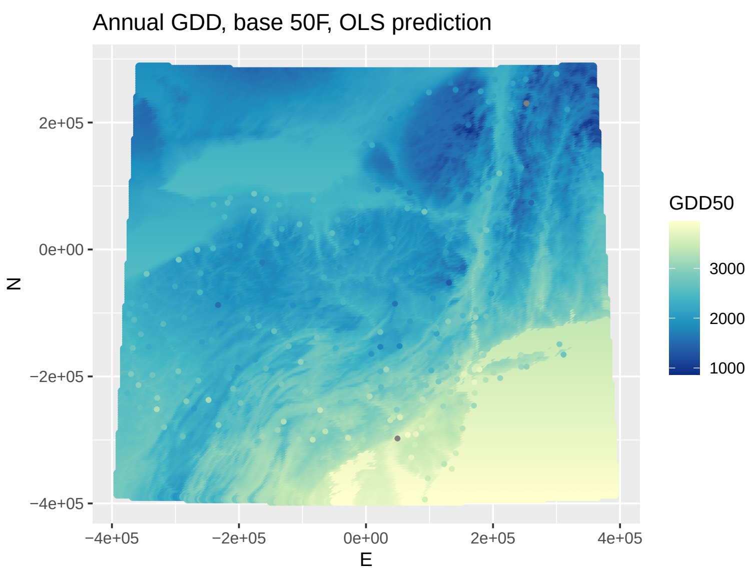
```
display.prediction.map <- function(.prediction,
                                   .plot.title,
                                   .legend.title,
                                   .plot.limits=NULL,
                                   .palette="YlGnBu") {
    ggplot(data=dem.ne.m.df) +
        # note must find the indirect name in the dataframe
        geom_point(aes(x=E, y=N,
                       color=.data[[as.name(.prediction)]])) +
        xlab("E") + ylab("N") + coord_fixed() +
        geom_point(aes(x=E, y=N, color=ANN_GDD50),
                   data=ne.df, shape=I(20)) +
        ggtitle(.plot.title) +
        scale_colour_distiller(name="GDD50",
                               space="Lab",
                               palette=.palette,
                               limits=.plot.limits)
}
```

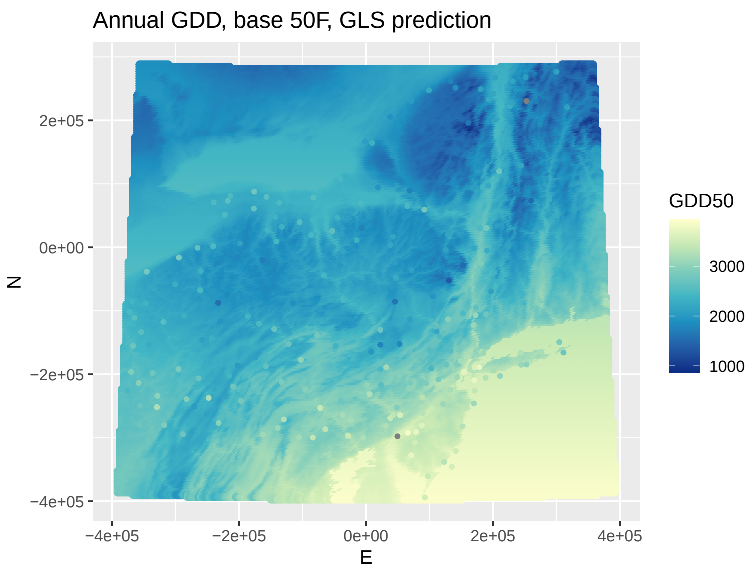
Task 36 : Plot these on the same visual scale, over the entire bounding box. •

The reason to visualize over the bounding box is to see the effect of extrapolation of a linear model beyond its calibration range, in this case Northing and elevation.

We now call the function for both predictions. The argument names used within the function are assigned to the names we give when calling the function. So the function operates on the map. Note that we use the same limits for the colour ramp, so the two maps can be directly compared.

```
ols.ix <- which(names(dem.ne.m.df)=="pred.ols")
gls.ix <- which(names(dem.ne.m.df)=="pred.gls")
# set up limits of the scale for the target variable
# use the extremes +/-10 GDD, to make sure that all values are shown
(gdd.pred.lim <- round(
    c(min(dem.ne.m.df[,c(ols.ix, gls.ix)])-10, max(dem.ne.m.df[,ols.ix, gls.ix])+10)))

## [1]  867 3936

display.prediction.map("pred.ols", "Annual GDD, base 50F, OLS prediction",
                       "GDD50", gdd.pred.lim)
display.prediction.map("pred.gls", "Annual GDD, base 50F, GLS prediction",
                       "GDD50", gdd.pred.lim)
```





Task 37 : Compute the differences between the OLS and GLS predictions, add them to the data frame, and display them. •

```
summary(dem.ne.m.df$diff.gls.ols <-
            dem.ne.m.df$pred.gls - dem.ne.m.df$pred.ols)
```

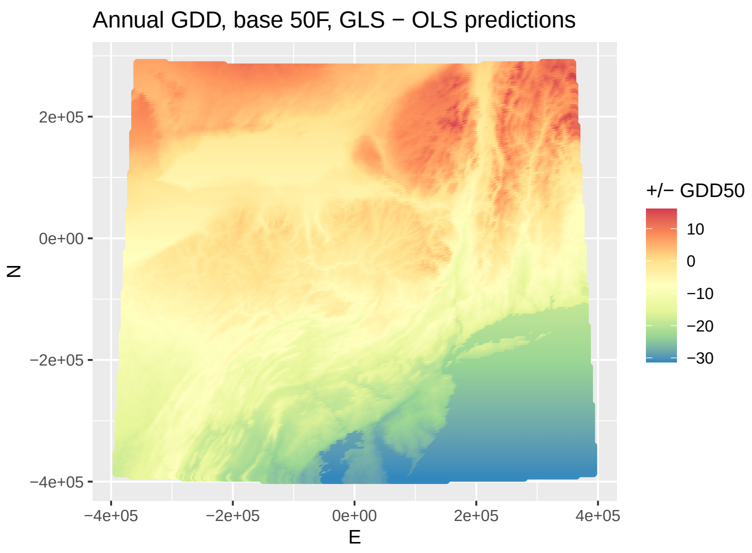
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -31.2889 -15.3346  -4.9698  -7.6704   0.2453  16.0963
```

The OLS and GLS predictions only vary slightly.

We also define a function to display difference maps. The default colour palette (§A) is `Spectral`, used for "diverging" palettes with a natural zero (so the two extremes are the most contrasting). This can be changed by the caller to show other kinds of maps.

```
display.difference.map <- function(.diff.map.name,
                                   .diff.map.title,
                                   .legend.name,
                                   .palette="Spectral") {
    ggplot(data=dem.ne.m.df) +
        geom_point(aes(x = .data[["E"]], y = .data[["N"]],
                       color=.data[[as.name(.diff.map.name)]])) +
        xlab("E") + ylab("N") + coord_fixed() +
        ggtitle(.diff.map.title) +
        scale_colour_distiller(name=.legend.name,
                               space="Lab",
                               palette=.palette)
    }
```

```
display.difference.map("diff.gls.ols",
                       "Annual GDD, base 50F, GLS - OLS predictions",
                       "+/- GDD50")
```



The GLS model predicts slightly higher to the north and at higher elevations.

7 Improving the trend prediction by Regression Kriging

In both GLS and OLS we have seen that the residuals are spatially-correlated. In §5.3 we fitted a variogram model to the GLS residuals:

```
print(vmf.r.gls)

##   model    psill    range
## 1   Nug     1.00     0.00
## 2   Exp 44558.48 17456.99
```

This exponential model has an effective spatial correlation range of about 52 km. So the residual at a prediction point is not simply the residual from the trend surface, but also depends on surrounding trend surface residuals, within this radius. Every location within the four-state area is within this effective range of one or more stations, and so the trend surface prediction can be locally adjusted in two steps:

1. krige the **residuals** to obtain the **local deviation** from the GLS trend surface. This deviation can be either "above" or "below" the surface;
2. **add** this to the trend surface prediction.

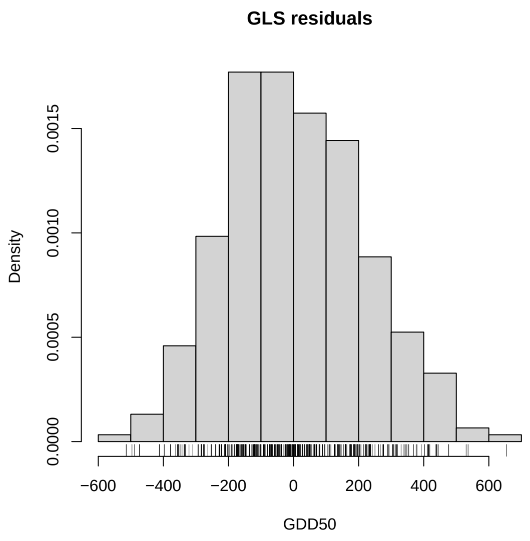
This procedure is called **Regression Kriging**[16] [8].

The "kriging" part of the "regression kriging" term only applies to the `residuals` from the trend, and thus uses a variogram model and correlation structure for these, not for the original values. Most of the spatial variation has been taken out by the trend surface (the "regression", $Z^*(\mathrm{s})$); this is the "regional" model. The kriging step is to adjust it locally (the "local spatial dependence" $\varepsilon(\mathrm{s})$), referring to Equation 1, the universal model of spatial distribution Note that there will still be unexplained variation ("pure noise", $\varepsilon'(\mathrm{s})$.

Task 38 : Summarize the GLS trend surface residuals with a histogram and numerical summary. •

```
summary(ne.m$gls.resid)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -514.041 -149.624   -3.980    8.496  160.856  653.749

hist(ne.m$gls.resid, freq=F,
     xlab="GDD50",
     main="GLS residuals")
rug(ne.m$gls.resid)
```

[16] 'Kriging' is named for the South African mining geostatistician Danie Krige (1919–2013), who developed the method in the 1950's for estimating gold reserves. His empirical method was formalized in the 1960's by Georges Matheron (1930–2000) working at the École de Mines in France. The theory had been previously developed in the 1930's by Andrey Kolmogorov (1903-1987) but was not practical until digital computers had been developed.



Notice that the mean residual is not zero. This is because GLS trades unbiasedness for precision of the trend coefficients.

Now that we have (1) the known points; (2) a fitted authorized variogram model, we can predict at any location. The following optional section explains the mathematics of the OK system.

7.1 * The Ordinary Kriging system

OK predicts at an unknown point as a **weighted linear average** from the n known points with value z_i to obtain the value at each unknown point z_0 (Eqn.14).

$$z_0 = \sum_{i=1}^{n} \lambda_i z_i \tag{14}$$

The weights λ_i must sum to 1 (i.e., estimation of the mean is unbiased). Many weighting schemes can satisfy this equation, for example nearest neighbour (Thiessen polygons) (§12.5), inverse-distance weighting (§12.4), average of points within some radius, or average of some number of neighbours. The unique aspect of OK is that these weights are selected to **minimize the prediction variance** – this is the reason OK is called a “Best Unbiased Linear Predictor” (BLUP).

Note: Always remember, this “best” is with reference to the fitted variogram model. And there is no way to objectively know if that model is correct. So whether OK is “best” in the real world is not proveable.

In OK the weights λ_i are determined from the **Ordinary Kriging System**, which is derived from an expression for the prediction variance, which is minimized to derive these equations.

Weights are found by solving:

$$\mathrm{A}\lambda = \mathrm{b} \tag{15}$$

where:

$$\mathrm{A} \;=\; \begin{bmatrix} \gamma(\mathrm{x}_1,\mathrm{x}_1) & \gamma(\mathrm{x}_1,\mathrm{x}_2) & \cdots & \gamma(\mathrm{x}_1,\mathrm{x}_N) & 1 \\ \gamma(\mathrm{x}_2,\mathrm{x}_1) & \gamma(\mathrm{x}_2,\mathrm{x}_2) & \cdots & \gamma(\mathrm{x}_2,\mathrm{x}_N) & 1 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \gamma(\mathrm{x}_N,\mathrm{x}_1) & \gamma(\mathrm{x}_N,\mathrm{x}_2) & \cdots & \gamma(\mathrm{x}_N,\mathrm{x}_N) & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix}$$

$$\lambda \;=\; \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \\ \psi \end{bmatrix} \qquad \mathrm{b} \;=\; \begin{bmatrix} \gamma(\mathrm{x}_1,\mathrm{x}_0) \\ \gamma(\mathrm{x}_2,\mathrm{x}_0) \\ \vdots \\ \gamma(\mathrm{x}_N,\mathrm{x}_0) \\ 1 \end{bmatrix}$$

In the A matrix the upper-left block $N \times N$ block is the **spatial correlation structure** of the observations; these are derived from the fitted variogram model. The last row ensures unbiasedness in estimating the spatial mean. The right-hand column is used to find the LaGrange multiplier that minimizes the variance. The kriging variance at a point is given by the scalar product of the weights (and multiplier) vector λ with the right-hand side of the kriging system.

$$\hat{\sigma}^2(\mathrm{x}_0) = \mathrm{b}^T\lambda \tag{16}$$

Note that λ includes as its last element ψ, which depends on covariance structure of the observation points. Note also that the prediction variance is computed only from semivariances, not from data values.

7.2 Predicting the residuals by Ordinary Kriging

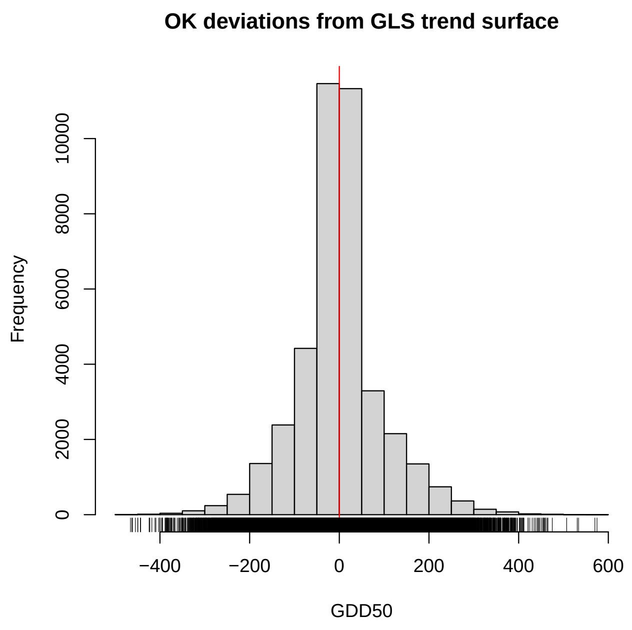
Task 39 : Predict the deviation from the trend surface at each location on the grid, using Ordinary Kriging (OK) of the GLS residuals, and display its summary. •

Note: We use OK instead of Simple Kriging (SK) because the *spatial* mean of the GLS residuals may not be zero. The non-spatial mean of the GLS mean is not required to be zero, as in OLS.

There are several R packages that implement kriging. Here we use the `krige` function of the `gstat` package, which uses the fitted variogram model as in `model` argument.

The points on which to krige are centres of the grid cells, which we converted to an `sf` geometry.

```
class(dem.ne.m.sf)

## [1] "sf"         "data.frame"

system.time(
  ok.gls.resid <- krige(gls.resid ~ 1,
                        loc=ne.m, newdata=dem.ne.m.sf,
                        model=vmf.r.gls)
)

## [using ordinary kriging]
##    user  system elapsed
##   2.895   0.006   2.903

summary(ok.gls.resid)

##    var1.pred           var1.var              geometry
##  Min.   :-465.7709   Min.   :  748   POINT        :40052
##  1st Qu.: -42.1851   1st Qu.:30931   epsg:NA      :    0
##  Median :  -0.0499   Median :38400   +proj=aea ...:    0
##  Mean   :  -0.6000   Mean   :36349
##  3rd Qu.:  30.2732   3rd Qu.:44759
##  Max.   : 574.7911   Max.   :44900

hist(ok.gls.resid$var1.pred,
     main = "OK deviations from GLS trend surface", xlab = "GDD50")
abline(v=0, col="red")
rug(ok.gls.resid$var1.pred)
```



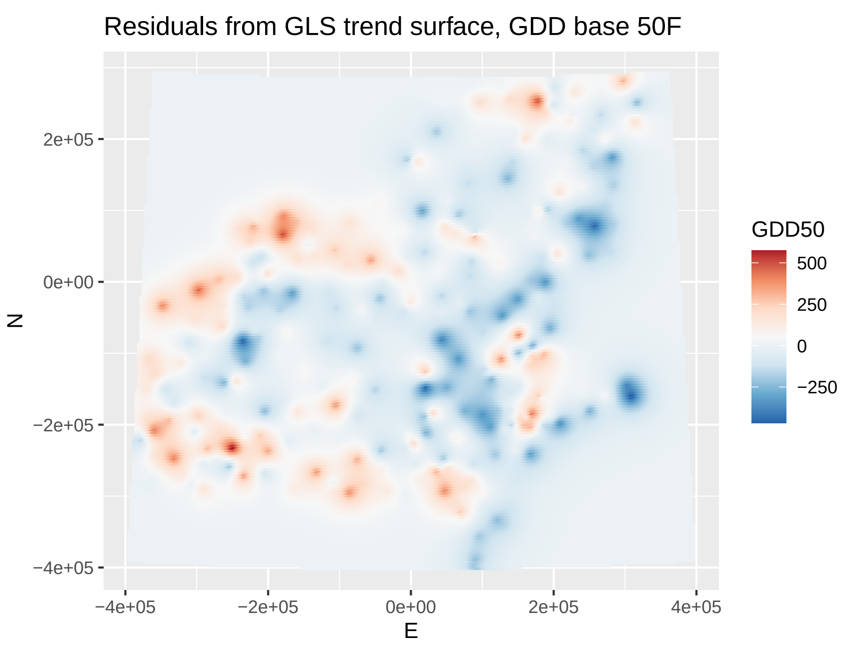
Half of the adjustments are between about ±40 GDD50, so not very many; however there are some quite large adjustments at the extremes. The kriging prediction variance has a very low value at a grid cell centre that must be close to an observation, but otherwise is fairly large; the median prediction standard deviation is 196 GDD50.

Task 40 : Add this residual and its prediction variance to the GLS trend surface data frame. •

```
dem.ne.m.df$ok.gls.resid <- ok.gls.resid$var1.pred
dem.ne.m.df$ok.gls.resid.var <- ok.gls.resid$var1.var
```

Task 41 : Display a map of the deviations. •

```
ggplot() +
    geom_point(aes(x=E, y=N, colour=ok.gls.resid), data=dem.ne.m.df) +
    xlab("E") + ylab("N") + coord_fixed() +
    ggtitle("Residuals from GLS trend surface, GDD base 50F") +
    scale_colour_distiller(name="GDD50", space="Lab", palette="RdBu")
```



Q15 : *Where are the largest adjustments to the GLS trend?* *Jump to A15* •

Q16 : *Notice there is no adjustment further than about 50 km from a station, why?* *Jump to A16* •

The predictions at these points are the **spatial** mean of the GLS residuals (*not* necessarily their arithmetic mean):

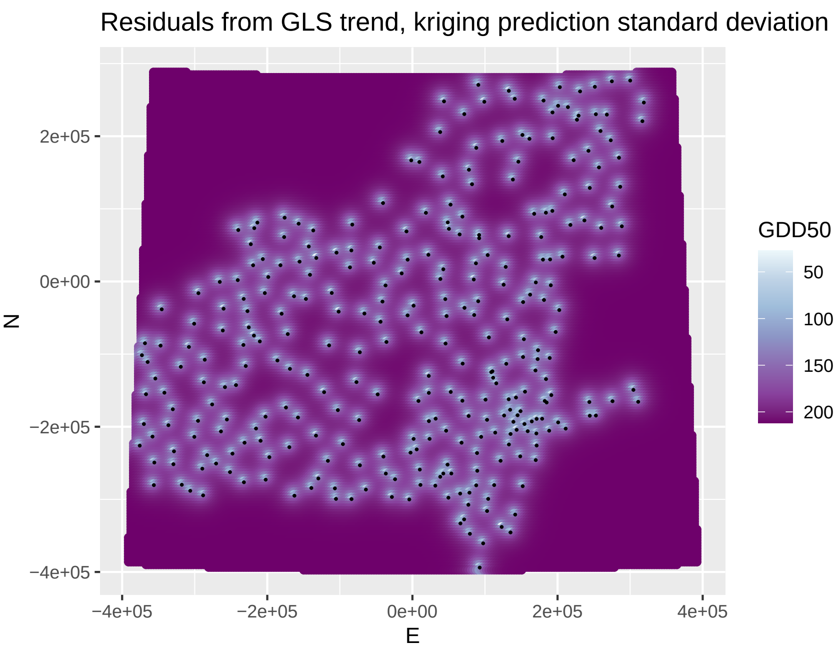
```
mean(dem.ne.m.df$ok.gls.resid)  # spatial mean over the grid

## [1] -0.6000178

mean(ne.m$gls.resid)            # arithmetic mean at observation points
```

```
## [1] 8.495601
```

Task 42 : Display the kriging prediction standard deviation of the residuals. •

This is the square root of the prediction variance, so in the same units as the prediction. We also show the observation points, to show how the kriging prediction variance depends on the point configuration.

```
ggplot() +
   geom_point(aes(x=E, y=N, colour=sqrt(ok.gls.resid.var)),
              data=dem.ne.m.df) + xlab("E") + ylab("N") +
   coord_fixed() +
   geom_point(aes(x=E, y=N),  data=ne.df, size=0.5,
              colour="black", shape=I(20)) +
   ggtitle("Residuals from GLS trend, kriging prediction standard deviation") +
   scale_colour_distiller(name="GDD50", space="Lab",
                          palette="BuPu", trans="reverse")
```

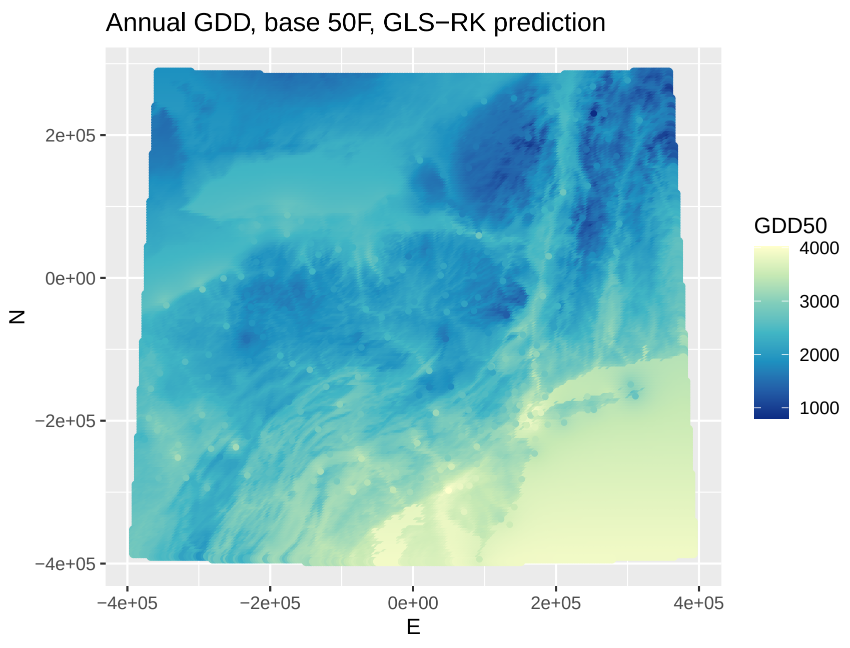


7.3 The GLS-Regression Kriging prediction

We have two predictions: the trend and the deviations from it. Adding these will give a final prediction.

Task 43 : Add the kriged GLS residuals to the trend surfaces for a final GLS-RK prediction. •

```
summary(dem.ne.m.df$pred.rkgls <-
            dem.ne.m.df$pred.gls + dem.ne.m.df$ok.gls.resid)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   921.4  1973.9  2354.9  2466.0  2827.7  3977.5

display.prediction.map("pred.rkgls", "Annual GDD, base 50F, GLS-RK prediction",
                       "GDD50")
```



Q17 : *How does this compare to the GLS trend surface map? Can you see the local adjustments?* *Jump to A17* •

7.4 Area of Applicability

In the previous section we predicted over a bounding box covering the four States. This predicted into areas where there were no observations, e.g., parts of CT, MA, NH, MD as well as Ontario. This is called **extrapolation**, as opposed to **interpolation**. Of course, some areas in the four States are outside the convex hull of the points, e.g., near the borders of these four States adjacent to States not in the study area. But here, (1) observation points are never far away, because (2) points were chosen to cover the area. Only (1) is valid outside the study area, and only for a limited distance.

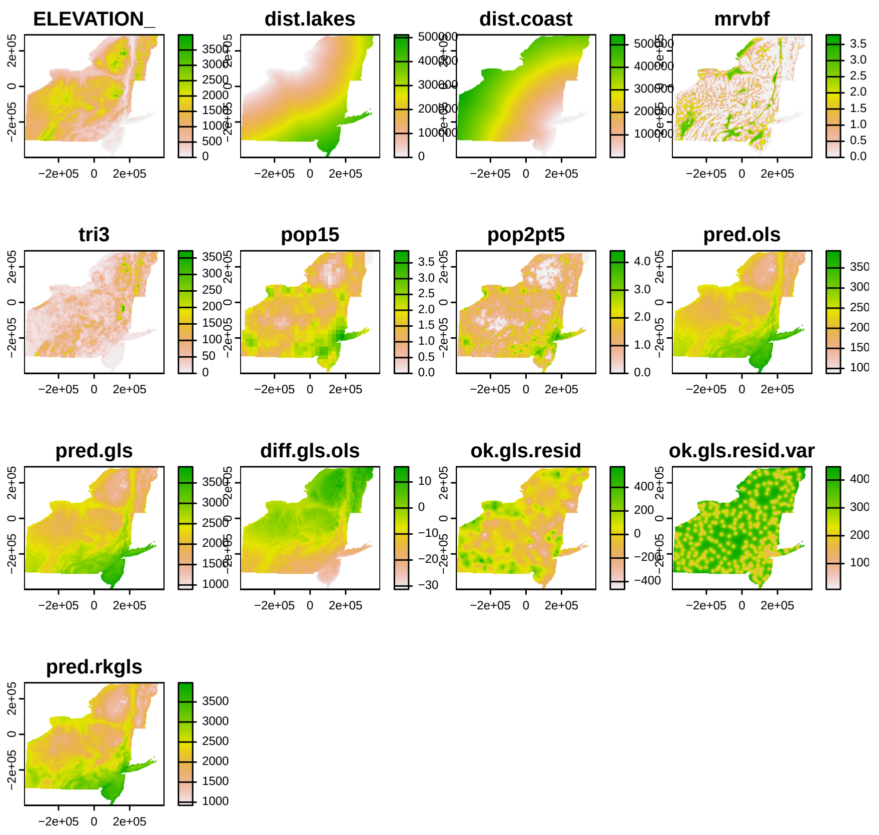
Q18 : *For what areas in the bounding box, outside the four States, are you* *confident that the prediction is as good as that inside? (Hint: for the OK part, see the kriging prediction variance map.)* *Jump to A18* •

Also, the map user from these four States will expect a map showing only these.

Task 44 : Limit the map to the four States. •

The map is now a dataframe. Convert to a `SpatRaster`, a class defined by the `terra` package, and then mask it with the State polygons.

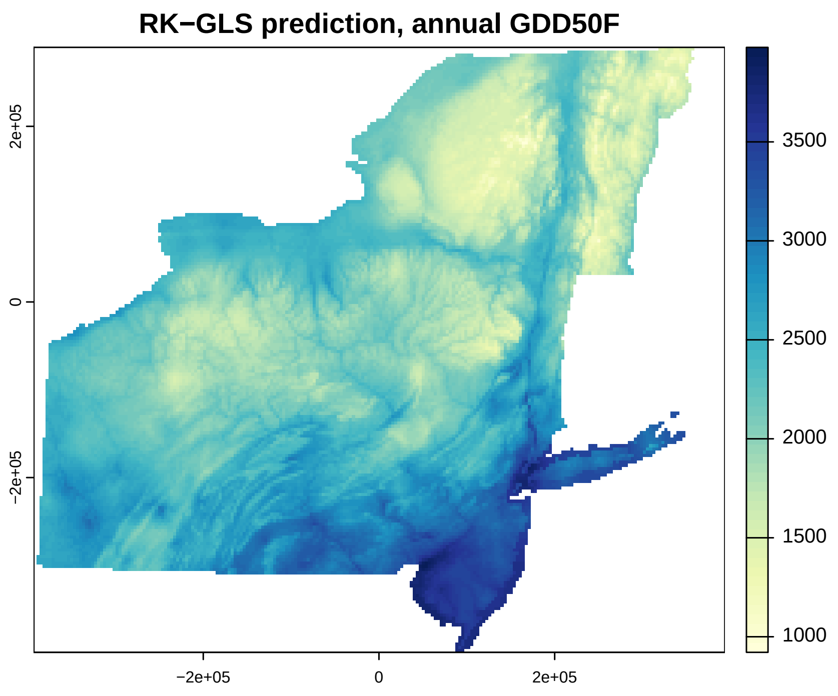
```
require(terra)
str(dem.ne.m.df)

## 'data.frame': 40052 obs. of  15 variables:
##  $ E               : num  -356674 -353224 -349774 -346324 -342874 ...
##  $ N               : num  288086 288086 288086 288086 288086 ...
##  $ ELEVATION_      : num  574 574 574 574 574 ...
##  $ dist.lakes      : num  172849 171344 169500 167491 165530 ...
##  $ dist.coast      : num  708701 706111 703528 700953 698386 ...
##  $ mrvbf           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ tri3            : num  0 0 0.00195 0.02403 0.13367 ...
##  $ pop15           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pop2pt5         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pred.ols        : num  1860 1860 1860 1860 1860 ...
##  $ pred.gls        : num  1866 1866 1866 1866 1866 ...
##   ..- attr(*, "label")= chr "Predicted values"
##  $ diff.gls.ols    : num  6.28 6.28 6.28 6.28 6.28 ...
##   ..- attr(*, "label")= chr "Predicted values"
##  $ ok.gls.resid    : num  0.000379 0.000404 0.000432 0.000462 0.000495 ...
##  $ ok.gls.resid.var: num  44900 44900 44900 44900 44900 ...
##  $ pred.rkgls      : num  1866 1866 1866 1866 1866 ...
##   ..- attr(*, "label")= chr "Predicted values"

# by default the first two fields are taken as the coordinates
# get the CRS from the plygon object that will be the mask
pred.ne.m.rast <- rast(dem.ne.m.df, crs=st_crs(state.ne.m)$proj4string)
print(pred.ne.m.rast)

## class       : SpatRaster
## dimensions  : 186, 228, 13  (nrow, ncol, nlyr)
## resolution  : 3450, 3704  (x, y)
## extent      : -392899.3, 393700.7, -399006.4, 289937.6  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=aea +lat_0=42.5 +lon_0=-76 +lat_1=39 +lat_2=44 +x_0=0 +y_0=0 +ellps=WGS84 +unit
## source(s)   : memory
## names       : ELEVATION_, dist.lakes,   dist.coast,    mrvbf,    tri3,   pop15, ...
## min values  :      0.000,        0.0, 2.326731e+00, 0.000000,   0.000, 0.00000, ...
## max values  :   4156.336,   648951.9, 7.087011e+05, 3.991617, 371.698, 3.88183, ...

pred.ne.m.rast <- mask(pred.ne.m.rast, state.ne.m)
terra::plot(pred.ne.m.rast)
```



The default colour ramp `rev(grDevices::terrain.colors(50))` is not suitable for all the layers. In the previous value maps we used the `"YlGnBu"` palette from the `RColorBrewer` package, which was called from within the `ggplot2` plotting functions. However, for the `plot` method applied to `terra` objects, we must load that package and explicitly call the palette with the `brewer.pal` function.

Any single layer can be selected:

```r
require(RColorBrewer)
terra::plot(pred.ne.m.rast, y = "pred.rkgls",
            col = colorRampPalette( brewer.pal(n = 9, name = "YlGnBu"))(64),
            main = "RK-GLS prediction, annual GDD50F")
```



8 Kriging with external drift (KED)

Another way to predict using both the trend and local deviations from it is the one-step method Kriging with External Drift (KED)[17]. This is a form of **kriging**, i.e., a linear weighted average from the n known points with value z_i to obtain the value at each unknown point z_0 (Eqn.17). The weights λ_i must sum to 1 (i.e., estimation of the mean is unbiased), and are selected to **minimize the prediction variance** – this is the reason OK is called a "Best Unbiased Linear Predictor" (BLUP).

$$z_0 = \sum_{i=1}^{n} \lambda_i z_i \tag{17}$$

The difference between KED and OK (§7.1) is that KED also includes co-variates in the kriging system, so that the linear trend with covariates and the local deviations at each prediction point are solved together to obtain the weights λ.

[17] KED is mathematically equivalent to what is called Universal Kriging (UK); that term is often reserved for KED when only coördinates are used as covariables.

8.1 * The Universal Kriging system

The weights λ_i are determined from the **Universal Kriging System**, which is derived from an expression for the prediction variance, which is minimized to derive these equations.

Weights are found by solving:

$$\mathrm{A_U}\lambda_\mathrm{U} = \mathrm{b_U} \tag{18}$$

where

$$\mathrm{A_U} = \begin{bmatrix} \gamma(\mathrm{x}_1,\mathrm{x}_1) & \cdots & \gamma(\mathrm{x}_1,\mathrm{x}_N) & 1 & f_1(\mathrm{x}_1) & \cdots & f_k(\mathrm{x}_1) \\ \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ \gamma(\mathrm{x}_N,\mathrm{x}_1) & \cdots & \gamma(\mathrm{x}_N,\mathrm{x}_N) & 1 & f_1(\mathrm{x}_N) & \cdots & f_k(\mathrm{x}_N) \\ 1 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ f_1(\mathrm{x}_1) & \cdots & f_1(\mathrm{x}_N) & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ f_k(\mathrm{x}_1) & \cdots & f_k(\mathrm{x}_N) & 0 & 0 & \cdots & 0 \end{bmatrix}$$

In this matrix the upper-left block $N \times N$ block is the **spatial correlation structure** of the **residuals** from the trend; these are derived from the fitted variogram model. The lower-left $k \times n$ block (and its transpose in the upper-right) are the **trend** predictor values at sample points. In KED these are the covariate values at the prediction points; in UK these are the coördinate values at these points. The rest of the matrix fits with λ_U and $\mathrm{b_U}$ to set up the solution:

$$\lambda_\mathrm{U} = \begin{bmatrix} \lambda_1 \\ \cdots \\ \lambda_N \\ \psi_0 \\ \psi_1 \\ \cdots \\ \psi_k \end{bmatrix} \qquad \mathrm{b_U} = \begin{bmatrix} \gamma(\mathrm{x}_1,\mathrm{x}_0) \\ \vdots \\ \gamma(\mathrm{x}_N,\mathrm{x}_0) \\ 1 \\ f_1(\mathrm{x}_0) \\ \vdots \\ f_k(\mathrm{x}_0) \end{bmatrix}$$

The λ_U vector contains the N weights for the sample points and the $k+1$ LaGrange multipliers (1 for the overall mean and k for the trend model). The $\mathrm{b_U}$ vector is structured like an additional column of $\mathrm{A_u}$, but referring to the point to be predicted. This contains the semivariances of the prediction point vs. the known points.

The kriging variance at a point is given by the scalar product of the weights (and multiplier) vector λ with the right-hand side of the kriging system.

$$\hat{\sigma}^2(\mathrm{x}_0) = \mathrm{b}_U^T \lambda_U \tag{19}$$

Good explanations of KED are from Webster and Oliver [24] and Goovaerts [6]; in §12 we explain Ordinary Kriging (OK), where there is no trend, only local interpolation.

8.2 Computing the empirical residual variogram

KED uses the `krige` method of the `gstat` package directly with the residual variogram, and so does not require a separate regression prediction step. KED as implemented by `krige` uses GLS to compute the trend component, with a covariance structure specified by the analyst, generally from fitting a variogram model to the residual variogram This differs from `gls`, which computes the covariance structure by REML.

In `gstat` the residuals are estimated from a linear model fit, using the `variogram` function with a formula for the trend. Since the covariance structure is not yet known, this must be by OLS.

Task 45 : Use the `variogram` function to compute the variogram of the residuals from a model of GDD50 predicted by Northing and square root of elevation, and fit an exponential model to it. Use a cutoff of 100 km and a bin size of 16 km, to have enough points in the closest bin, and to avoid very local effects. Compare to the variogram computed directly from the OLS residuals in §4.4. •

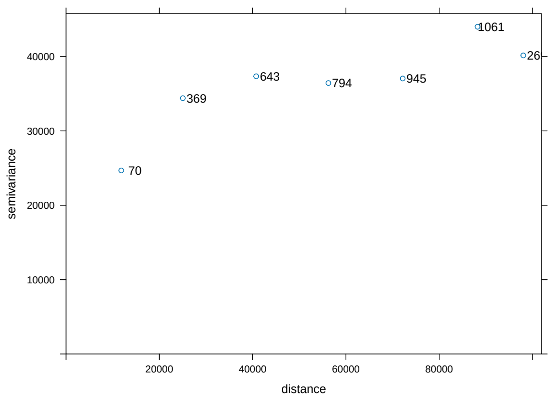
We use the `variogram` function, but instead of a null formula (right-hand side 1) to specify the spatial mean, we specify a formula for the trend. The left-hand side is the variable of interest, not the residuals from a previous step. However, since `N` is a predictor, there must be field in the object. It is in the `geometry` field of the `sf` object, but not explicitly in the data frame. So, we add it from the geometry, so it can be used in the linear model formula for the residual variogram. The Northing is the second coördinate in the coördinate matrix.

We need to do the same for the prediction object.

```
ne.m$N <- st_coordinates(ne.m)[, 2]
dem.ne.m.sf$N <- st_coordinates(dem.ne.m.sf)[, 2]
```

Now we can use these objects in KED.

```
v.ked <- variogram(ANN_GDD50 ~ sqrt(ELEVATION_) + N, locations=ne.m,
                   cutoff=100000, width=16000)
plot(v.ked, pl=T)
```



It is exactly the same empirical variogram as we computed in §4.4, because these are exactly the same residuals from the same OLS solution to the same linear model and dataset. The difference is that we don't need to find the trend surface coefficients, we just need the trend surface residuals in order to compute the variogram for KED.

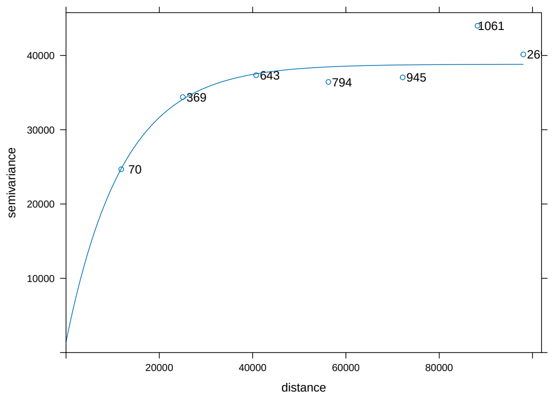
8.3 Fitting the residual variogram model

We require a continuous function of semivariance vs. separation, so that for any separation we can compute the semivariance to be used in the kriging equations. This must be an **authorized** variogram model.

Here, an exponential model appears to fit the empirical variogram of the residuals; this is a common and the simplest choice of variogram model.

The `fit.variogram` function fits variogram models specified with the `vgm` "variogram model" function. We initialize the weighted least-squares (WLS) fit to the empirical variogram with our eyeball estimates.

```
(vmf.ked <- fit.variogram(v.ked, vgm(15000, "Exp", 20000, 20000)))

##   model     psill    range
## 1   Nug  1435.842     0.00
## 2   Exp 37382.603 12106.96

plot(v.ked, plot.numbers=TRUE, model=vmf.ked)
```



The effective range of the exponential model is three times the `range` parameter, so here about 363 km. This implies that there is local structure, not explained by the covariables, to this range.

With this covariance structure, we can now predict by KED, again specifying the dependence of the target variable on the covariables.

8.4 Predicting with KED

Task 46 : Compute the KED prediction and its variance over the prediction grid. •

We call `krige` with a model formula that shows the linear dependence on covariables, exactly the same formula that we used to compute the empirical variogram of the residuals in the previous step. These two formulas must be identical.

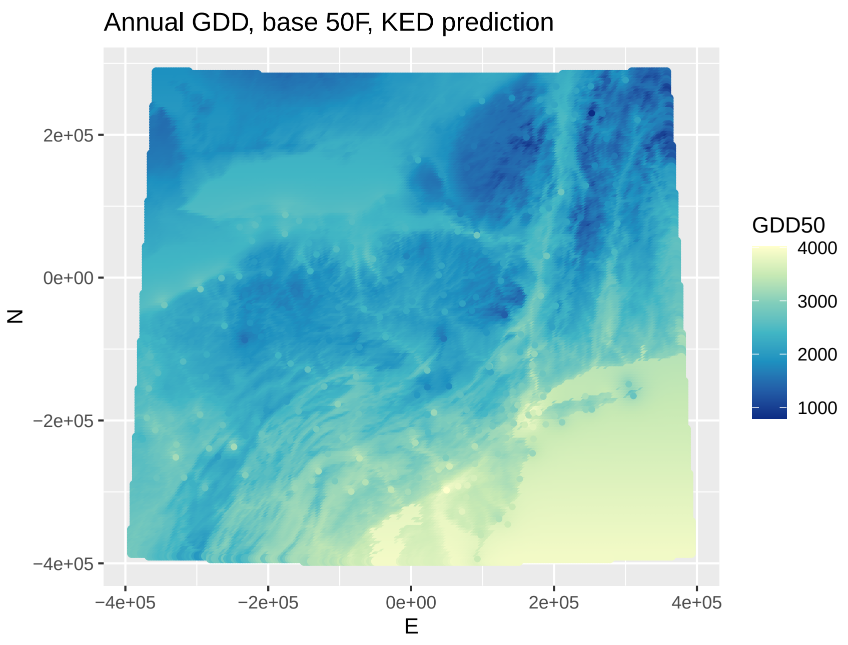
```
k.ked <- krige(ANN_GDD50 ~ sqrt(ELEVATION_)+ N, locations=ne.m,
                      newdata=dem.ne.m.sf, model=vmf.ked)

## [using universal kriging]

summary(k.ked)

##    var1.pred         var1.var              geometry
##  Min.   : 915.5   Min.   : 3689   POINT         :40052
##  1st Qu.:1984.0   1st Qu.:33102   epsg:NA       :    0
##  Median :2350.7   Median :37415   +proj=aea ...:    0
##  Mean   :2468.4   Mean   :35264
##  3rd Qu.:2823.3   3rd Qu.:39500
##  Max.   :3944.8   Max.   :42441
```

Note that the kriging prediction variance is also computed; this is known because by definition kriging minimizes it.

Task 47 : Display the prediction. •

To do this we add the prediction to the grid data frame and then display this as a map.

```
dem.ne.m.df$pred.ked <- k.ked$var1.pred
display.prediction.map("pred.ked",
                       "Annual GDD, base 50F, KED prediction",
                       "GDD50")
```



The predictions will be slightly different from the RK-GLS predictions, because the spatial correlation of the residuals was estimated from the OLS trend surface, from the GLS fit.

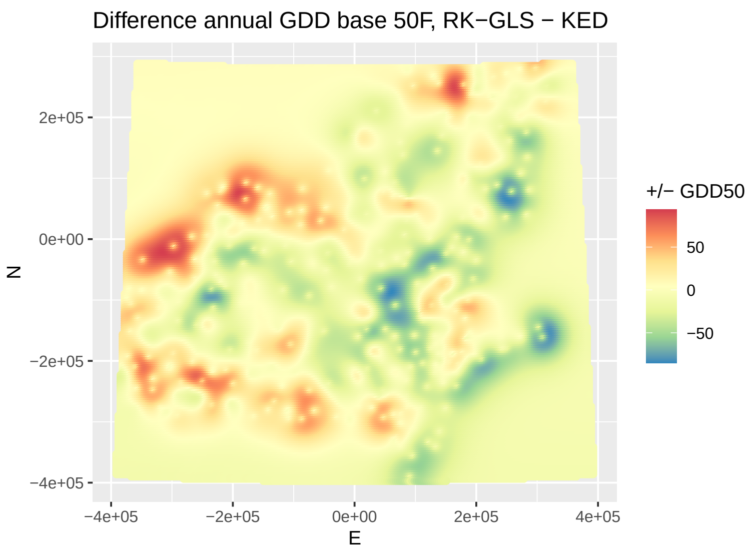
Task 48 : Compute and display the differences between RK-GLS and KED over the grid. •

```
summary(dem.ne.m.df$diff.rkgls.ked <-
            dem.ne.m.df$pred.rkgls - dem.ne.m.df$pred.ked)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -85.152 -17.335  -3.036  -2.370   9.744  93.423
```

Half of the differences are quite small, under about 15 GDD50. There are a few larger differences, but all less than 100 GDD50 (compare with the sample mean, 2518).

Display the locations of the differences:

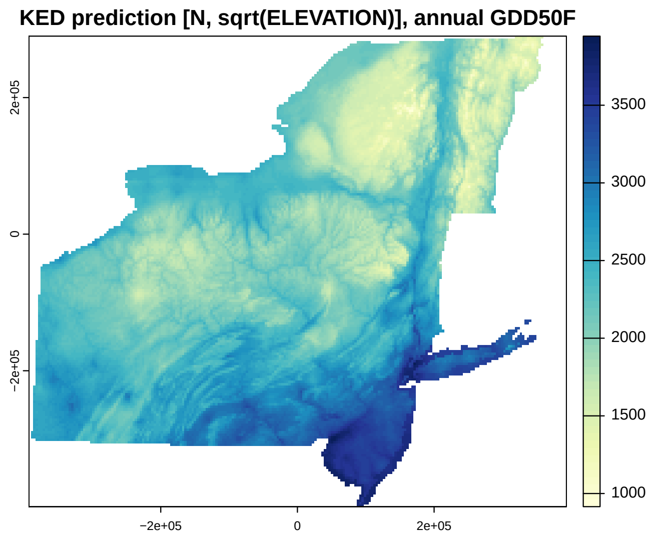
```
display.difference.map("diff.rkgls.ked",
                       "Difference annual GDD base 50F, RK-GLS - KED",
                       "+/- GDD50")
```



The largest positive residuals (RK-GLS predicts higher) are along Lake Erie and western Lake Ontario. The cooling lake effect which we saw in the OLS residuals is increased in the GLS residuals, since the GLS trend predicts somewhat lower than the OLS trend in this area. So the RK is higher here. The largest negative residuals are in the Catskills and southern VT, where the GLS trend predicted somewhat higher than the OLS trend.

Task 49 : Show the prediction limited to the four States. •

```
# by default the first two fields are taken as the coordinates
pred.ked.rast <- rast(dem.ne.m.df, crs=st_crs(state.ne.m)$proj4string)["pred.ked"]
pred.ked.rast <- mask(pred.ked.rast, state.ne.m)
terra::plot(pred.ked.rast, y = "pred.ked",
            col = colorRampPalette( brewer.pal(n = 9, name = "YlGnBu"))(64),
            main = "KED prediction [N, sqrt(ELEVATION)], annual GDD50F")
```



Challenge: There is a strong positive anomaly (RK-GLS predicts higher but just locally) near the Chazy station, on Lake Champlain in the centre North. There are several nearby stations also on the Lake. What is the reason for this anomaly?

In this section we have seen that KED has some practical advantages over RK-GLS:

1. It is easier to implement;
2. The covariance structure is estimated beforehand, and there is no risk that the procedure might not converge on a solution, as in REML;
3. It gives a prediction variance in the same step.

However, we can see from the results in this case study that there can be some fairly large differences in predictions.

8.5 Accuracy assessment

An objective way to evaluate the predictive power of KED is by **Leave-one-out cross-validation** (LOOCV). Here each point is removed from the dataset in turn, and predicted by the others, using the fitted variogram model. If the observation points well represent the total population, as they do here by design of the weather station network, this gives a good estimate of the prediction error.

Task 50 : Compute and summarize the LOOCV for this KED prediction. •

The `krige.cv` function of the `gstat` package computes this:

```
kcv.ked <- krige.cv(ANN_GDD50 ~ sqrt(ELEVATION_)+ N, locations=ne.m, model=vmf.ked)
```

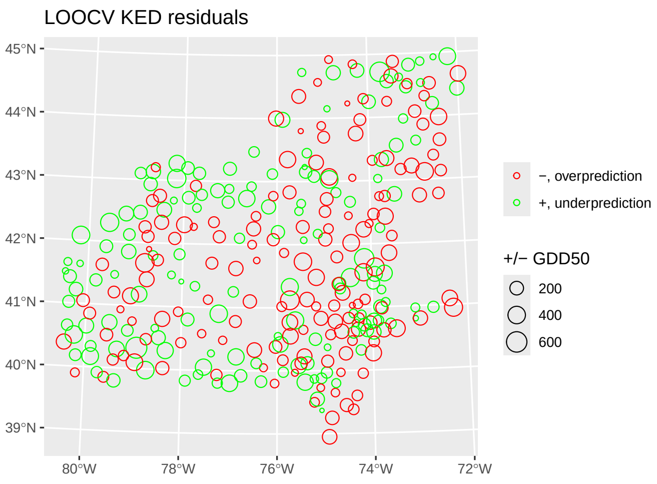
```
summary(kcv.ked$residual)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -477.630 -138.430  -11.306   -1.444  130.210  645.663
```

Overall the results are fairly good, but there are some large prediction errors at both extremes. An overall measure is the **root of the mean squared error**, RMSE:

```
(loocv.ked.rmse <- sqrt(sum(kcv.ked$residual^2)/length(kcv.ked$residual)))

## [1] 199.4083
```

Task 51 : Display a bubble plot of the LOOCV residuals. •

```
bubble.sf("kcv.ked", "residual", "GDD50", .title="LOOCV KED residuals")
```



There are several regions with intermixed fairly large under- and over-predictions; this means that in these regions there are local factors not accounted for. Other regions are consistently over- or under-predicted (Vermont mountains, Lake Ontario plain, respectively).

8.6 KED in a local neighbourhood

An advantage of KED over GLS-RK is that KED can be applied in some local neighbourhood, so that the relation with the covariables (here, Northing, Easting, square root of elevation) is re-fit at each prediction point. Besides the obvious computational advantage (fewer points → less computation), this allows a varying effect of the covariates over space. In our example, it may be that the effect on GDD50 of +100 km Northing may be more towards the south of the region than the north, or vice versa; or it may be a smaller effect in a north-south trending large valley such as the Hudson or Lake Champlain. The effect of elevation may be more, or less, in the Adirondacks in northern New York compared to the Allegheny Plateau in Pennsylvania.

Note: The linear model is *not* re-solved at each point; the UK system (§8.1) implicitly includes these in the solution. The $A_{U}a$ matrix includes the covariance betwen the neighbourhood points, as well as their values of the covariates, and the b_{U} vector includes the covariance between the neighbourhood points and the prediction point, as well as the prediction point's covariate values.

Whether the kriging is global or local, the b_{U} vector must be computed at each prediction point. For local kriging a full $A_{U}a$ matrix of all the observation points can be rapidly cut down to the set of local points closest to the prediction point.

Note: This has some relation to Geographically-Weighted Regression, where the trend surface is explicitly re-computed at each prediction point, using only the points in some neighbourhood. But in local KED we also consider the spatial correlation between the residuals of the local trend.

Task 52 : Recompute the KED prediction over the study area of §8.4 with a local neighbourhood. •

Q19 : *Why we not need or want to re-compute the residual variogram model (§8.3)?* *Jump to A19* •

The `krige` package has two optional arguments that can be used to implement this:

1. `nmax` "maximum number of neighbours to use";

2. `maxdist` "maximum distance to a point to use"; this can be used along with `nmin` "minimum number of neighbours to use" to ensure that no predictions are made with few points. This latter will produce `NA` "not available" values if there are prediction points too far from the minimum number

We prefer `nmax` because here the points are well-distributed, and we can use the kriging prediction variance to find areas that are too poorly-predicted.

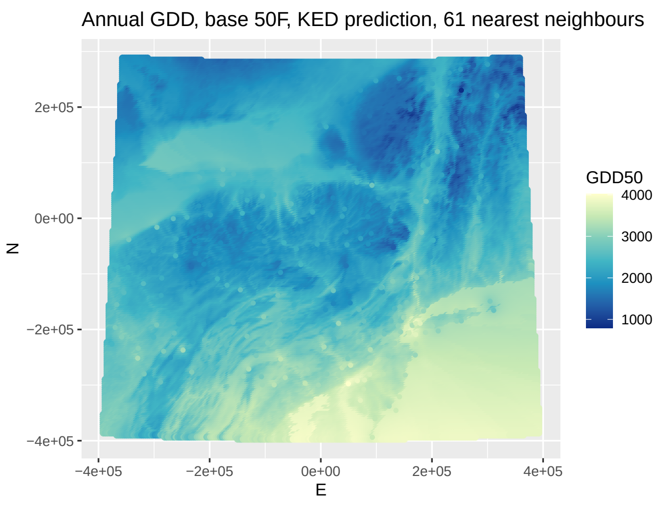
The obvious question is how to determine this number. One way is to try different numbers and compare their cross-validation statistics (see the Challenge at the end of this §). Here we choose to use 20% of the points, i.e., 305/5 ≈ 61 to show how the method works.

```
ked.n.max <- 61  # change this to try other numbers of neighbours
k.ked.nn <- krige(ANN_GDD50 ~ sqrt(ELEVATION_)+ N, locations=ne.m,
                  newdata=dem.ne.m.sf, model=vmf.ked,
                  nmax=ked.n.max)

## [using universal kriging]

summary(k.ked.nn)
```

```
##    var1.pred          var1.var                 geometry
##  Min.   : 906.4   Min.   : 3829   POINT         :40052
##  1st Qu.:1963.6   1st Qu.:33697   epsg:NA       :    0
##  Median :2379.5   Median :38411   +proj=aea ...:    0
##  Mean   :2468.2   Mean   :37254
##  3rd Qu.:2859.3   3rd Qu.:42122
##  Max.   :3947.8   Max.   :69912
```

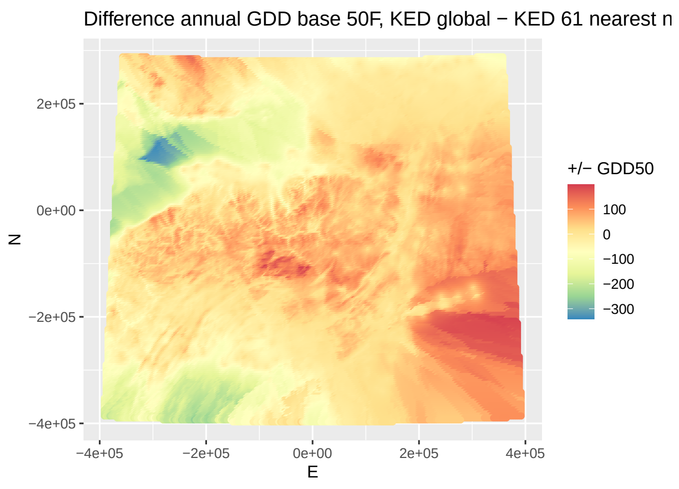
Task 53 : Display the prediction map. •

```
dem.ne.m.df$pred.ked.nn <- k.ked.nn$var1.pred
dem.ne.m.df$pred.ked.nn.sd <- sqrt(k.ked.nn$var1.var)
display.prediction.map("pred.ked.nn",
                       paste("Annual GDD, base 50F, KED prediction,",
                             ked.n.max, "nearest neighbours"),
                       "GDD50")
```



Task 54 : Compute and display a map of the difference in predictions between the global and local KED predictions. •

```
summary(dem.ne.m.df$diff.ked <- dem.ne.m.df$pred.ked - dem.ne.m.df$pred.ked.nn)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -339.9301  -36.2570   11.5159    0.1955   54.9584  198.5996

display.difference.map("diff.ked",
                       paste("Difference annual GDD base 50F, KED global - KED",
                             ked.n.max,"nearest neighbours"),
                       "+/- GDD50")
```



Q20 : *Where are the largest differences? Explain.* *Jump to A20* •

Task 55 : Compute and display the cross-validation statistics; compare them to global KED. •

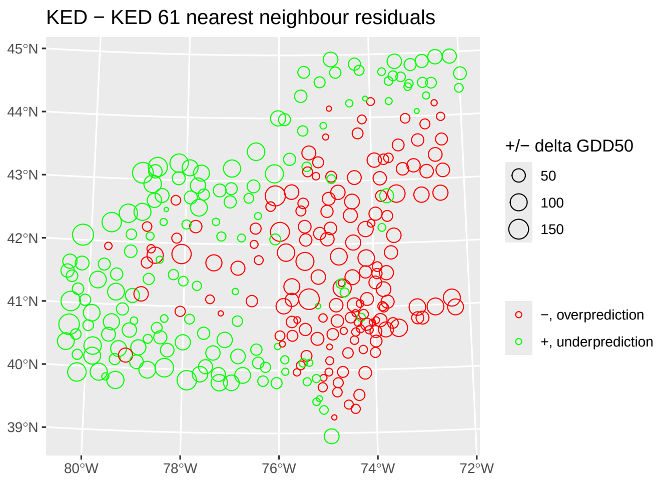
```
kcv.ked.nn <- krige.cv(ANN_GDD50 ~ sqrt(ELEVATION_)+ N,
                       locations=ne.m, model=vmf.ked,
                       nmax=ked.n.max)
```

```
summary(kcv.ked.nn$residual)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -430.555 -144.898  -18.085   -5.355  117.227  617.697

summary(kcv.ked$residual)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -477.630 -138.430  -11.306   -1.444  130.210  645.663

(loocv.ked.nn.rmse <- sqrt(sum(kcv.ked.nn$residual^2)/length(kcv.ked.nn$residual)))

## [1] 191.7587

(loocv.ked.rmse <- sqrt(sum(kcv.ked$residual^2)/length(kcv.ked$residual)))

## [1] 199.4083
```

Q21 : *Which KED, global or local, gives the best cross-validation results? What can you conclude about the strength of regional to local effects on GDD50?* *Jump to A21* •

Task 56 : Display a bubble plot of the difference between the global and local cross-validation residuals. •

```
summary(kcv.ked$diff <- kcv.ked$residual - kcv.ked.nn$residual)

##       Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -165.0122  -31.0292    0.0038    3.9111   36.4572  174.4893

bubble.sf("kcv.ked", "diff", "delta GDD50",
       .title=paste("KED - KED",
                    ked.n.max, "nearest neighbour residuals"))
```



Q22 : *Where are the largest differences? Does this seem to be geographically-consistent? Can you explain?* *Jump to A22*
•

Challenge: Experiment with different numbers of neighbours to find the optimum for local KED.

Challenge: Since the trend surface is now fit locally, it may be that Easting is significant in some parts of the region. Refit the global residual variogram with this included in the linear model, and use it in the kriging prediction formula. How much and where does this change the prediction?

8.7 * Demonstration that KED uses GLS to determine the trend

Above we stated that KED as implemented by `krige` uses GLS to compute the trend component, with a covariance structure specified by the analyst, i.e., from modelling the residual variogram. This section shows the difference between this prediction and one where the trend is computed by OLS.

We first compute the OK trend surface using `krige` *with a null model*, that is, not using any local information; this is the OLS prediction of the trend. This is the equivalent of using the `lm` function, but is more convenient because it directly produces a gridded data structure, as in kriging.

```
k.ok <- krige(ANN_GDD50 ~ sqrt(ELEVATION_)+ N, locations=ne.m,
                newdata=dem.ne.m.sf, model=NULL)

## [ordinary or weighted least squares prediction]
```

We then krige the residuals from the OLS trend of the known points; we computed those in §4.3, and computed their variogram in §4.4.

```
k.okr <- krige(ols.resid ~ 1, locations=ne.m,
               newdata=dem.ne.m.sf, model=vmf.r.ols)

## [using ordinary kriging]
```

We then add these together to get a final prediction of the trend and the local deviations from it, which is what KED does in one step:

```
k.ok$rk.pred <- k.ok$var1.pred + k.okr$var1.pred
```

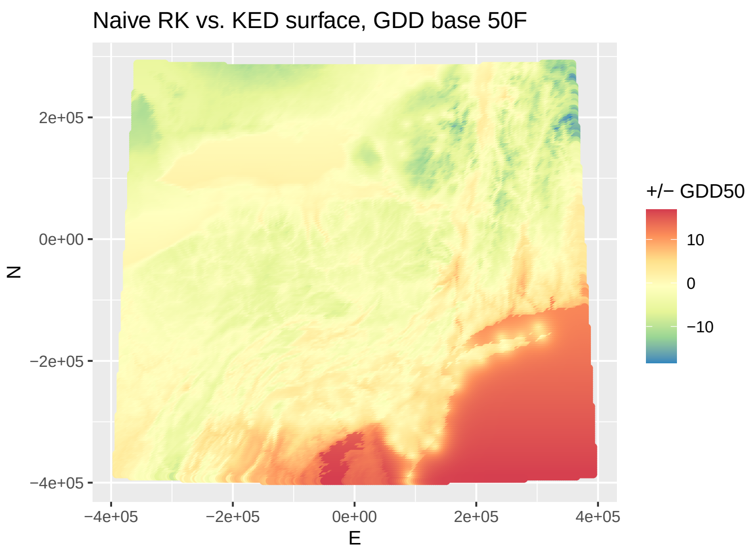
Finally, compare this to the KED prediction, and compute their differences:

```
k.ok$diff.pred <- k.ok$rk.pred - k.ked$var1.pred
summary(k.ok$rk.pred); summary(k.ked$var1.pred)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     901    1980    2349    2469    2825    3948
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   915.5  1984.0  2350.7  2468.4  2823.3  3944.8

summary(k.ok$diff.pred)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -18.381  -3.932  -1.066   0.142   1.695  16.812
```

We see a smaller magnitude difference here as when we compared KED to RK/GLS in the previous §.

The geographic distribution of the differences is because of the different trend surface:

```
dem.ne.m.df$diff.okrk.ked.pred <- k.ok$diff.pred
display.difference.map("diff.okrk.ked.pred",
                       "Naive RK vs.\ KED surface, GDD base 50F",
                       "+/- GDD50")
```



We also saw differences between GLS and OLS for the calibration points, in §5.5 and for the prediction grid in §6; here the differences are smaller because we also kriged the residuals from the two trend surfaces.

So this shows that KED as implemented by `krige` does use GLS, not OLS, to compute the trend component. The difference with RK/GLS is that the covariance structure is based on the OLS trend, not fit at the same time as the trend surface coefficients.

9 Generalized Additive Models

Generalized Additive Models (GAM) are similar to multiple linear regression, except that each term in the linear sum of predictors need not be the predictor variable itself, but can be an empirical smooth function of it. So instead of the linear model of k predictors:

$$y_i = \beta_0 + \sum_k \beta_k x_{k,i} + \varepsilon_i \tag{20}$$

we allow *functions* f_k of these:

$$y_i = \beta_0 + \sum_k f_k(x_{k,i}) + \varepsilon_i \tag{21}$$

The advantage is that non-linear relations in nature can be fit, without any need to try transformations or to fit piecewise regressions. If this is a better model fit, it should result in better predictions. The model is additive, so the marginal contribution of each predictor to the model fit can be determined. The disadvantage is that it is just an empirical fit and can not be extrapolated beyond the range of calibration. A further disadvantage is that the choice of function is arbitrary; it is generally some smooth function of the predictor, with the degree of smoothness determined by cross-validation.

> **Note:** The GAM should never be extrapolated (there is no data to support it), whereas a polynomial can, with caution, be extrapolated, on the theory that the data used to fit the model extends outside the range. This is of course very dangerous for higher-order polynomials, which are a main competitor to GAM.

Hastie et al. [7, §9.1] give a thorough explanation of GAM; a simplified explanation of the same material is given in James et al. [10, §7.7]. In a geostatistical setting, we can choose the coördinates as the predictors (as in a trend surface) but fit these with smooth functions, rather than polynomials. We can also fit any other predictor this way, e.g., in this example, the elevation.

The smooth functions can be chosen in many ways; the most common are cubic splines with knots at each value of the predictor. But we first examine whether a smooth curve, rather than one line (as in linear regression) better matches the dependence of the annual GDD50 on the three possible predictors.

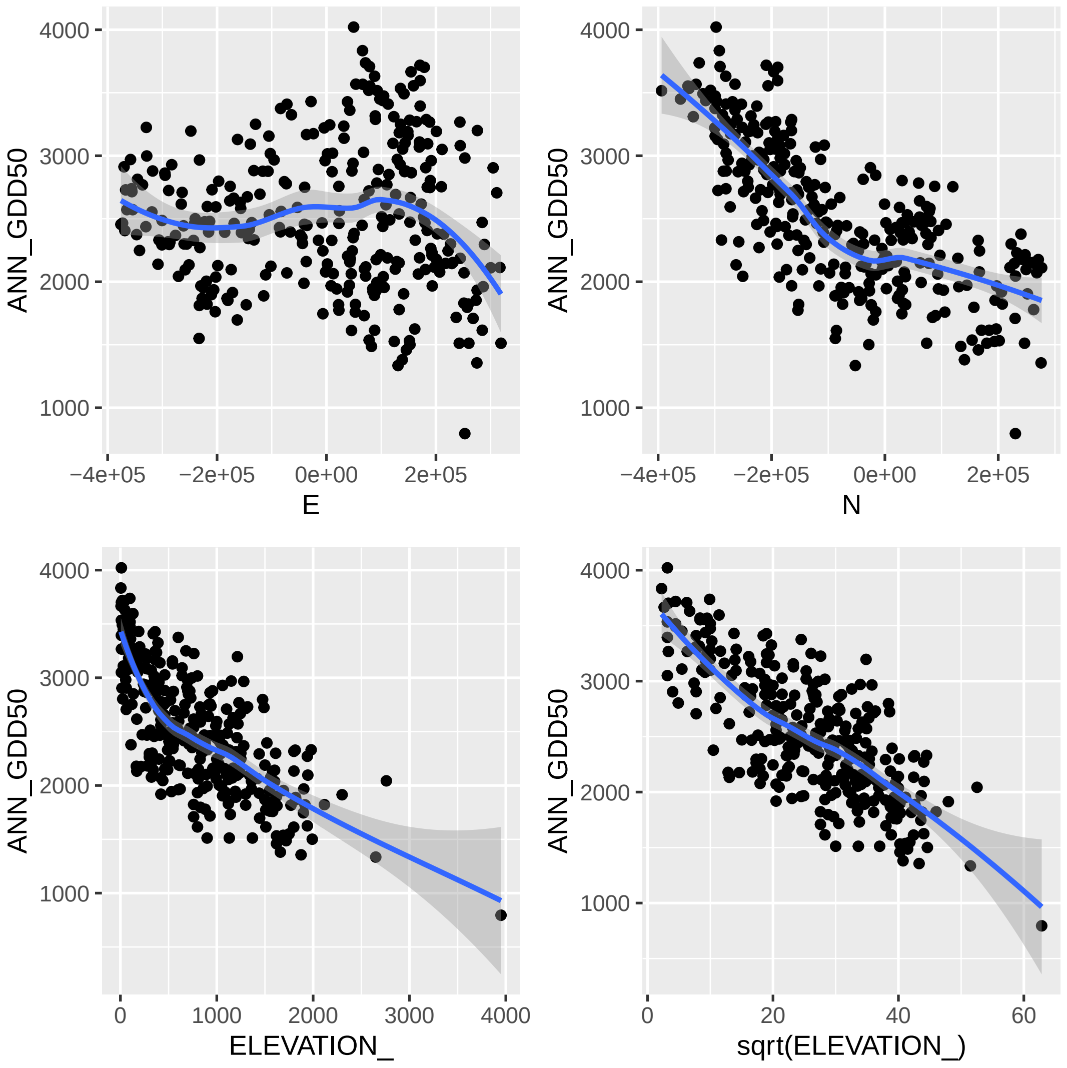
Task 57 : Display a scatterplot of the three predictors against the annual GDD50, with an empirical smoother. •

We use the `ggplot2` graphics package, introduced in §3, to produce the scatterplot and show a smoother with standard error. A simple way to visualize the trend is with a local polynomial regression, provided with the `loess` function, and incorporated into the scatterplot with the `geom_smooth` function.

> **Note:** The `loess` function has an `span` argument, which controls the degree of smoothing by setting the neighbourhood for the local fit as a proportion of the number of points. The default `span=0.75` thus uses the 3/4 of the total points closest each point. These are then weighted so that closer points have more weight; see `?loess` for details. The default works well in most situations, and here we only want a visual impression, not a "best fit" in a statistical sense.

We use the `gridExtra` package, which includes a function `grid.arrange` to arrange saved plots in a grid.

```
g1 <- ggplot(ne.df, aes(x=E, y=ANN_GDD50)) +
  geom_point() +
  geom_smooth(method="loess")
g2 <- ggplot(ne.df, aes(x=N, y=ANN_GDD50)) +
  geom_point() +
  geom_smooth(method="loess")
g3 <- ggplot(ne.df, aes(x=ELEVATION_, y=ANN_GDD50)) +
  geom_point() +
  geom_smooth(method="loess")
g4 <- ggplot(ne.df, aes(x=sqrt(ELEVATION_), y=ANN_GDD50)) +
  geom_point() +
  geom_smooth(method="loess")
require(gridExtra)
grid.arrange(g1, g2, g3, g4, ncol = 2)

## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```



Q23 : *Do these marginal relations appear to be linear in the predictors?*

Jump to A23 •

These marginal plots motivate us to try a GAM.

9.1 Fitting a Generalized Additive Model

GAM can be fit in R with the `gam` function of the `mgcv` "Mixed GAM Computation Vehicle" package. This specifies the model with a formula, as with `lm`, but terms can now be arbitrary functions of predictor variables, not just the variables themselves or simple transformations that apply to the whole range of the variable, e.g. `sqrt` or `log`. Smooth functions of one or more variables are specified with the `s` function of the `mgcv` package.

Task 58 : Load the `mgcv` package into the workspace. •

```
require(mgcv)
```

Common practice in GAM for models using coördinates is to smooth them together with a bivariate smoother, by default a thin plate regression spline; see §11 below for details. Other covariates, here the elevation, are smoothed with penalized regression splines. These control the degree of smoothness by penalizing increasingly complex models, i.e., those with more curvature; see the help text `?s` "defining smooths in GAM formulae" for details. In practice the default parameters work well.

Task 59 : Fit a GAM to the annual GDD50 at the observation stations, with the predictors being a two-dimensional thin-plate spline of the coördinates and a one-dimensional penalized regression spline of the elevation. •

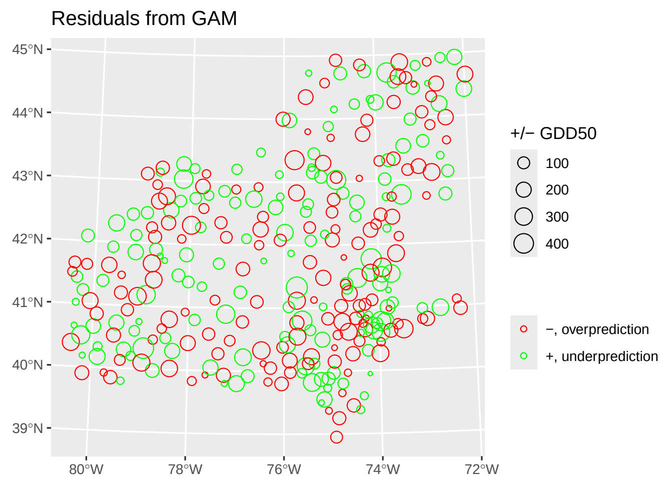
```
m.g.xy <- gam(ANN_GDD50 ~ s(E, N) + s(ELEVATION_), data=ne.df)
summary(m.g.xy)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## ANN_GDD50 ~ s(E, N) + s(ELEVATION_)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2517.518      9.986   252.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                  edf Ref.df    F p-value
## s(E,N)        23.529 27.300 37.8  <2e-16 ***
## s(ELEVATION_)  8.521  8.922 51.6  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.908   Deviance explained = 91.7%
## GCV =  34111  Scale est. = 30415     n = 305

summary(residuals(m.g.xy))

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -374.469 -110.746   -3.646    0.000   95.597  481.658
```

Q24 : *How well does this model fit the calibration observations?* *Jump to A24* •

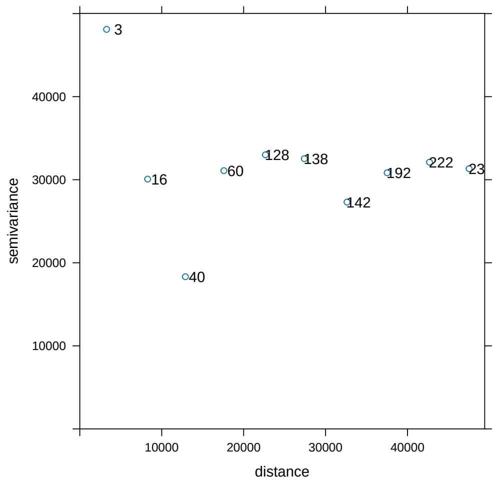
An important consideration is whether the residuals have any spatial structure; recall this is why we replaced OLS with GLS.

Task 60 : Plot the residuals as a bubble plot, and examine their spatial structure with a variogram. •

```
ne.m$resid.m.g.xy <- residuals(m.g.xy)
bubble.sf("ne.m", "resid.m.g.xy", "GDD50", "Residuals from GAM")
```



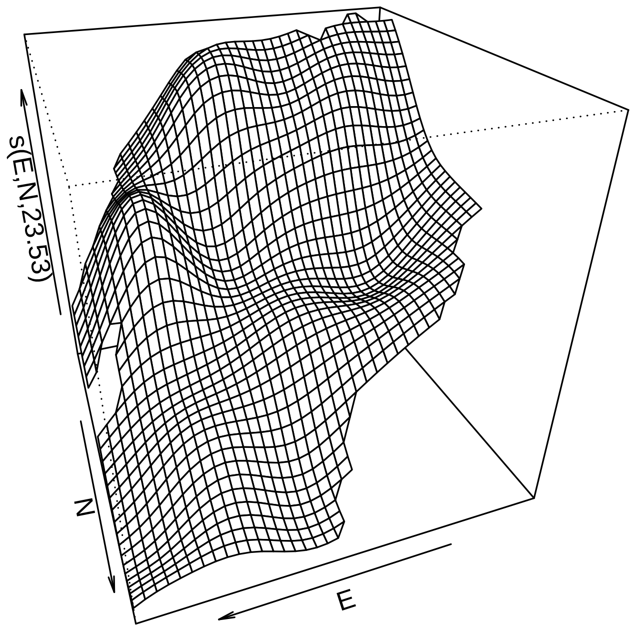
```
vr <- variogram(resid.m.g.xy ~ 1, loc=ne.m, cutoff=50000, width=5000)
plot(vr, pl=T)
```



Q25 : *Does there appear to be any local spatial correlation of the residuals? Does the empirical variogram support your conclusion?* *Jump to A25* •

The `plot.gam` function of the `mgcv` package displays the marginal smooth fit. For the 2D surface (model term `s(E,N)`, this is shown as a wireframe plot if the optional `scheme` argument is set to 1. The `select` argument selects which model term to display. We orient it to see lowest GDD towards viewer, using the `theta` argument:

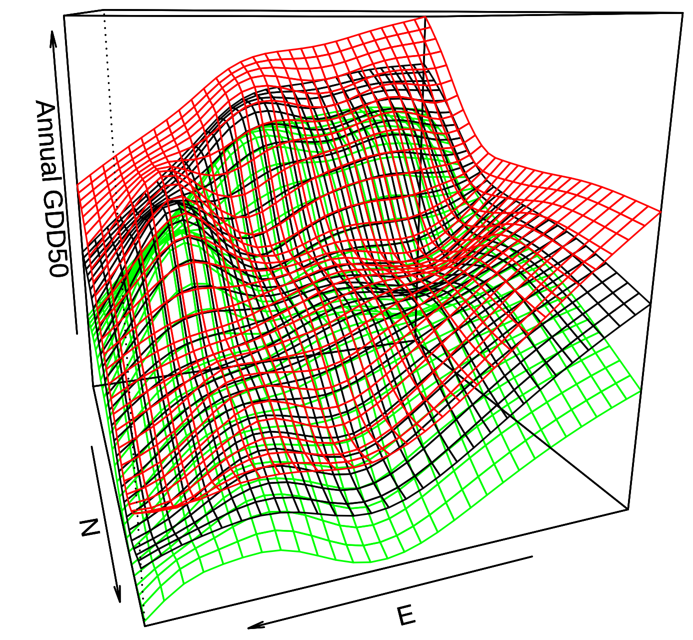
```
plot.gam(m.g.xy, rug=T, se=T, select=1,
         scheme=1, theta=30+130, phi=30)
```



Q26 : *Does the GAM 2D trend differ from a linear trend surface?* *Jump to A26* •

This surface can also be shown with the `vis.gam` function of the `mgcv` package, also showing ± 1 standard error of fit:

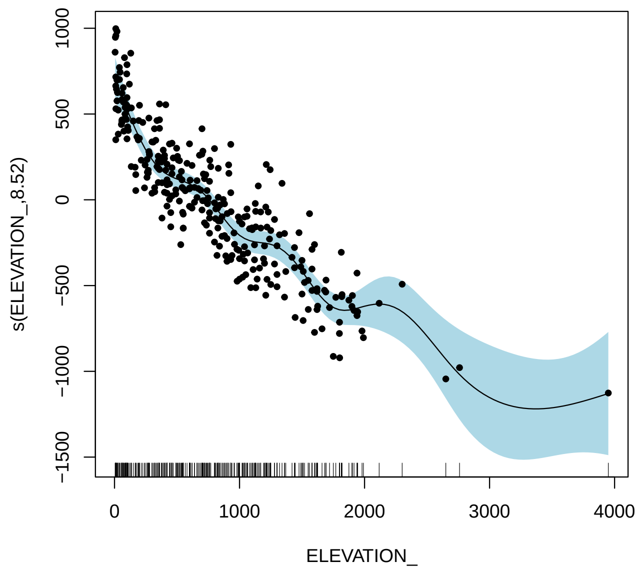
```
vis.gam(m.g.xy, plot.type="persp", color="terrain",
        theta=160, zlab="Annual GDD50", se=1.96)
```



red/green are +/− 1.96 s.e.

The marginal relation with elevation can be presented as a scatterplot, with the confidence intervals and residuals from the fit:

```r
plot.gam(m.g.xy, select=2, rug=T, se=T, residuals=T, pch=20,
         shade=T,  seWithMean=T, shade.col="lightblue")
```

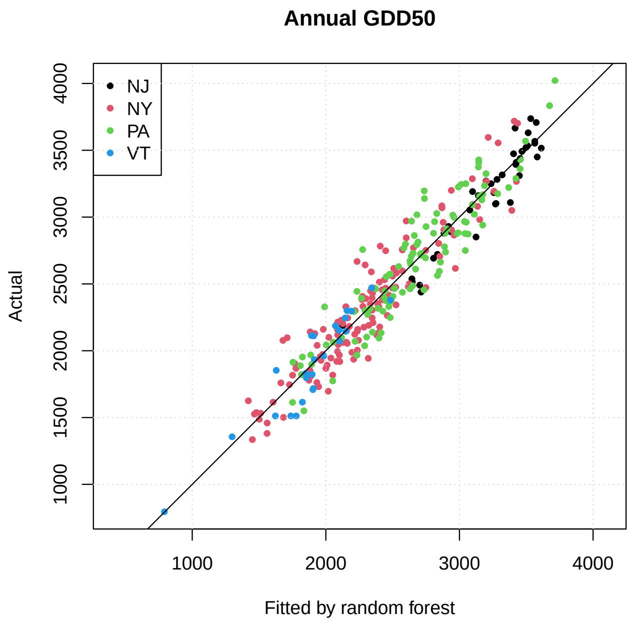


Q27 : *Does the fitted marginal relation with elevation appear to be linear?*
Jump to A27 •

Notice the very large confidence interval at the high elevations – we have so few points there that the smoother is quite uncertain in this range.

Task 61 : Compare the GAM model fits with the actual values. •

```
(rmse.gam <- sqrt(sum(residuals(m.g.xy)^2)/length(residuals(m.g.xy))))

## [1] 164.6781

plot(ne.m$ANN_GDD50 ~ predict(m.g.xy, newdata=ne.df),
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by random forest", ylab="Actual",
     main="Annual GDD50")
legend("topleft", levels(ne.df$STATE), pch=20, col=1:4)
grid(); abline(0,1)
```



The RMSE is 164.7; there are no observations that are particularly badly-fit.

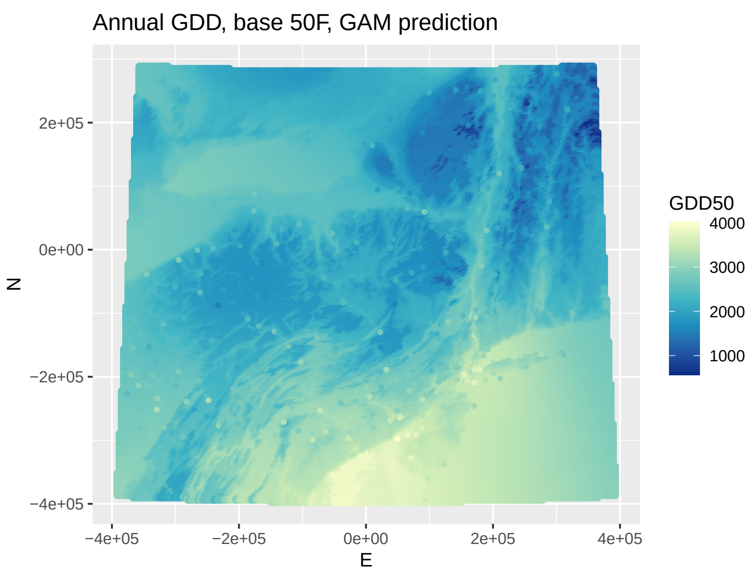
9.2 GAM prediction over the study area

Since we now have a model which uses the covariables known across the prediction grid, we can use the model to predict.

Task 62 : Predict the annual GDD50, and the standard error of prediction, across the prediction grid, using the fitted GAM, and display the predictions. •

The `predict.gam` function predicts from a fitted GAM. The `se.fit` optional argument specifies that the standard error of prediction should also be computed.

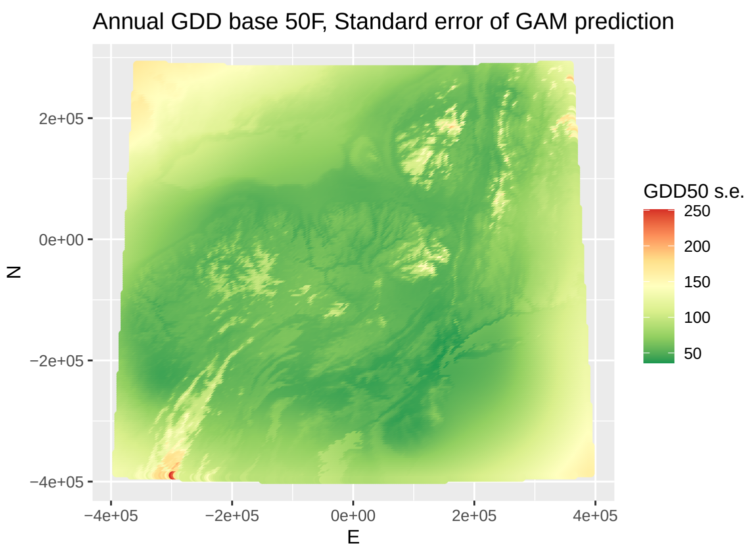
```
tmp <- predict.gam(object=m.g.xy, newdata=dem.ne.m.df, se.fit=TRUE)
summary(tmp$fit)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     563    2030    2417    2446    2841    3890

summary(tmp$se.fit)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   36.01   55.90   65.61   75.45   89.32  250.56

dem.ne.m.df$pred.gam <- tmp$fit
dem.ne.m.df$pred.gam.se <- tmp$se.fit
display.prediction.map("pred.gam", "Annual GDD, base 50F, GAM prediction",
                       "GDD50")
```



This map shows more detail than the OLS and GLS maps, especially in the high elevations and along the Atlantic coast.

Task 63 : Display the map of the standard errors of prediction. •

```
ggplot() +
    geom_point(aes(x=E, y=N, colour=pred.gam.se), data=dem.ne.m.df) +
    xlab("E") + ylab("N") + coord_fixed() +
    ggtitle("Annual GDD base 50F, Standard error of GAM prediction") +
    scale_colour_distiller(name="GDD50 s.e.", space="Lab", palette="RdYlGn",
                           direction=-1)
```



Consistent with the marginal plots, we see that the standard error is much higher at the highest elevations, where there are few observations to support the GAM.

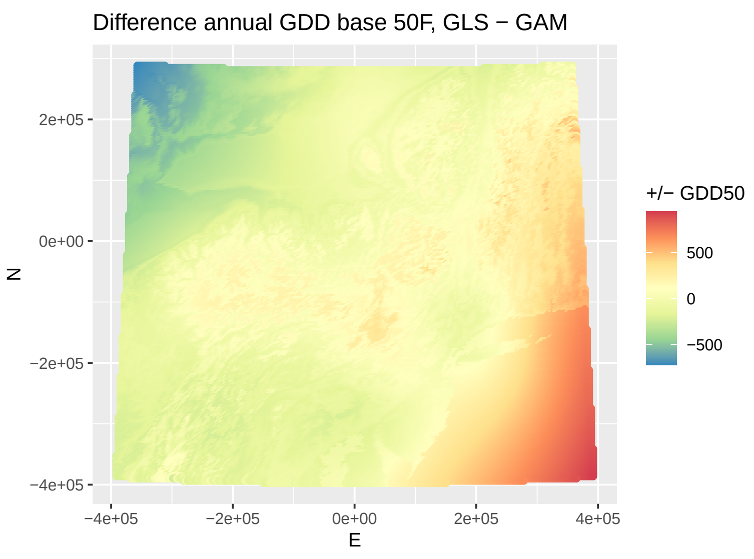
An obvious question is where this map differs from the GLS and GLS-RK maps.

Task 64 : Compute the differences in GLS and GAM model predictions over the grid, summarize numerically, and display as a difference map. •

```
summary(dem.ne.m.df$diff.gls.gam <-
            dem.ne.m.df$pred.gls - dem.ne.m.df$pred.gam)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -721.75 -136.10   -3.71   20.65  147.47  942.64
```

There are some large differences. See where these are located:

```
display.difference.map("diff.gls.gam",
                       "Difference annual GDD base 50F, GLS - GAM",
                       "+/- GDD50")
```



Q28 : *Where are the largest differences between the GAM and GLS predictions?* *Jump to A28*

•

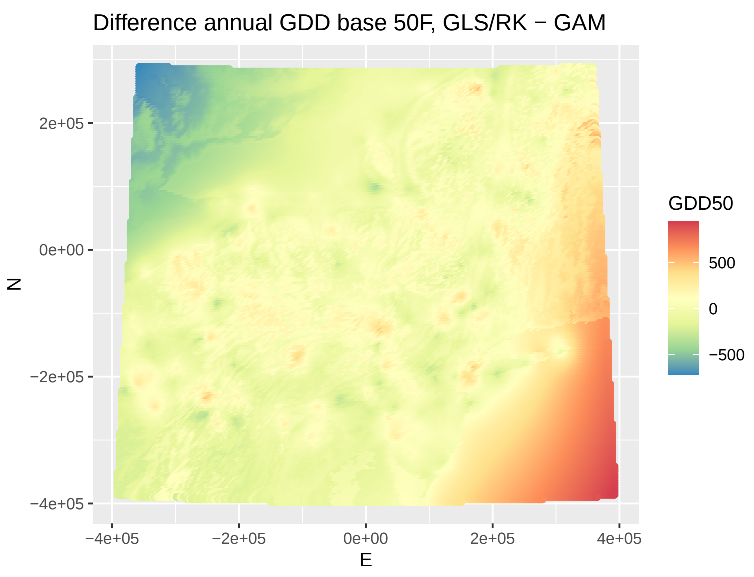
The RK component of GLS-RK allowed for local adjustment to the overall trend surface, based on local spatial dependence. GAM also adjusts locally, via its smoothers. How close are these predictions?

Task 65 : Compute the differences in GLS-RK and GAM model predictions over the grid, summarize numerically, and display as a difference map. •

```
dem.ne.m.df$diff.rkgls.gam <- dem.ne.m.df$pred.rkgls - dem.ne.m.df$pred.gam
summary(dem.ne.m.df$diff.rkgls.gam)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -721.753 -104.737   -5.354   20.046  117.922  942.634
```

There are some large differences. See where these are located:

```
ggplot() +
    geom_point(aes(x=E, y=N, colour=diff.rkgls.gam), data=dem.ne.m.df) +
    xlab("E") + ylab("N") + coord_fixed() +
    ggtitle("Difference annual GDD base 50F, GLS/RK - GAM") +
    scale_colour_distiller(name="GDD50", space="Lab", palette="Spectral")
```



Q29 : *Where are the largest differences between the GAM and GLS-RK predictions?* *Jump to A29* •

10 Data-driven models

A data-driven model is an alternative to linear modelling. It makes no assumptions about linearity; rather, it uses a set of **regression trees**. These partition the **feature space** of predictors into a set of rectangles in the dimensions of the feature space, i.e., defined by limits of each predictor in feature space. These rectangles each then have a simple prediction model, in the simplest case just a constant, which is a single predicted value of the response variable for all combinations of predictor variables in that feature-space rectangle. The **advantages** of this approach are:

1. no assumption that the functional form is the same throughout the range of the predictors;
2. over-fitting can be avoided by specifying large enough rectangles; their optimum size can be calculated by cost-complexity pruning.

This is a *high-variance, low-bias* method. This means that different fits of the model with different subsets of the data may result in quite different fitted models (high variance), but the predictions not be systematically different from the expected values (low bias).

A **disadvantage** of this approach is that, unlike linear models. it can not not extrapolate outside of its range of calibration, i.e., the multivariate feature-space limits of its predictors. But that might equally be considered an **advantage**[18], because it avoids any assumptions about the relation between target and predictors outside the calibration space.

Note: Hastie et al. [7, §9.2] give a thorough explanation of a tree-based regression method known as CART ("Classification and Regression Trees") [2]; these are implemented in R by the `rpart` "Recursive Partitioning" package. A simplified explanation of the same material is given in James et al. [10, §8.1].

10.1 Regression trees

We first start with the simplest data-driven approach: the **regression tree**. This replaces a regression equation, as developed in the previous section, with a **decision tree** based only only optimal splitting of the response variable by the predictors.

10.1.1 Fitting a regression tree model

The procedure is as follows:

1. We first specify the response variable and the possible predictors.
2. We specify a calibration ("training") dataset, as for the linear model.
3. The algorithm then looks for one predictor variable that "best" splits the data into two groups, and the value of that predictor on which to split. Intuitively, "best" refers to the maximum reduction in sum of within-group sums of squares in the response variable, compared to its overall sum of squares with no split; this is the same measure as used in Analysis of Variance (ANOVA); symbolically the reduction is $\mathrm{SS_T} - (\mathrm{SS_L} + \mathrm{SS_R})$, where L, R represent the "left" and "right" branches of the tree.
4. Following the split, this process is applied separately to both subgroups; this continues recursively until the subgroups either reach a minimum size (specified by us) or until no improvement can be made; that is the sum of the within-groups sum of squares can not be further reduced.
5. This model is then **pruned**, i.e., some branches are combined, by cross-validation, to avoid over-fitting.

Regression trees are implemented in the `rpart` function of the `rpart` package.

Task 66 : Load the `rpart` package. •

```
require(rpart)
```

Task 67 : Compute a regression tree for the response GDD50, from the

[18] "Elk naadeel heeft zijn voordeel" – Johann Cruijff

predictors `N`, `E`, and `ELEVATION_`. Note that there is no need to transform any predictor. •

The `rpart` function has several control options: (1) the minimum number of observations which can be considered for a split (using the `minsplit` argument); and (2) the minimum value of a complexity parameter (using the `cp` argument). This corresponds to the improvement in R^2 with each split. A small complexity parameter (close to 0) grows a larger tree, which may be over-fitting.

We set these to allow maximum splitting: split even if only two cases, using the `minsplit` optional argument. Also specify a small complexity parameter with the `cp` optional argument: keep splitting until there is less than 0.3% improvement in (unadjusted) R^2.

The model formulation is the same as for linear modelling: specify the predictand (dependent variable) on the left side of the `~` formula operator and the predictors on the right side, separated by the `+` formula operator. Note there is no interaction possible in tree models; the predictors are considered separately when determining which to use for a split.

```
m.rt <- rpart(ANN_GDD50 ~ N + E + ELEVATION_,
                  data=ne.df,
                  minsplit=2,
                  cp=0.003)
```

Task 68 : Display the fitted tree in text form. •

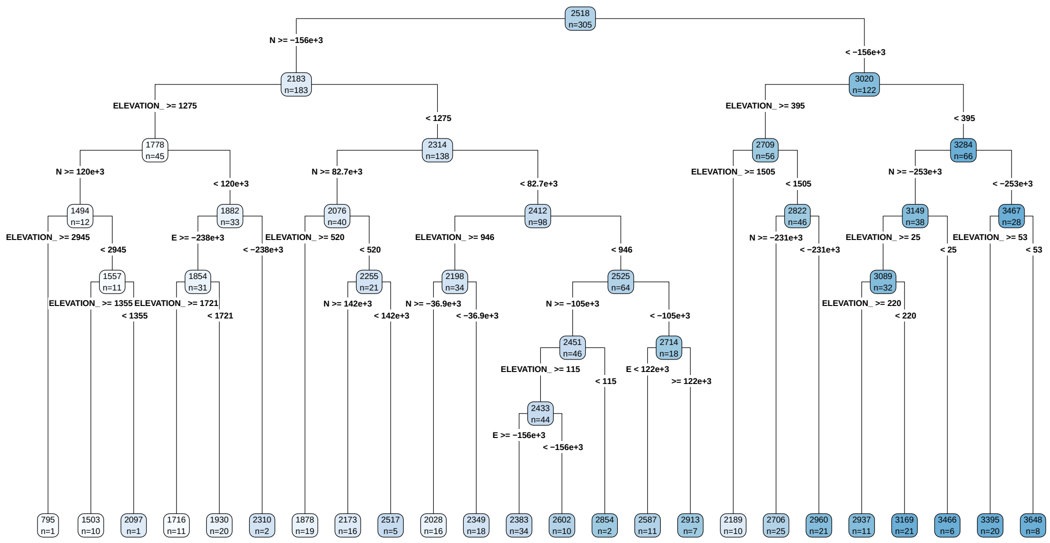
```
print(m.rt)

## n= 305
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 305 100137700.0 2517.518
##     2) N>=-155967.2 183  27412290.0 2182.536
##       4) ELEVATION_>=1275 45   3699167.0 1778.200
##         8) N>=119836.3 12    921737.0 1493.500
##          16) ELEVATION_>=2945 1         0.0  795.000 *
##          17) ELEVATION_< 2945 11    389480.0 1557.000
##            34) ELEVATION_>=1355 10     68720.0 1503.000 *
##            35) ELEVATION_< 1355 1         0.0 2097.000 *
##         9) N< 119836.3 33   1451091.0 1881.727
##          18) E>=-237967.7 31   1059064.0 1854.065
##            36) ELEVATION_>=1721 11    354595.6 1715.818 *
##            37) ELEVATION_< 1721 20    378607.8 1930.100 *
##          19) E< -237967.7 2       612.5 2310.500 *
##       5) ELEVATION_< 1275 138  13957200.0 2314.384
##        10) N>=82678.87 40   2854040.0 2076.125
##          20) ELEVATION_>=520 19    568665.7 1878.263 *
##          21) ELEVATION_< 520 21    868542.6 2255.143
##            42) N>=142190.2 16    195405.8 2173.375 *
##            43) N< 142190.2 5    223838.8 2516.800 *
##        11) N< 82678.87 98   7905649.0 2411.633
##          22) ELEVATION_>=946 34   1936208.0 2197.941
##            44) N>=-36918.74 16    529437.0 2028.250 *
##            45) N< -36918.74 18    536519.1 2348.778 *
##          23) ELEVATION_< 946 64   3592056.0 2525.156
##            46) N>=-104679.8 46   1681030.0 2451.326
##              92) ELEVATION_>=115 44   1336795.0 2433.023
##               184) E>=-156122.8 34    803000.0 2383.382 *
##              185) E< -156122.8 10    165155.6 2601.800 *
##             93) ELEVATION_< 115 2      5202.0 2854.000 *
##           47) N< -104679.8 18   1019503.0 2713.833
##             94) E< 122469.2 11    451684.0 2587.000 *
##             95) E>=122469.2 7    112794.9 2913.143 *
##     3) N< -155967.2 122  21387870.0 3019.992
##       6) ELEVATION_>=395 56   6252615.0 2709.107
##        12) ELEVATION_>=1505 10    202742.9 2189.100 *
##        13) ELEVATION_< 1505 46   2757956.0 2822.152
##          26) N>=-231009.3 25   1153309.0 2706.320 *
##          27) N< -231009.3 21    869901.0 2960.048 *
##       7) ELEVATION_< 395 66   5130590.0 3283.773
##        14) N>=-252628.4 38   2433245.0 3148.526
##          28) ELEVATION_>=25 32   1337325.0 3088.969
##            56) ELEVATION_>=220 11    161948.7 2936.545 *
##            57) ELEVATION_< 220 21    785949.2 3168.810 *
##          29) ELEVATION_< 25 6    377040.8 3466.167 *
##        15) N< -252628.4 28   1058940.0 3467.321
##          30) ELEVATION_>=52.5 20    420733.0 3394.950 *
##          31) ELEVATION_< 52.5 8    271573.5 3648.250 *
```

Task 69 : Plot the regression tree. •

The tree is graphed with the `rpart.plot` function of the `rpart.plot` package.

The `rpart.plot` function has several options to control the display; we choose to show the values of the response variable at the interior nodes as well as at the leaves, and to show the number of observations in each split and leaf. The information is the same as given with a printout of the model object, but easier to visualize.

```r
require(rpart.plot)
rpart.plot(m.rt, digits=3, type=4, extra=1)
```



Q30 : *What is the first (root) splitting variable? At what value is the split? What is the mean value of GDD50 of the whole dataset, and of the two branches? How many observations in each branch?* Jump to A30 •

Although there is no model with coefficients, we can still see which predictor variables had the most influence on the tree.

Task 70 : Display the variable importance as a proportion. •

This is the proportion of the variance explained by the tree which is due to each variable.

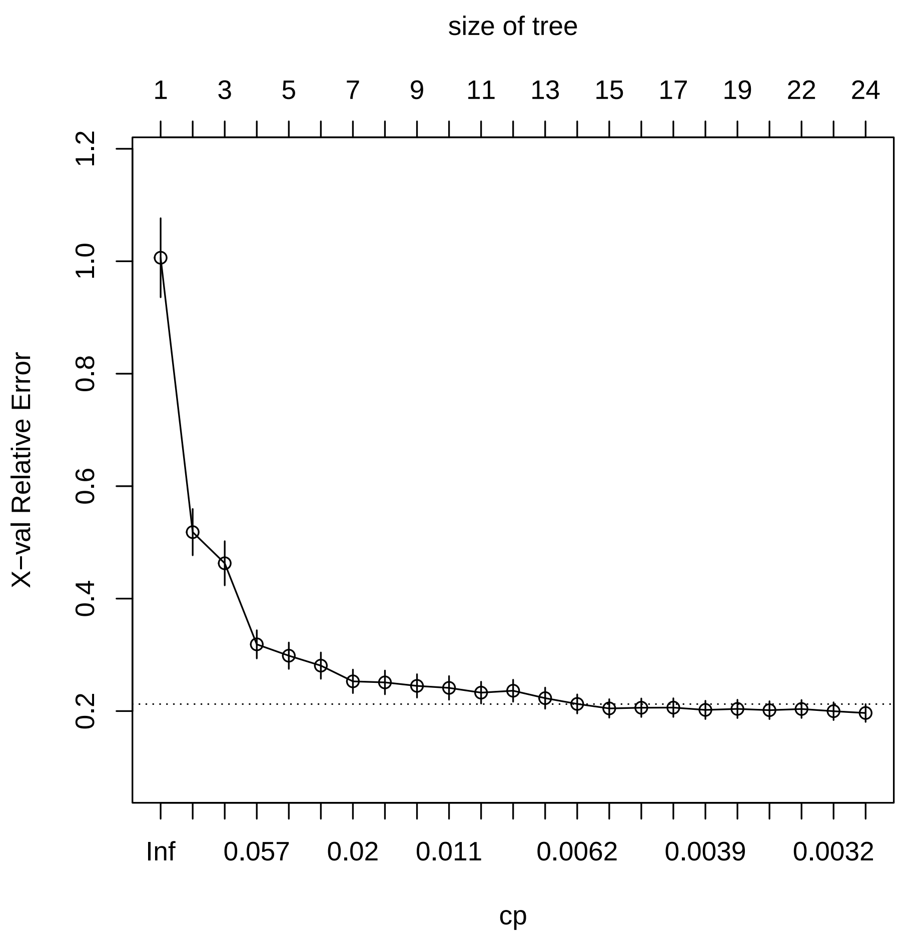
```
x <- m.rt$variable.importance
data.frame(variableImportance = 100 * x / sum(x))

##            variableImportance
## N                    48.90337
## ELEVATION_           38.60227
## E                    12.49436
```

Q31 : *Which variables are most important?* *Jump to A31* •

We now examine the reduction in fitting and cross-validation error with the `printcp` "print the complexity parameter" function.

Task 71 : Print and plot the error rate vs. the complexity parameter and tree size. •

```
printcp(m.rt)

##
## Regression tree:
## rpart(formula = ANN_GDD50 ~ N + E + ELEVATION_, data = ne.df,
##     minsplit = 2, cp = 0.003)
##
## Variables actually used in tree construction:
## [1] E          ELEVATION_ N
##
## Root node error: 100137734/305 = 328320
##
## n= 305
##
##           CP nsplit rel error  xerror     xstd
## 1  0.5126697      0  1.000000 1.00622 0.070339
## 2  0.0999090      1  0.487330 0.51823 0.041222
## 3  0.0974250      2  0.387421 0.46287 0.039271
## 4  0.0328739      3  0.289996 0.31868 0.025099
## 5  0.0319311      4  0.257122 0.29848 0.023483
## 6  0.0237411      5  0.225191 0.28081 0.023420
## 7  0.0163615      6  0.201450 0.25288 0.020905
## 8  0.0141488      7  0.185089 0.25099 0.021131
## 9  0.0132452      8  0.170940 0.24478 0.020829
## 10 0.0089030      9  0.157695 0.24123 0.021066
## 11 0.0086905     10  0.148792 0.23276 0.019475
## 12 0.0073373     11  0.140101 0.23617 0.019555
## 13 0.0071789     12  0.132764 0.22297 0.018837
## 14 0.0053152     13  0.125585 0.21269 0.016959
## 15 0.0045440     14  0.120270 0.20473 0.016469
## 16 0.0044868     15  0.115726 0.20596 0.016475
## 17 0.0039088     16  0.111239 0.20617 0.016620
## 18 0.0038889     17  0.107330 0.20208 0.016268
## 19 0.0036613     18  0.103441 0.20385 0.016333
## 20 0.0035335     19  0.099780 0.20161 0.015913
## 21 0.0032541     21  0.092713 0.20369 0.016004
## 22 0.0032032     22  0.089459 0.19978 0.015816
## 23 0.0030000     23  0.086256 0.19658 0.015869

plotcp(m.rt)
```



Note: Your results will likely be different. This is because the cross-validation makes a *random* split of the full dataset into a number of subsets for model building and evaluation. Each run gives a different random split.

The `xerror` field in the summary shows the **cross-validation error**; that is, applying the model to the original data split K-fold, each time excluding some observations. If the model is over-fitted, the cross-validation error increases; note that the fitting error, given in the `error` field, always decreases. By default, the split is 10-fold; this can be modified by the `control` argument to the `rpart` function.[19]

Q32 : *Does this model appear to be overfit? Why or why not? What appears to be the optimum complexity parameter to avoid over-fitting? Jump to A32* •

Task 72 : Prune the tree back to complexity level estimated from the previous answer. •

We do this with the `prune` function, specifying the `cp` "complexity parame-

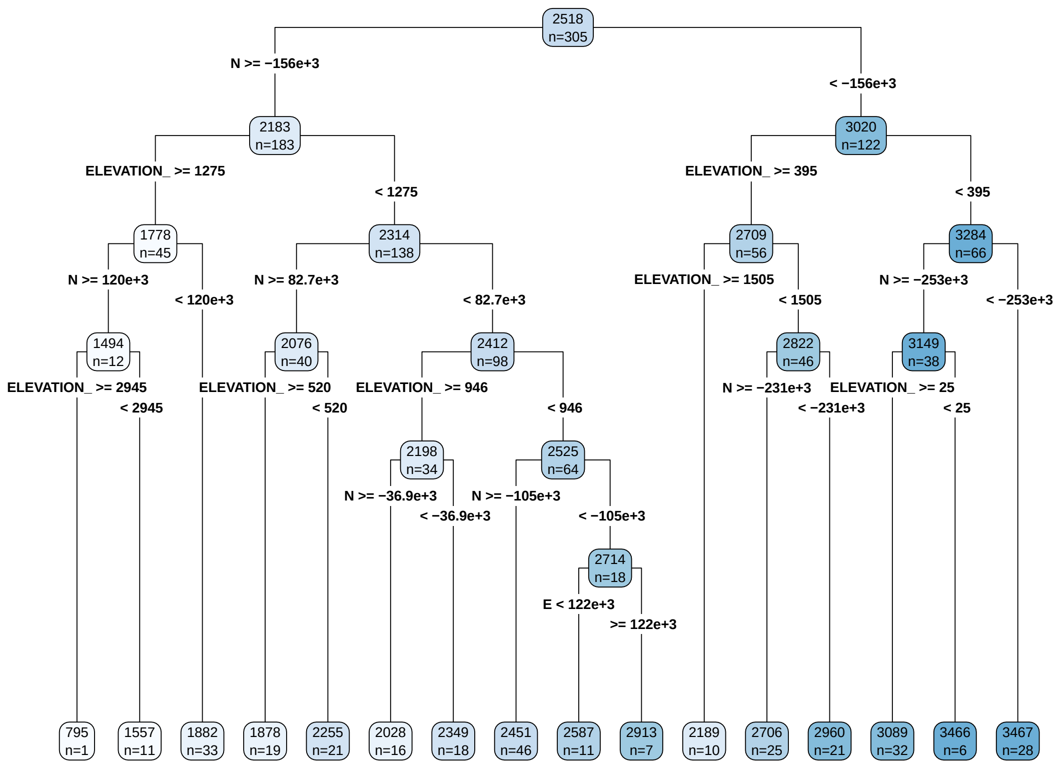
[19] See the help for `rpart.control`.

ter" argument.

```
(m.rt.p <- prune(m.rt, cp=0.0045))

## n= 305
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 305 100137700.0 2517.518
##    2) N>=-155967.2 183  27412290.0 2182.536
##      4) ELEVATION_>=1275 45   3699167.0 1778.200
##        8) N>=119836.3 12    921737.0 1493.500
##         16) ELEVATION_>=2945 1         0.0  795.000 *
##         17) ELEVATION_< 2945 11    389480.0 1557.000 *
##        9) N< 119836.3 33   1451091.0 1881.727 *
##      5) ELEVATION_< 1275 138  13957200.0 2314.384
##       10) N>=82678.87 40   2854040.0 2076.125
##         20) ELEVATION_>=520 19    568665.7 1878.263 *
##         21) ELEVATION_< 520 21    868542.6 2255.143 *
##       11) N< 82678.87 98   7905649.0 2411.633
##         22) ELEVATION_>=946 34   1936208.0 2197.941
##           44) N>=-36918.74 16    529437.0 2028.250 *
##           45) N< -36918.74 18    536519.1 2348.778 *
##         23) ELEVATION_< 946 64   3592056.0 2525.156
##           46) N>=-104679.8 46   1681030.0 2451.326 *
##           47) N< -104679.8 18   1019503.0 2713.833
##             94) E< 122469.2 11    451684.0 2587.000 *
##             95) E>=122469.2 7    112794.9 2913.143 *
##    3) N< -155967.2 122  21387870.0 3019.992
##      6) ELEVATION_>=395 56   6252615.0 2709.107
##       12) ELEVATION_>=1505 10    202742.9 2189.100 *
##       13) ELEVATION_< 1505 46   2757956.0 2822.152
##         26) N>=-231009.3 25   1153309.0 2706.320 *
##         27) N< -231009.3 21    869901.0 2960.048 *
##      7) ELEVATION_< 395 66   5130590.0 3283.773
##       14) N>=-252628.4 38   2433245.0 3148.526
##         28) ELEVATION_>=25 32   1337325.0 3088.969 *
##         29) ELEVATION_< 25 6    377040.8 3466.167 *
##       15) N< -252628.4 28   1058940.0 3467.321 *
```

Task 73 : Plot the pruned regression tree. •

```
rpart.plot(m.rt.p, digits=3, type=4, extra=1)
```



Q33 : *How does this tree differ from the original regression tree?* *Jump to A33* •

We can compare the fitted vs. actual values.

Task 74 : Use the pruned regression tree to predict at the calibration points. •

We do this with the `predict` method applied to a `rpart` object; this automatically calls function `predict.rpart`. The points to predict and the values of the predictor variables at those points are supplied in a dataframe as argument **newdata**. We count the number of predicted values with the **unique** function; there is only one value per "box" in the feature space defined by the predictor variables.

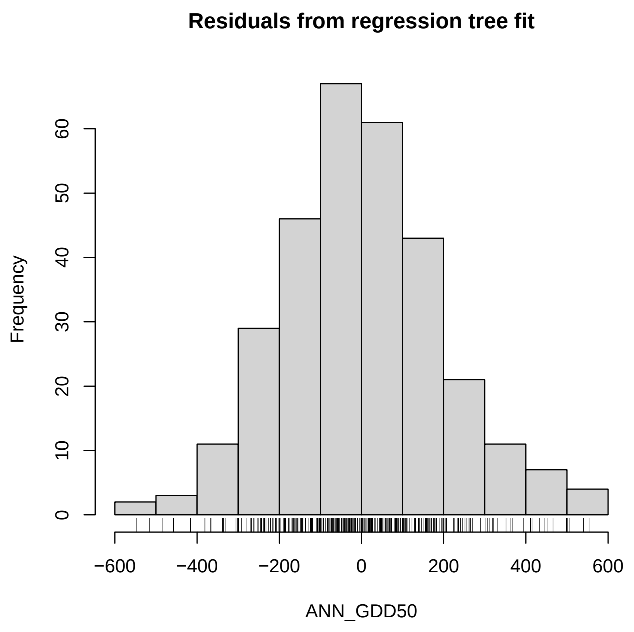
```
summary(p.rt.p <- predict(m.rt.p, newdata=ne.df))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##     795    2028    2451    2518    2960    3467
```

Task 75 : Count the unique predicted values. •

```
length(unique(p.rt.p))

## [1] 16
```

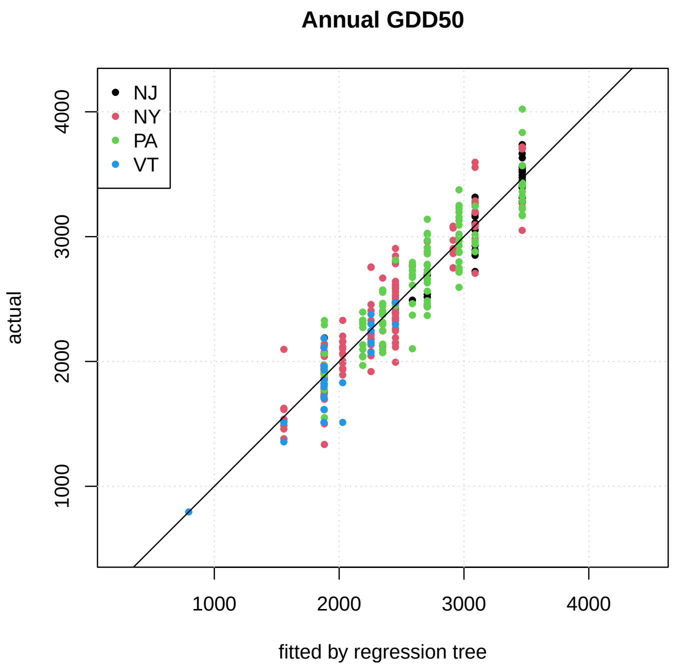
Task 76 : Compute the residuals: (actual - predicted) and the RMSE. Summarize them and present as a histogram. •

```
summary(residuals.rt.p <- ne.df$ANN_GDD50 - p.rt.p)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## -546.73 -122.73  -10.14    0.00  127.90  553.68

hist(residuals.rt.p, main="Residuals from regression tree fit",
     xlab="ANN_GDD50")
rug(residuals.rt.p)
sqrt(mean(residuals.rt.p^2)/length(residuals.rt.p))

## [1] 11.16128
```



Task 77 : Plot the actual vs. fitted values. •

```
plot(ne.df$ANN_GDD50 ~ p.rt.p, asp=1, pch=20,
     xlab="fitted by regression tree", ylab="actual",
      xlim=c(500,4200), ylim=c(500,4200),
      col=ne.df$STATE,
      main="Annual GDD50")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid()
abline(0,1)
```



Q34 : *How many unique values are predicted by the pruned regression tree? How close is the fit to the actual values, compared to the OLS or GLS models? Explain.* *Jump to A34* •

10.1.2 Regression tree prediction over the study area

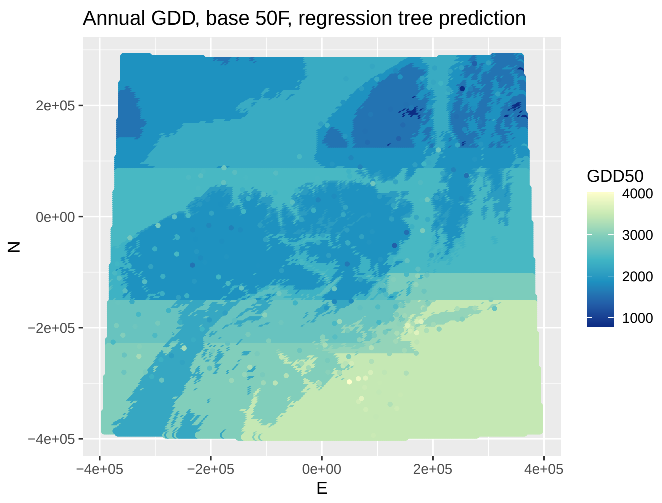
Task 78 : Predict over the grid with the regression tree model, add the results to the dataframe, and summarize them. •

Task 79 : Display the regression tree surface. •

```
dem.ne.m.df$pred.rt <- predict(m.rt.p, newdata=dem.ne.m.df)
summary(dem.ne.m.df$pred.rt)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     795    1882    2255    2432    2960    3467
```

We also add the point set in the same colour scheme; if the point is visible it means the residual (lack of fit) is large.

```
display.prediction.map("pred.rt",
                       "Annual GDD, base 50F, regression tree prediction",
                       "GDD50")
```



Q35 : *What is the spatial pattern of the regression tree prediction? Explain why.* *Jump to A35* •

10.2 Random forests

There are several problems with regression trees:

1. A small change in the sample set, for example a missing or erroneous observation, can radically change the tree.
2. Pruning the tree to avoid overfitting is also somewhat subjective.
3. Correlated predictors can appear in the tree as surrogates for each other, depending on the details of the calibration set; this makes interpretation difficult.
4. Trees do not give smooth predictions; rather they only predict a single value in each "box". Although this is statistically justified by the model, it is certainly disconcering to the map user.

To solve these problems, a method known as "random forests" was developed; see Hastie et al. [7, §15] (advanced) or James et al. [10, §8] (simplified). There are a lot of details to this method, but the basic idea is straightforward:

1. Build a **large number of regression trees**, independently, using *different* sets of observations; these are built by *sampling with replacement* from the actual observations; this is a technique known as *bagging* or *bootstrap aggregation*.

2. Save all these trees; when predicting, use all of them and **average their predictions**.
3. In addition we can summarize the whole set of trees to see how different they are, thus how robust is the final model.
4. Also, for each tree we can use observations that were not used to construct it for true **validation**, called *out-of-bag* validation. This gives a good idea of the true prediction error.

The first step may seem suspicious. The underlying idea is that what we observe is the best sample we have of reality; if we sampled again we'd expect to get similar values. So we simulate re-sampling by sampling from this same set, *with replacement*, to get a sample of the same size but a different sample. If this idea bothers you, read Efron and Gong [4], Shalizi [22] or Hastie et al. [7, §8.2].

So, how well does the random forest method work for a spatially-distributed variable?

We know that by definition a linear model, whether OLS or GLS fit, will vary smoothly with the predictors. That is, if these predictors vary smoothly in space the map of the linear model predictions will also look smooth. In this example the predictor Northing is by definition smooth, and the predictor elevation often changes smoothly, so we can expect a smooth prediction map. However, random forests are not linear. They are based on an ensemble of regression trees, with no requirement for smooth cut points.

Another attractive feature of random forests is that there is no need for predictor variable selection to avoid colinearity. Since the predictors to compare are randomly chosen at each split, it is possible for "minor" predictors which would not be included in a parametric approach to contribute to some of the trees, and thus to the ensemble prediction. In this example Easting was not used in the regression models, but should be used here. The relative importance of the predictors is reported by the RF function.

10.2.1 Fitting a Random Forest model

Task 80 : Fit a random forest model of GDD50 based on all three possible covariates: elevation, Northing and Easting. •

There are several R packages that implement random forests. A very fast implementation, widely used, is provided by the `ranger` package [29].

```
require(ranger)
```

The `ranger` function of the `ranger` package fits this model. This model requires two parameters:

1. the number of trees in the forest, optional argument `num.trees`, default value 500;

2. the number of predictors to compare at each split, optional argument `mtry`, default value is 1/3 of the predictors.

Here we accept the default for the number of predictors, which in this case will be one of the three predictors, but require more than the default number of trees

We also specify the optional `importance` argument as `"permutation"` to see two measures of predictor importance, as explained below.

Since this does not assume linearity we can use the untransformed station elevation.

```
m.rf <- ranger(ANN_GDD50 ~ ELEVATION_ + N + E,
                     data=ne.df, num.trees=1200,
                     importance="permutation")
# proportional importance
ranger::importance(m.rf)/sum(ranger::importance(m.rf))*100

## ELEVATION_          N          E
##   45.41475   43.63581   10.94944

ranger::importance(m.rf)/dim(ne.df)[1]

## ELEVATION_          N          E
##   724.1184   695.7541   174.5840
```

The "permutation" measure of predictor importance is the sum of differences of predictions errors in the out-of-bag (OOB) validations if the predictor is removed from the set. Here we divide by their sum to get the relative importance.

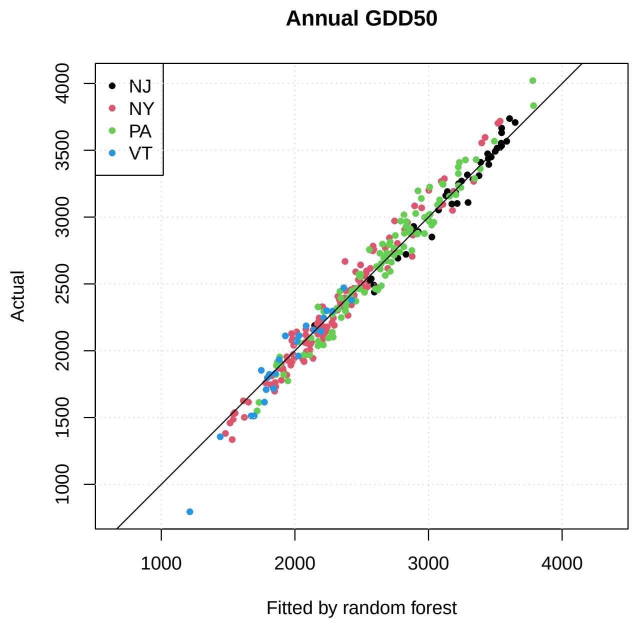
Q36 : *Which predictor is most important?* *Jump to A36* •

Task 81 : Plot the actual vs. fitted values from the random forest model. Compute the mean error (bias, ME) and the root mean squared error (RMSE). •

The `predict` function gives predicted values based on a model. To predict back at the calibration points, the `newdata` argument to the `predict` function must specify this point set.

> **Note:** Each run of the random forest will produce different fits, so your graph may not look exactly like this one.

```
summary(rf.fits <- predict(m.rf, data = ne.df)$predictions)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1214    2121    2457    2518    2896    3786

plot(ne.m$ANN_GDD50 ~ rf.fits,
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by random forest",
     ylab="Actual",
     main="Annual GDD50")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid()
abline(0,1)
```



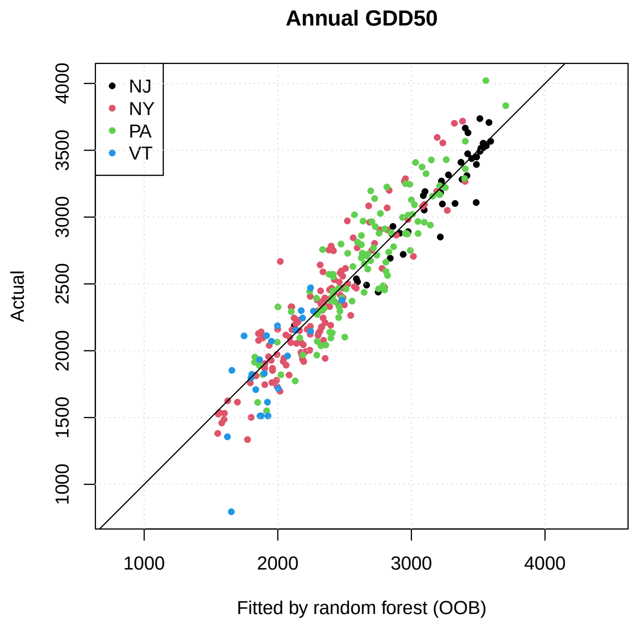
```
summary(rf.resid <- ne.m$ANN_GDD50 - rf.fits)

##       Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -418.9161  -64.2898   -7.4186   -0.8869   50.9118  292.6357

(rf.me <- mean(rf.resid))

## [1] -0.8869138

(rf.rmse <- sqrt(mean(rf.resid^2)/length(rf.resid)))

## [1] 5.67573
```

The fits are very close. However, this is not a good measure of the prediction accuracy. For that we use out-of-bag (OOB) **cross-validation**.

OOB predictions are automatically computed with the `ranger` function, at the same time it builds the forest. Each point is predicted as the average of the RF predictions for those regression trees where that point was *not* included in the “bag”.

Task 82 : Plot the actual vs. out-of-bag validation values from the random forest model. Compute the mean error (bias, ME) and the root mean squared error (RMSE). •

```
summary(rf.oob <- m.rf$predictions)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1550    2140    2456    2520    2862    3706

plot(ne.m$ANN_GDD50 ~ rf.oob,
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by random forest (OOB)",
     ylab="Actual",
     main="Annual GDD50")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid()
abline(0,1)
```



```
summary(rf.oob.resid <- ne.m$ANN_GDD50 - rf.oob)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -857.065 -125.679  -18.347   -2.488  115.325  649.654

(rf.oob.me <- mean(rf.oob.resid))

## [1] -2.488285

(rf.oob.rmse <- sqrt(mean(rf.oob.resid^2)/length(rf.oob.resid)))

## [1] 11.59619
```

This is a realistic view of the prediction accuracy. It is much more scattered around the 1:1 line than the calibration fit. Note in particular the high negative residual for Mt. Mansfield (VT); this is the blue point at the bottom of the plot.

Task 83 : Compare the ME and RMSE for the model fits and out-of-bag residuals. •

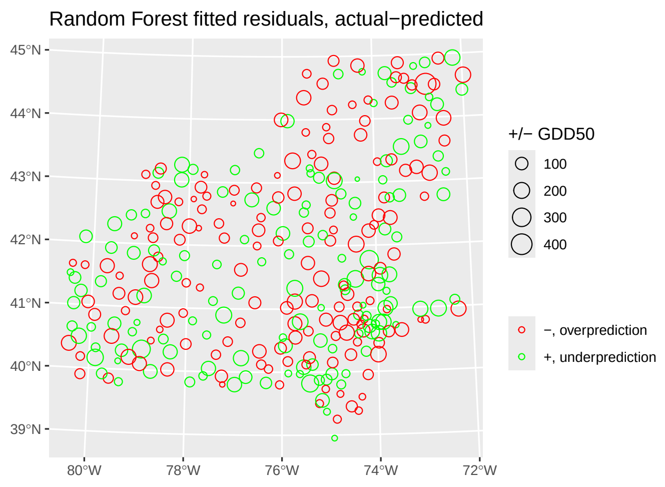
```
(rf.oob.me/rf.me)

## [1] 2.805555

(rf.oob.rmse/rf.rmse)

## [1] 2.043119
```

Q37 : *How does the OOB RMSE compare to the RMSE from the model fits at the training points? Which is more realistic as a measure of prediction accuracy?* *Jump to A37* •

Task 84 : Extract the RF model residuals, summarize them, and show them as a bubble plot. •

The model residuals are computed from the fits at each point and the actual values; for this we use the fits, not the out-of-bag estimates.

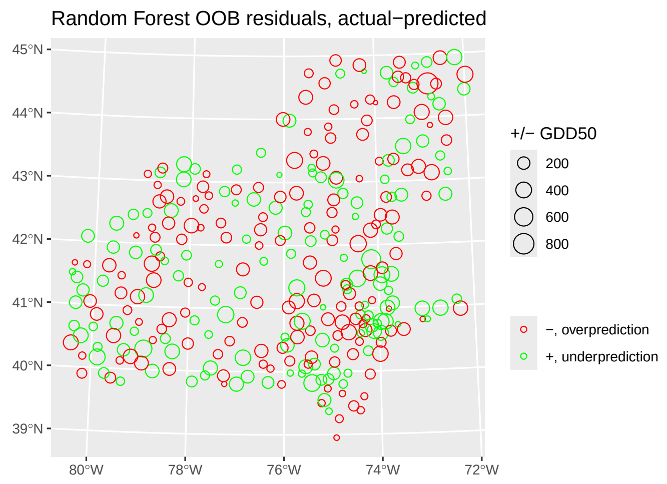
```
ne.m$rf.resid <- (ne.df$ANN_GDD50 - rf.fits)
summary(ne.m$rf.resid)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -418.9161  -64.2898   -7.4186   -0.8869   50.9118  292.6357

bubble.sf("ne.m", "rf.resid", "GDD50",
          "Random Forest fitted residuals, actual-predicted")
```



The residuals are fairly similar in range to the linear model. There are some very poorly-fit points. Notice that the mean residual is *not* zero, as in the least-squares fit of the linear model.

Task 85 : Compute the RF out-of-bag residuals and compare them to the RF model residuals. •

```
ne.m$rf.resid.oob <- (ne.df$ANN_GDD50 - rf.oob)
summary(ne.m$rf.resid.oob)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -857.065 -125.679  -18.347   -2.488  115.325  649.654

bubble.sf("ne.m", "rf.resid.oob", "GDD50",
          "Random Forest OOB residuals, actual-predicted")
```



Compare the fitted and OOB residuals:

```
summary(ne.m$rf.resid.oob/ne.m$rf.resid)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -121.909    1.835    2.008    1.441    2.195   12.600
```

As seen in the 1:1 plots and in the bubble plot legenfs, the out-of-bag residuals are much larger: The mean ratio of the two is 1.44.

Task 86 : List the eight worst-fit points, sorted by their absolute residuals, along with the residual from the GLS fit linear model. •

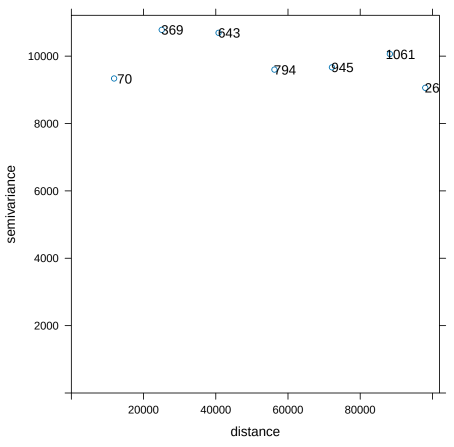
```
(ix <- order(abs(ne.m$rf.resid.oob), decreasing=TRUE)[1:8])

## [1] 293 118 220 232 115 226 146 278

ne.m[ix,c("STATE","STATION_NA","ELEVATION_",
                     "gls.resid", "rf.resid")]

## Simple feature collection with 8 features and 5 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -247918.1 ymin: -297715.7 xmax: 252835.5 ymax: 230248.4
## Projected CRS: +proj=aea +lat_0=42.5 +lat_1=39
##          +lat_2=44 +lon_0=-76 +ellps=WGS84 +units=m
##      STATE               STATION_NA ELEVATION_  gls.resid  rf.resid
## 4716    VT          MOUNT MANSFIELD       3950  -13.33249 -418.9161
## 3092    NY              MOHONK LAKE       1245  439.87959  292.6357
## 3830    PA                JOHNSTOWN       1214  653.74848  274.8954
## 3842    PA              MARCUS HOOK         10  411.03014  240.0095
## 3089    NY          MIDDLETOWN 2 NW        700  413.05043  225.0110
## 3836    PA         LAURELTON CENTER        800  392.09110  200.7551
## 3120    NY           SLIDE MOUNTAIN       2650 -354.36183 -195.9762
## 3888    PA WILKES BRE SCTN AP AVOCA        930  288.51349  199.7027
##                    geometry
## 4716   POINT (252835.5 230248.4)
## 3092  POINT (153668.2 -79515.15)
## 3830 POINT (-247918.1 -237035.8)
## 3842   POINT (49632.7 -297715.7)
## 3089  POINT (129349.2 -113350.6)
## 3836 POINT (-102705.5 -177146.3)
## 3120  POINT (130736.3 -52170.55)
## 3888    POINT (22581.52 -130037)

names(ne.m)

##  [1] "STATION_ID"         "STATE"              "STATION_NA"
##  [4] "LATITUDE_D"         "LONGITUDE_"         "ELEVATION_"
##  [7] "OID_"               "COOP_ID"            "STATE_1"
## [10] "STN_NAME"           "LAT_DD"             "LONG_DD"
## [13] "ELEV_FT"            "JAN_GDD50"          "FEB_GDD50"
## [16] "MAR_GDD50"          "APR_GDD50"          "MAY_GDD50"
## [19] "JUN_GDD50"          "JUL_GDD50"          "AUG_GDD50"
## [22] "SEP_GDD50"          "OCT_GDD50"          "NOV_GDD50"
## [25] "DEC_GDD50"          "ANN_GDD50"          "geometry"
## [28] "dist.lakes"         "dist.coast"         "mrvbf"
## [31] "tri3"               "pop15"              "pop2pt5"
## [34] "ols.resid"          "gls.resid"          "diff.gls.ols.resid"
## [37] "N"                  "resid.m.g.xy"       "rf.resid"
## [40] "rf.resid.oob"
```

The RF almost always comes closer than the GLS regression to these worse-fit points, because it is only using fairly similar points, in terms of the predictors (N, E, elevation) to predict, and does not try to fit a regional trend, which must consider all the calibration points.

However, the lowest actual GDD value (Mt. Mansfield in VT) is badly under-predicted by the random forest model. This is because it is so unlike any other point, since its elevstion is so much higher than the others. In the linear model this has strong leverage (highest elevation by far, well to the North) and is thus closely fit. The other poorly-predicted stations are also “unusual” in their covariate-space neighbourhood.

Task 87 : Compute and display an empirical variogram of the RF model residuals. •

```
v.rf <- variogram(rf.resid ~ 1, locations=ne.m, cutoff=100000, width=16000)
plot(v.rf, pl=T)
```



Q38 : *Do the residuals have spatial structure? This depends again on each run of the model; your results will look different from the ones presented here.* *Jump to A38* •

Task 88 : Compare the statistics of the regression model residuals. •

```
summary(ne.m$ols.resid)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -532.764 -153.151   -7.838    0.000  155.435  641.758

summary(ne.m$rf.resid)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -418.9161  -64.2898   -7.4186   -0.8869   50.9118  292.6357

summary(ne.m$gls.resid)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -514.041 -149.624   -3.980    8.496  160.856  653.749

sd(ne.m$ols.resid)

## [1] 210.8974

sd(ne.m$rf.resid)

## [1] 99.28129

sd(ne.m$gls.resid)

## [1] 211.0888
```

The statistics are similar; there is no clear “winner” Note that the GLS and RF models are not unbiased – the mean residual is not zero.

10.2.2 Random Forest prediction over the study area

We can use the RF model to predict over the study area, as we did in §6 for the OLS and GLS models.

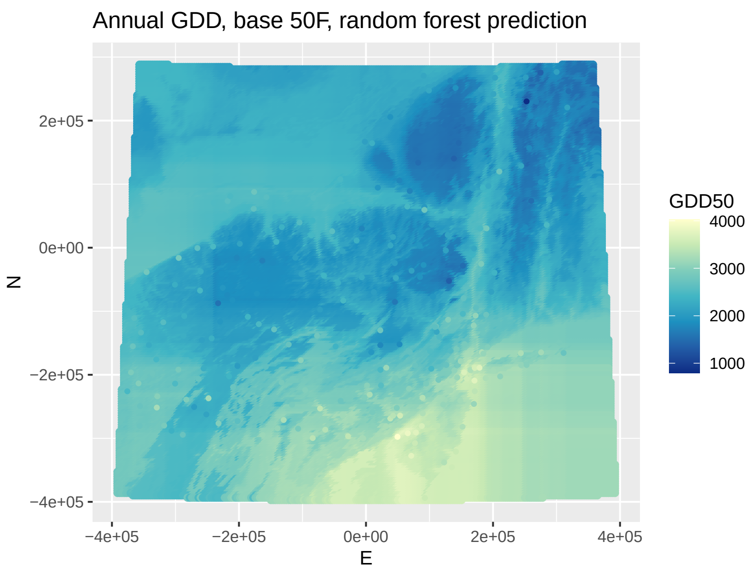
Task 89 : Predict over the grid with the RF model, add the results to the dataframe, and summarize them. •

```
dem.ne.m.df$pred.rf  <- predict(m.rf, data=dem.ne.m.df)$prediction
summary(dem.ne.m.df$pred.rf)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1359    2069    2382    2458    2827    3770
```

Task 90 : Display the RF surface. •

We also add the point set in the same colour scheme; if the point is visible it means the residual (lack of fit) is large.

```
display.prediction.map("pred.rf",
                       "Annual GDD, base 50F, random forest prediction",
                       "GDD50")
```



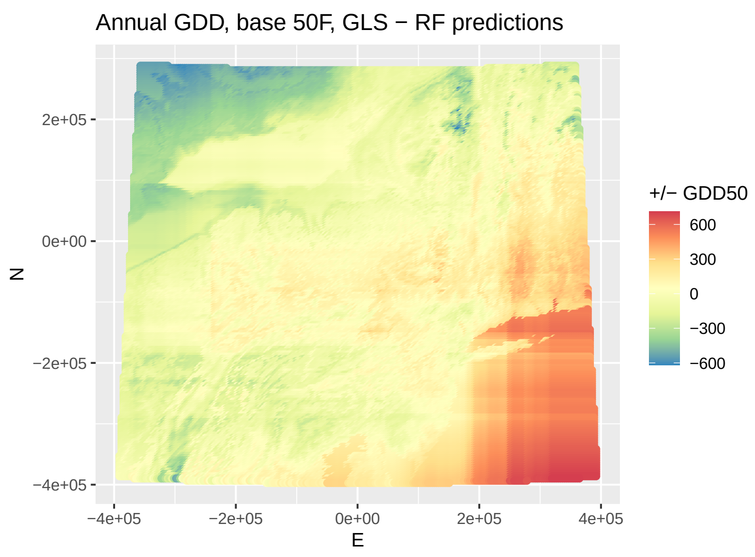
Q39 : *What is the difference in the spatial pattern between the RF surface and the RK-GLS surface?* *Jump to A39* •

Q40 : *Why is there only one predicted value in most of Lake Erie, another in the north-most part of Lake Erie, Lake Ontario, and the adjacent areas of Ontario (Canada)?* *Jump to A40* •

Task 91 : Compute the differences between the GLS and RF predictions, add them to the data frame, and display them. •

```
summary(dem.ne.m.df$diff.gls.rf <-
            dem.ne.m.df$pred.gls - dem.ne.m.df$pred.rf)

##      Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -617.782 -121.130   -7.530    8.498  115.189  709.333
```

```
display.difference.map("diff.gls.rf",
                       "Annual GDD, base 50F, GLS - RF predictions",
                       "+/- GDD50")
```



Q41 : *Why are the largest discrepancies in the east, especially in Connecticut and Massachusetts, and in the northwest (Ontario)?* Jump to A41 •

Q42 : *Why is the border of PA with OH on the west visible in this difference map?* Jump to A42 •

Q43 : *Why are there positive differences (GLS-RK greater than RF) in the PA mountains and in the Catskills and Taconics in NY? Why does this not occur in the Adirondacks?* Jump to A43 •

Q44 : *What do you conclude about the geographical area of applicability of the random forest prediction?* Jump to A44 •

10.3 Tuning data-driven models

Data-driven models have parameters that control their behaviour and can significantly affect their predictive power.

For example, regression trees (§10.1) can be adjusted by the minimum number of observations which can be considered for a split (using the `minsplit` argument) and the minimum value of a complexity parameter (using the `cp` argument).

Random forests (§10.2) can also be controlled by the minimum number of observations in a terminal node (optional argument `nodesize`), as well as the number of predictors to compare at each split (optional argument `mtry`). In the `randomForest` function these have default values of 5 and 1/3 of the number of predictors, respectively. These can have a large influence on the resulting forest. Too small terminal nodes will result in over-fit trees, too large in poorer fits. Too many predictors tested at each split will not allow less powerful predictors into the forest; too few will result in many poorly-fitted trees.

> **Note:** The number of trees `ntree` also has an influence on the random forest model. Too few trees will cause repeated model fits to be too variable, too many wastes computing time. This parameter is not optimized as such, a large value is used and the graph of out-of-bag RMSE vs. number of trees is examined to select an appropriate value.

So how to decide on these parameter values? A powerful method is to examine a space of possible values, and select the best values by the performance of each model. The performance is often evaluated by repeated **cross-validation**:

1. For each combination of parameters to be optimized:
    (a) Split the dataset into some disjunct subsets, for example 10, by random sampling.
    (b) For each subset:
        i. Fit the model with the selected parameters on all but one of the subsets.
        ii. Predict at the remaining subset, i.e., the one not used for model building, with the fitted model.
        iii. Compute the goodness-of-fit statistics of fitting, typically the root mean square error (RMSE) of prediction and the squared correlation coefficient between the actual and fitted values, i.e., R^2 against a 1:1 line.
    (c) Average the statistics for the disjunct subsets.
2. Search the table of results for the best results: lowest RMSE and highest R^2.

These values are then the recommended ones to fit a final model on all data.

The `caret` "**C**lassification **A**nd **RE**gression **T**raining" package [11] implements this procedure.

```
require(caret)
```

This package is highly flexible and can be used to optimize 239 kinds of data-driven models, including the ones we use in this tutorial. To see the list of possible models:

```
length(names(getModelInfo()))

## [1] 239
```

```
head(names(getModelInfo()),24)

##  [1] "ada"             "AdaBag"          "AdaBoost.M1"
##  [4] "adaboost"        "amdai"           "ANFIS"
##  [7] "avNNet"          "awnb"            "awtan"
## [10] "bag"             "bagEarth"        "bagEarthGCV"
## [13] "bagFDA"          "bagFDAGCV"       "bam"
## [16] "bartMachine"     "bayesglm"        "binda"
## [19] "blackboost"      "blasso"          "blassoAveraged"
## [22] "bridge"          "brnn"            "BstLm"
```

Many more details are given in the `caret` package on-line book[20]. The package is highly adaptable, and before applying it in a production environment, carefully read the documentation. Here we just present a simple case.

The principal function of the `caret` package is `train`, which implements the cross-validation procedure and reports the optimum combination of parameters. To use this, we have to set up the following arguments:

1. a design matrix `x` of predictor values for each observation;
2. a response vector `y` of known values for each observation;
3. the `tuneGrid` argument with the names and range of values for the parameters to be tuned;
4. the `method` argument, which says which data-driven method to optimize;
5. the `trControl` argument, which specifies the optimization criterion, using the `trainControl` function.

Task 92 : Optimize a random forest model. •

The `caret` package supports random forest models implemented in the `ranger` package.

We create a matrix with all combinations of the two parameters, with the `expand.grid` function. The columns are named the same as the tuning parameters, which we obtain with the `getModelInfo` function:

```
getModelInfo("ranger")$ranger$parameters

##       parameter     class                         label
## 1          mtry   numeric #Randomly Selected Predictors
## 2     splitrule character                Splitting Rule
## 3 min.node.size   numeric            Minimal Node Size
```

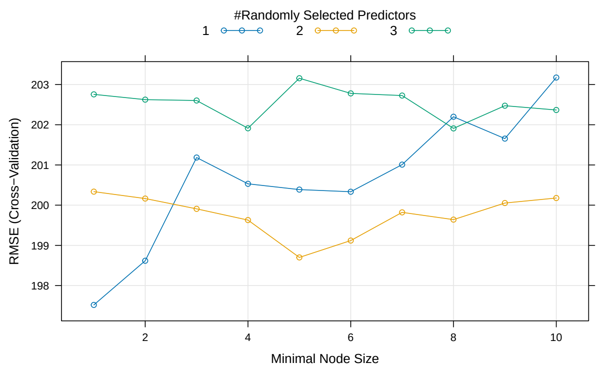
For the meaning of each of these, see `?ranger` and the explanations in the journal article. Here we only have three predictors, so `mtry` can vary from 1 to 3. The splitting rule is set to `"variance"`, the normal criterion for regression trees: the split is where the between-class variance is maximized and the within-class variance is minimized. The minimum node size `min.node.size` defaults to 5, we try smaller (more complex trees in the forest) and larger (less complex) values.

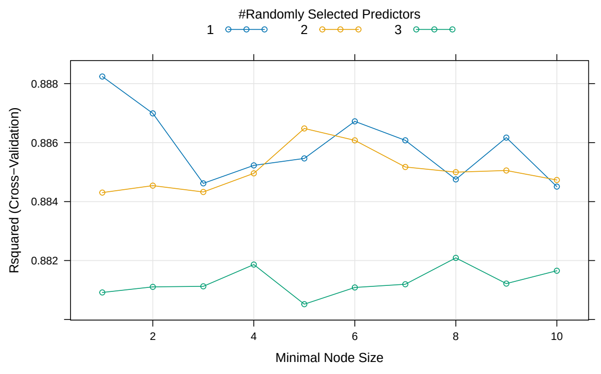
[20] http://topepo.github.io/caret/index.html

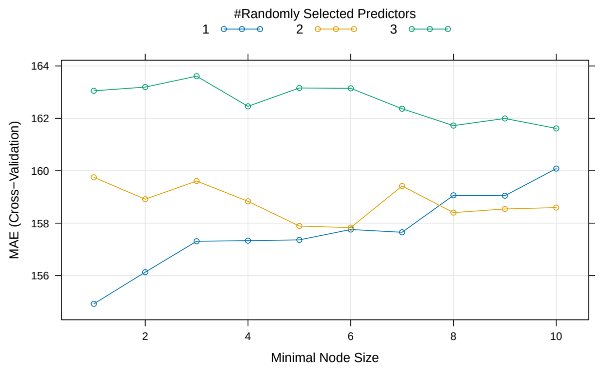
```
dim(preds <- ne.df[, c("E", "N", "ELEVATION_")])

## [1] 305   3

length(response <- ne.df[, "ANN_GDD50"])

## [1] 305

system.time(
   ranger.tune <- train(x = preds, y = response, method="ranger",
                 tuneGrid = expand.grid(.mtry = 1:3,
                                        .splitrule = "variance",
                                        .min.node.size = 1:10),
                 trControl = trainControl(method = 'cv'))
)

##    user  system elapsed
##  20.124   2.303   4.964
```

```
print(ranger.tune)

## Random Forest
##
## 305 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 274, 274, 273, 274, 276, 275, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  RMSE      Rsquared   MAE
##   1      1             197.5190  0.8882416  154.9209
##   1      2             198.6177  0.8869913  156.1293
##   1      3             201.1845  0.8846177  157.3081
##   1      4             200.5310  0.8852271  157.3311
##   1      5             200.3875  0.8854635  157.3609
##   1      6             200.3338  0.8867208  157.7577
##   1      7             201.0084  0.8860766  157.6522
##   1      8             202.1988  0.8847521  159.0614
##   1      9             201.6555  0.8861695  159.0470
##   1     10             203.1736  0.8845083  160.0833
##   2      1             200.3364  0.8843045  159.7509
##   2      2             200.1645  0.8845410  158.9134
##   2      3             199.9086  0.8843256  159.6087
##   2      4             199.6293  0.8849575  158.8335
##   2      5             198.7004  0.8864801  157.8884
##   2      6             199.1197  0.8860764  157.8304
##   2      7             199.8209  0.8851734  159.4180
##   2      8             199.6407  0.8849978  158.4018
##   2      9             200.0531  0.8850541  158.5416
##   2     10             200.1777  0.8847316  158.5939
##   3      1             202.7557  0.8809174  163.0506
##   3      2             202.6245  0.8811076  163.1930
##   3      3             202.6026  0.8811257  163.6100
##   3      4             201.9123  0.8818638  162.4594
##   3      5             203.1566  0.8805201  163.1596
##   3      6             202.7799  0.8810873  163.1450
##   3      7             202.7258  0.8811961  162.3679
##   3      8             201.9091  0.8820909  161.7236
##   3      9             202.4731  0.8812187  161.9962
##   3     10             202.3668  0.8816545  161.6169
##
## Tuning parameter 'splitrule' was held constant at a value of variance
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 1, splitrule
##  = variance and min.node.size = 1.

names(ranger.tune$result)
```

```
## [1] "mtry"          "splitrule"     "min.node.size" "RMSE"
## [5] "Rsquared"      "MAE"           "RMSESD"        "RsquaredSD"
## [9] "MAESD"

ix <- which.min(ranger.tune$result$RMSE)
ranger.tune$result[ix, c(1,3,4)]

##   mtry min.node.size    RMSE
## 1    1             1 197.519

ix <- which.max(ranger.tune$result$Rsquared)
ranger.tune$result[ix, c(1,3,5)]

##   mtry min.node.size  Rsquared
## 1    1             1 0.8882416

ix <- which.min(ranger.tune$result$MAE)
ranger.tune$result[ix, c(1,3,6)]

##   mtry min.node.size      MAE
## 1    1             1 154.9209

plot.train(ranger.tune, metric="RMSE")
plot.train(ranger.tune, metric="Rsquared")
plot.train(ranger.tune, metric="MAE")
```







In this case the three optimization criteria give different recommendations. Here it's clear that for `mtry` that 2 is optimal for all of RMSE, MAE and R^2. This is different from the default $3/3 = 1$. For all of these `min.node.size`=6 is best.

Each run of `train` will give different results, because of the different random splits into test and train sets.

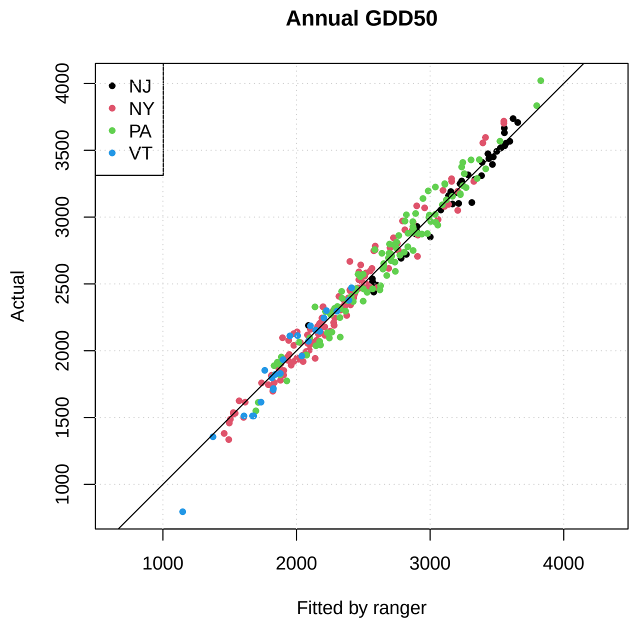
Once we've selected an optimal combination we fit a final model.

Task 93 : Build an optimal model and display its fit to known points. •

The `ranger` function requires a formula argument (as does `randomForest`):

```
(ranger.rf <- ranger(ANN_GDD50 ~ N + E + ELEVATION_, data=ne.df,
                    mtry=2, min.node.size=5))

## Ranger result
##
## Call:
##  ranger(ANN_GDD50 ~ N + E + ELEVATION_, data = ne.df, mtry = 2,      min.node.size = 5)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      305
## Number of independent variables:  3
## Mtry:                             2
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       40600.49
## R squared (OOB):                  0.8767443
```

```
summary(ranger.fits <- predict(ranger.rf, data=ne.df)$predictions)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1148    2104    2461    2519    2902    3828

plot(ne.df$ANN_GDD50 ~ ranger.fits,
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by ranger",
     ylab="Actual",
     main="Annual GDD50")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid(); abline(0,1)
```

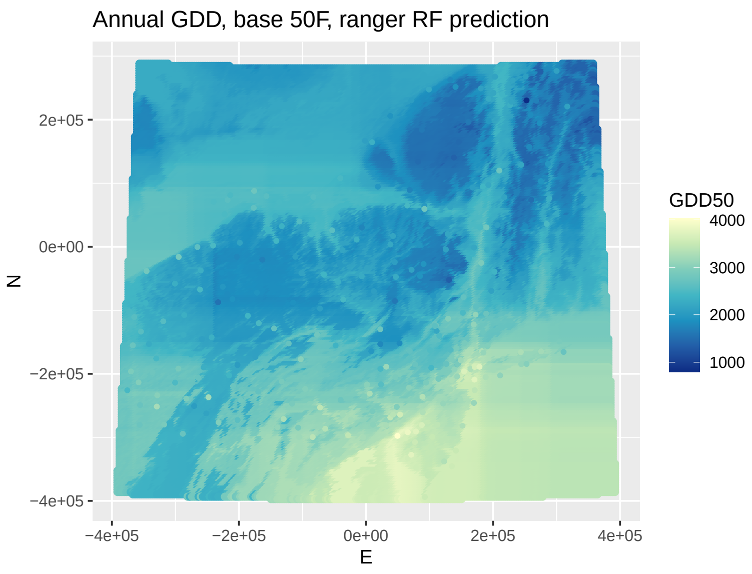


Task 94 : Predict over the grid with the optimized `ranger` model, add the results to the dataframe, and summarize them. •

```
summary(dem.ne.m.df$pred.ranger <-
            predict(ranger.rf, data=dem.ne.m.df)$predictions)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1210    2014    2320    2444    2837    3817
```

Task 95 : Display the map. •

```
display.prediction.map("pred.ranger",
                       "Annual GDD, base 50F, ranger RF prediction",
                       "GDD50")
```



10.4 Cubist

A popular data-driven model is Cubist, derived from the C4.5 models [20] but extensively modified and implemented as R package `Cubist`, an R port of the Cubist GPL C code released by RuleQuest[21].

```
library(Cubist)
```

A good introduction to Cubist is by Kuhn and Johnson [12, §8.7] as well as the vignette distributed with the `Cubist` package:

```
vignette("cubist")
```

Cubist is similar to regression trees, but instead of single values at leaves it creates a multivariate linear regression for the cases in the leaf. As the vignette explains:

> "A tree is grown where the terminal leaves contain linear regression models. These models are based on the predictors used in previous splits. Also, there are intermediate linear models at each step of the tree. A prediction is made using the linear regression model at the terminal node of the tree, but is 'smoothed' by taking into account the prediction from the linear model in the previous node of the tree (which also occurs recursively up the tree). The tree is reduced to a set of rules, which initially are paths from the top of the tree to the bottom. Rules are eliminated via pruning and/or combined for simplification."

The advantage over regression trees is that the predictions are continous,

[21] http://rulequest.com/cubist-info.html

not discrete values equal to the number of leaves in the regression tree. The advantage over random forests is that the model can be interpreted, to a certain extent. A disadvantage of Cubist is that its algorithm is not easy to understand; however its results are generally quite good.

Cubist models can be improved in two ways: (1) with "committees" of models and (2) by adjusting predictions based on nearest neighbours in feature (predictor) space.

committees

Committees are a form of boosting. A set of model trees are built in sequence. The first tree is the standard Cubist best tree, using the original data in the training set. Subsequent trees are built from adjusted versions to the training set. If the previous Cubist tree over(under)-predicted a value, the response is adjusted down(up)ward for the next model, before it is fit. The final prediction is the average of the predictions from each model tree. The idea here is that the predictions by the sequence of trees vary around the "true" value.

nearest neighbours

This prediction from the set of trees can then be adjusted using the values of some number of **nearest neighbours** in feature space. The idea here is that the overall model fits all the training data, but locally we may have some unknown factor that operates only in a local region of feature space, so if we have data from that region, we should give it more weight. Specifically, if the single model or committee predict a value $\widehat{y}$, the adjusted prediction based on the K nearest-neighbours in feature space is:

$$\widehat{y}' = \frac{1}{K} \sum_{i=1}^{K} w_i \left[t_i + (\widehat{y} - \widehat{t_i}) \right] \tag{22}$$

where t_i is the actual value of the neighbour, $\widehat{t_i}$ is its value predicted by the model tree(s), and w_i is the weight given to this neighbour for the adjustment, based on its distance D_i from the target point. These are computed as $w_i = 1/(D_i + 0.5)$ and normalized to sum to one.

In addition, to guard against using neighbours that are too far from the target point, the proposed set of neighbors are filtered based on the average pairwise distance of data points in the training set. In our case study, it is likely that Mt. Mansfield (VT) is too far from any other observation points to have any neighbours.

Obviously, the question is how many committees and neighbours to use, if any. The `caret` package can be used to tune these parameters (S10.3.

Task 96 : Determine the optimum number of committees and neighbours for a Cubist model to predict the growing degree days from Northing, Easting, and Elevation. •

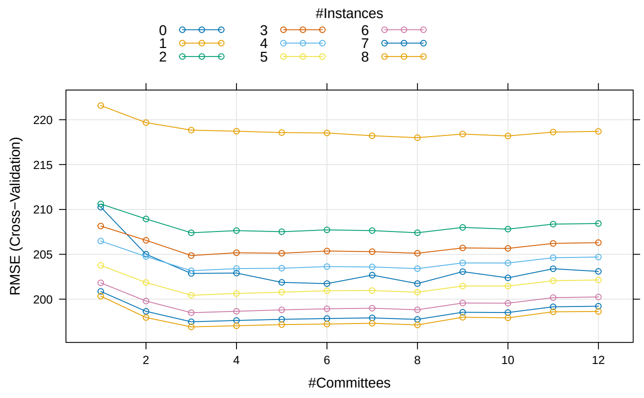
We do this with the `train` function, specifying the `method` argument as `'cubist'`.

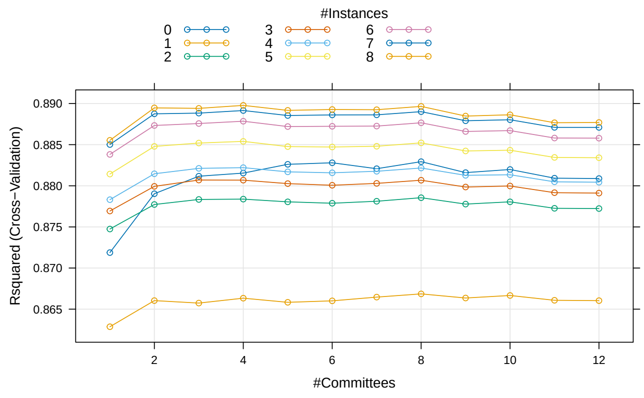
```
all.preds <- ne.df[, c("N", "E", "ELEVATION_")]
all.resp <- ne.df[ , "ANN_GDD50"]
```

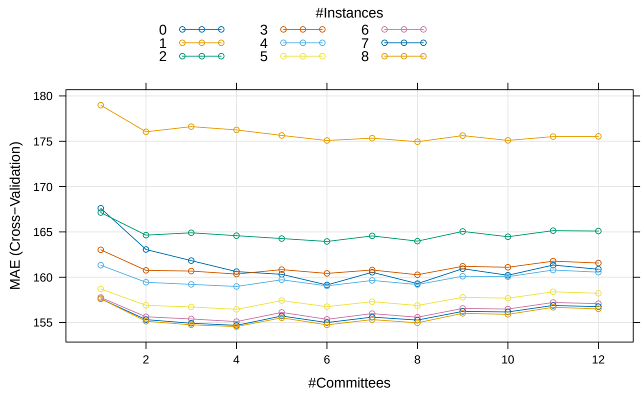
```
system.time(
       cubist.tune <- train(x = all.preds, y = all.resp, "cubist",
                     tuneGrid = expand.grid(.committees = 1:12,
                                            .neighbors = 0:8),
                     trControl = trainControl(method = 'cv'))
       )

##    user  system elapsed
##   2.515   0.011   2.531
```

```
print(cubist.tune)

## Cubist
##
## 305 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 273, 275, 273, 275, 273, 273, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##    1          0          210.2573  0.8718786  167.6044
##    1          1          221.5769  0.8628628  178.9769
##    1          2          210.6112  0.8747445  167.1347
##    1          3          208.1492  0.8769341  163.0249
##    1          4          206.4790  0.8783113  161.3244
##    1          5          203.7662  0.8814272  158.7046
##    1          6          201.8184  0.8838142  157.7601
##    1          7          200.8640  0.8850150  157.6147
##    1          8          200.3459  0.8855277  157.6202
##    2          0          205.0058  0.8790333  163.0586
##    2          1          219.6760  0.8660398  176.0418
##    2          2          208.9411  0.8777240  164.6446
##    2          3          206.5594  0.8799319  160.7576
##    2          4          204.7528  0.8814618  159.4444
##    2          5          201.8613  0.8847895  156.9040
##    2          6          199.7938  0.8873324  155.6350
##    2          7          198.6461  0.8887497  155.3224
##    2          8          197.9578  0.8894739  155.1466
##    3          0          202.8691  0.8811647  161.8315
##    3          1          218.8401  0.8657395  176.6138
##    3          2          207.4001  0.8783374  164.9011
##    3          3          204.8648  0.8806994  160.6794
##    3          4          203.1704  0.8821180  159.2110
##    3          5          200.4419  0.8851970  156.7189
##    3          6          198.4931  0.8875712  155.3964
##    3          7          197.4822  0.8888352  154.9244
##    3          8          196.9101  0.8894136  154.7508
##    4          0          202.9098  0.8815437  160.6203
##    4          1          218.7164  0.8663360  176.2514
##    4          2          207.6422  0.8783791  164.5841
##    4          3          205.1735  0.8806853  160.3500
##    4          4          203.4089  0.8822065  158.9820
##    4          5          200.6324  0.8853992  156.4587
##    4          6          198.6518  0.8878509  155.1050
##    4          7          197.6268  0.8891508  154.6902
##    4          8          197.0396  0.8897766  154.5571
##    5          0          201.8732  0.8826101  160.3219
##    5          1          218.5642  0.8658323  175.6422
##    5          2          207.5171  0.8780473  164.2672
##    5          3          205.1187  0.8802642  160.8382
##    5          4          203.4495  0.8816942  159.7198
##    5          5          200.7792  0.8847665  157.4239
##    5          6          198.8099  0.8872013  156.1177
##    5          7          197.7537  0.8885439  155.7233
##    5          8          197.1678  0.8891688  155.5246
##    6          0          201.7332  0.8827977  159.1467
##    6           1          218.5206  0.8660110  175.0839
##    6           2          207.7230  0.8778793  163.9450
##    6           3          205.3653  0.8800645  160.4224
##    6           4          203.6321  0.8815789  159.0473
##    6           5          200.9397  0.8847180  156.7534
##    6           6          198.9269  0.8872355  155.3609
##    6           7          197.8466  0.8886171  155.0002
##    6           8          197.2299  0.8892757  154.7465
##    7           0          202.6818  0.8820772  160.5479
##    7           1          218.2121  0.8664657  175.3425
##    7           2          207.6407  0.8781200  164.5592
##    7           3          205.2958  0.8802922  160.7983
##    7           4          203.5931  0.8817662  159.6460
##    7           5          200.9625  0.8848071  157.3061
##    7           6          198.9925  0.8872597  155.9881
##    7           7          197.9128  0.8886249  155.6001
##    7           8          197.3177  0.8892481  155.3269
##    8           0          201.7402  0.8829329  159.2934
##    8           1          217.9982  0.8668571  174.9493
##    8           2          207.4035  0.8785513  163.9813
##    8           3          205.1164  0.8806774  160.2740
##    8           4          203.4035  0.8821619  159.1970
##    8           5          200.7747  0.8852188  156.8818
##    8           6          198.8194  0.8876589  155.5882
##    8           7          197.7478  0.8890136  155.2713
##    8           8          197.1300  0.8896540  154.9691
##    9           0          203.0686  0.8816014  160.9398
##    9           1          218.3989  0.8663564  175.6231
##    9           2          207.9935  0.8777747  165.0536
##    9           3          205.7117  0.8798509  161.2064
##    9           4          204.0382  0.8812716  160.1078
##    9           5          201.4694  0.8842362  157.7847
##    9           6          199.5703  0.8866051  156.5587
##    9           7          198.5427  0.8879052  156.2332
##    9           8          197.9861  0.8884772  156.0188
##   10           0          202.3774  0.8819816  160.2178
##   10           1          218.1874  0.8666635  175.0949
##   10           2          207.8090  0.8780487  164.4668
##   10           3          205.6584  0.8799742  161.0972
##   10           4          204.0275  0.8813430  160.0624
##   10           5          201.4581  0.8843262  157.6882
##   10           6          199.5502  0.8867127  156.4890
##   10           7          198.5114  0.8880297  156.1709
##   10           8          197.9246  0.8886317  155.9063
##   11           0          203.3921  0.8809251  161.3420
##   11           1          218.6205  0.8660776  175.5133
##   11           2          208.3638  0.8772632  165.1390
##   11           3          206.2135  0.8791630  161.7741
##   11           4          204.6161  0.8804841  160.7965
##   11           5          202.0548  0.8834497  158.3949
##   11           6          200.1707  0.8858075  157.2140
##   11           7          199.1492  0.8871015  156.8858
##   11           8          198.5938  0.8876680  156.6825
##   12           0          203.0938  0.8808801  160.8818
##   12           1          218.6963  0.8660353  175.5390
##   12           2          208.4334  0.8772327  165.0972
##   12           3          206.3024  0.8791101  161.5688
##   12           4          204.6967  0.8804346  160.5577
##   12           5          202.1413  0.8834106  158.2277
##   12           6          200.2471  0.8857904  157.0856
##   12           7          199.2170  0.8870987  156.7516
##   12           8          198.6299  0.8876975  156.5114
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 3 and
##  neighbors = 8.
```

```r
plot(cubist.tune, metric="RMSE")
plot(cubist.tune, metric="Rsquared")
plot(cubist.tune, metric="MAE")
```







Your results will be different, because of the randomness in the splits from `method`. So, your choice of optimal values may also be different than those presented here.

In this case committees improve the model. Although the best result is with 10 committees, looking at the graphs we can see that 6 gives almost equally good results, so we prefer the simpler model. Neighbours definitely do improve the model. Using one neighbour (the closest in feature space) makes the model much worse – too much fine adjustment to the training set. Using two to seven neighbours gives improvement, using eight is only slightly better. This shows that the overall model can benefit by local adjustment.

Task 97 : Built an "optimum" Cubist model. •

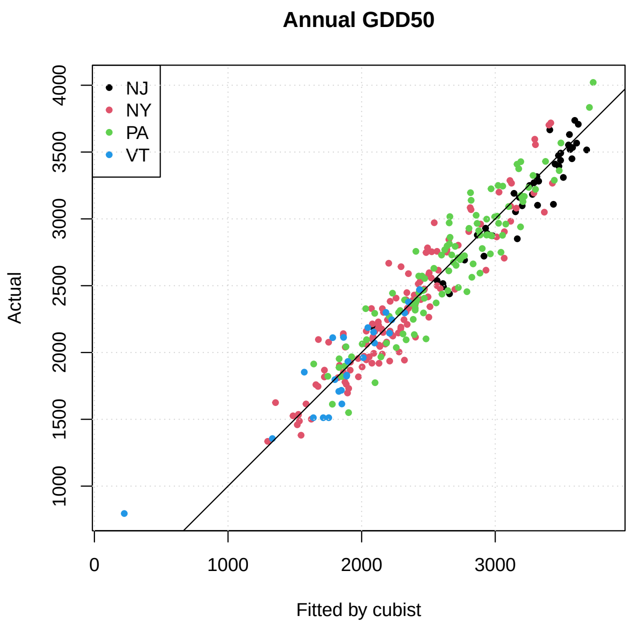
The model is fit with the `cubist` function; the optimal number of committees is specified at model building with the `committees` argument. Adjustment by neighbours is done at prediction, because only then do we know which point we are predicting, hence which are its nearest neighbours. This is specified by the `neighbors` to the `predict` function of the `Cubist` package.

```
require(Cubist)
c.model <- cubist(x = all.preds, y = all.resp, committees=6)
summary(c.model)

##
## Call:
## cubist.default(x = all.preds, y = all.resp, committees = 6)
##
##
## Cubist [Release 2.07 GPL Edition]  Sat May 11 11:01:09 2024
## ---------------------------------
##
##     Target attribute `outcome'
##
## Read 305 cases (4 attributes) from undefined.data
##
## Model 1:
##
##   Rule 1/1: [109 cases, mean 2114.8, range 795 to 2845, est err 151.9]
##
##     if
##  N > -20368.92
##     then
##  outcome = 2715.4 - 0.496 ELEVATION_ - 0.00105 N - 0.00068 E
##
##   Rule 1/2: [196 cases, mean 2741.5, range 1335 to 4021, est err 168.7]
##
##     if
##  N <= -20368.92
##     then
##  outcome = 2704 - 0.00264 N - 0.606 ELEVATION_ - 0.00021 E
##
## Model 2:
##
##   Rule 2/1: [305 cases, mean 2517.5, range 795 to 4021, est err 171.2]
##
##  outcome = 2826.9 - 0.599 ELEVATION_ - 0.00205 N - 0.00023 E
##
## Model 3:
##
##   Rule 3/1: [174 cases, mean 2171.9, range 795 to 3084, est err 172.4]
##
##     if
##  N > -145485.8
##     then
##  outcome = 2642.4 - 0.474 ELEVATION_ - 0.00077 E - 0.00045 N
##
##   Rule 3/2: [74 cases, mean 2729.1, range 1774 to 3428, est err 178.6]
##
##     if
##  N <= -145485.8
##  ELEVATION_ > 330
##     then
##  outcome = 1956.9 - 0.00513 N - 0.457 ELEVATION_ - 9e-05 E
##
##   Rule 3/3: [76 cases, mean 3090.3, range 2078 to 4021, est err 182.4]
##
##     if
##  ELEVATION_ <= 330
##     then
##  outcome = 3106.6 - 2.148 ELEVATION_ - 0.00188 N
##
## Model 4:
##
##   Rule 4/1: [305 cases, mean 2517.5, range 795 to 4021, est err 170.7]
##
##  outcome = 2842.7 - 0.613 ELEVATION_ - 0.00206 N - 0.0003 E
##
## Model 5:
##
##   Rule 5/1: [174 cases, mean 2171.9, range 795 to 3084, est err 174.1]
##
##     if
##  N > -145485.8
##     then
##  outcome = 2629.8 - 0.462 ELEVATION_ - 0.00071 E - 0.00043 N
##
##   Rule 5/2: [74 cases, mean 2729.1, range 1774 to 3428, est err 183.9]
##
##     if
##  N <= -145485.8
##  ELEVATION_ > 330
##     then
##  outcome = 1925.4 - 0.0052 N - 0.45 ELEVATION_ - 6e-05 E
##
##   Rule 5/3: [76 cases, mean 3090.3, range 2078 to 4021, est err 182.7]
##
##     if
##  ELEVATION_ <= 330
##     then
##  outcome = 3112.7 - 2.176 ELEVATION_ - 0.00185 N
##
## Model 6:
##
##   Rule 6/1: [305 cases, mean 2517.5, range 795 to 4021, est err 171.1]
##
##  outcome = 2854.2 - 0.623 ELEVATION_ - 0.00208 N - 0.00036 E
##
##
## Evaluation on training data (305 cases):
##
##     Average  |error|              169.9
##     Relative |error|               0.36
##     Correlation coefficient        0.93
##
##
##  Attribute usage:
##    Conds  Model
##
##     43%   100%    N
##     16%   100%    ELEVATION_
##            92%    E
##
##
## Time: 0.0 secs
```

Rule 1 splits at `N = -20368.92`, near the centre of the map. There is then a slightly different linear regression for the two halves. The elevation and Northing coefficients are both larger for the southern half. Rule 2 has no split, just a single linear model. It is not the same as the linear model fit in §4, because it is fit based on the values adjusted by Rule 1 predictions. Rule 3 again splits on Northing, but much further south, at `N = -145485.8`, and on low elevations, `330`.

Task 98 : Examine the fit of the Cubist model to the known points. •

```
# predictive accuracy
cubist.fits <- predict(c.model, newdata=all.preds,
                       neighbors=cubist.tune$bestTune$neighbors)
## Test set RMSE
sqrt(mean((cubist.fits - all.resp)^2))

## [1] 164.9617

cor(cubist.fits, all.resp)^2  # R^2

## [1] 0.918133

plot(ne.df$ANN_GDD50 ~ cubist.fits,
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by cubist",
     ylab="Actual",
     main="Annual GDD50")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid(); abline(0,1)
```

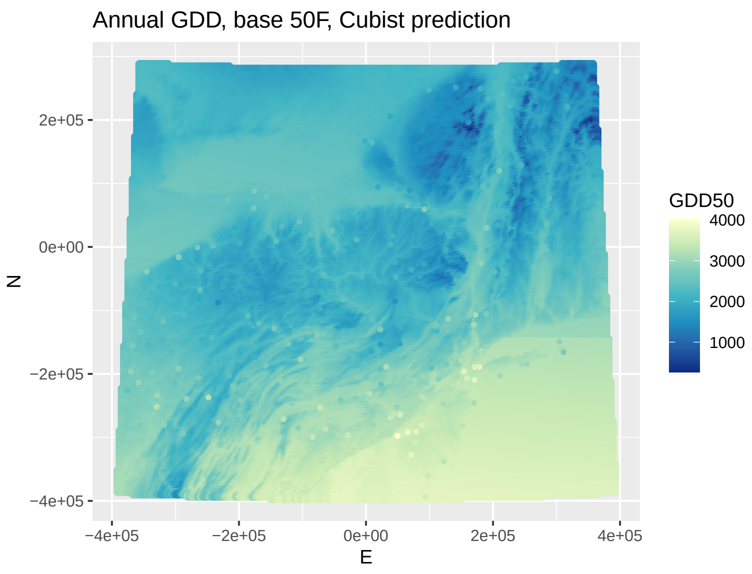


Notice that this Cubist model severely underpredicts the lowest GDD50 at Mount Mansfield (VT). This is because it is too far in elevation space from any other observation points to have any neighbours that could modify the linear regressions in the rule set, which reduce GDD50 with elevation.

Task 99 : Predict over the study area with the Cubist model and display the resulting map. •

```
summary(dem.ne.m.df$pred.cubist <-
            predict(c.model, newdata=dem.ne.m.df,
                              neighbours=cubist.tune$bestTune$neighbors))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   265.7  2040.0  2363.4  2461.9  2898.8  3748.3
```

```
display.prediction.map("pred.cubist",
                       "Annual GDD, base 50F, Cubist prediction",
                       "GDD50")
```



Although several of the rules split on Northing, that is not visible in this map, because the results of each rule are averaged.

10.5 Additional covariables

Machine learning models are generally applied to problems with large numbers of predictors. In the example above we only used three. To illustrate the more common situation, we add some additional predictors that might be related to agricultural climate.

- The two Great Lakes in this region (Erie and Ontario) may have a local climate effect, because of the high heat capacity of the lake water, which can extend the late summer into the early fall.
- The Atlantic Ocean has a local cooling effect along the shore, especially on Long Island.
- Local terrain may influence climate. For example, narrow valleys in the Finger Lakes regions are known as “frost pockets” and often have morning ground fogs in spring and early summer.
  - The **multiresolution index of valley bottom flatness** (MRVBF) [5] identifies valley bottoms based on their topographic signature as flat low-lying areas, at increasingly-broad scales, and combines these into a single index.

  – The **terrain ruggedness index** (TRI) [21] expresses heterogeneity. It is the sum change in elevation between a grid cell and its eight neighbours.

- Population density may affect local climate. Urban areas affect the local climate, typically making it warmer, whereas very sparsely-populated rural areas are typically cooler, with more precipitation as snow.

10.6 Models with the extended set of predictors

Now we investigate whether any of these covariables separately or in combination improve modelling and prediction of the regional climate, specifically the 30-year 1971-2000 average annual growing degree days, base 50°F.

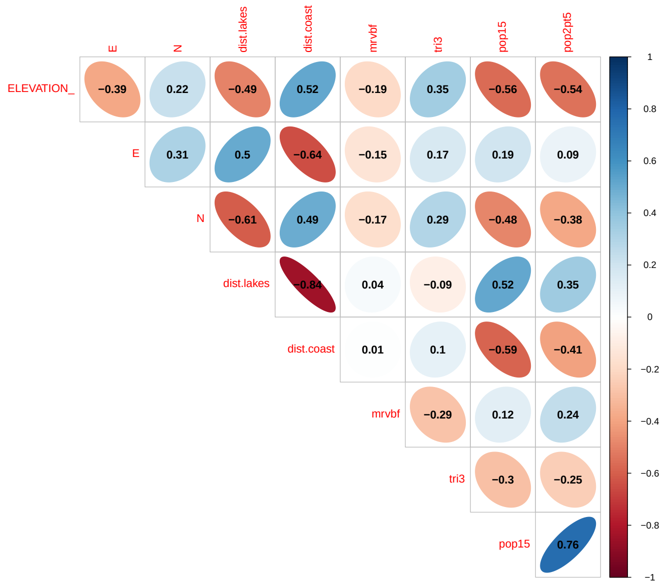
```
pred.field.names <- c("ELEVATION_", "E", "N",
                      "dist.lakes", "dist.coast",
                      "mrvbf", "tri3", "pop15", "pop2pt5")
pred.field.names.base <- c("ELEVATION_", "E", "N")
(model.formula <- paste("ANN_GDD50 ~",
                        paste0(pred.field.names, collapse=" + ")))

## [1] "ANN_GDD50 ~ ELEVATION_ + E + N + dist.lakes + dist.coast + mrvbf + tri3 + pop15 + pop2pt5"

(model.formula.base <- paste("ANN_GDD50 ~",
                             paste0(pred.field.names.base, collapse=" + ")))

## [1] "ANN_GDD50 ~ ELEVATION_ + E + N"

names(dem.ne.m.df)

##  [1] "E"                 "N"                  "ELEVATION_"
##  [4] "dist.lakes"        "dist.coast"         "mrvbf"
##  [7] "tri3"              "pop15"              "pop2pt5"
## [10] "pred.ols"          "pred.gls"           "diff.gls.ols"
## [13] "ok.gls.resid"      "ok.gls.resid.var"   "pred.rkgls"
## [16] "pred.ked"          "diff.rkgls.ked"     "pred.ked.nn"
## [19] "pred.ked.nn.sd"    "diff.ked"           "diff.okrk.ked.pred"
## [22] "pred.gam"          "pred.gam.se"        "diff.gls.gam"
## [25] "diff.rkgls.gam"    "pred.rt"            "pred.rf"
## [28] "diff.gls.rf"       "pred.ranger"        "pred.cubist"
```

10.6.1 Relation among predictors

First, investigate the relation among the predictors.

Pairwise correlation Examine the pairwise relation between predictors with Pearson (parametric) correlation.

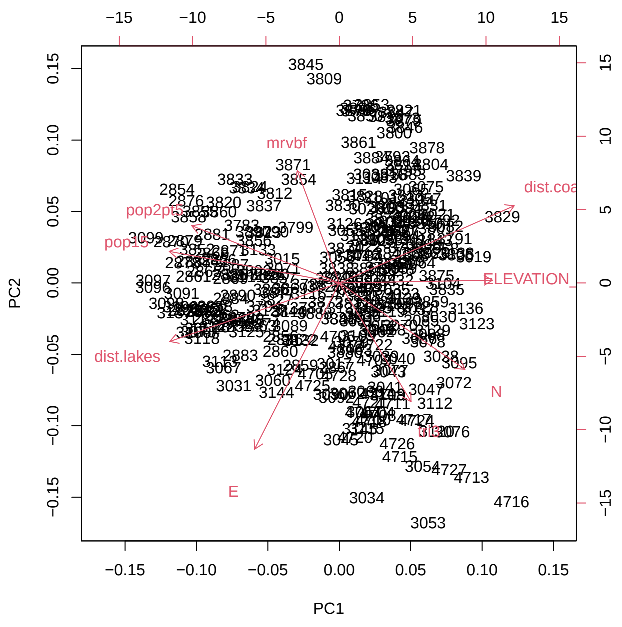
```
cor(ne.df[, pred.field.names])

##             ELEVATION_           E          N  dist.lakes   dist.coast
## ELEVATION_   1.0000000 -0.38731894  0.2229499 -0.49108955  0.516957928
## E           -0.3873189  1.00000000  0.3120844  0.50453701 -0.640536062
## N            0.2229499  0.31208438  1.0000000 -0.60954121  0.491465725
## dist.lakes  -0.4910895  0.50453701 -0.6095412  1.00000000 -0.843458541
## dist.coast   0.5169579 -0.64053606  0.4914657 -0.84345854  1.000000000
## mrvbf       -0.1942348 -0.14834963 -0.1745231  0.03825679  0.007069881
## tri3         0.3464743  0.16615012  0.2902955 -0.08670920  0.104353048
## pop15       -0.5603737  0.19132709 -0.4805457  0.51570539 -0.588043299
## pop2pt5     -0.5409465  0.08570716 -0.3810346  0.35285482 -0.406210533
##                    mrvbf       tri3      pop15    pop2pt5
## ELEVATION_ -0.194234793  0.3464743 -0.5603737 -0.54094650
## E          -0.148349630  0.1661501  0.1913271  0.08570716
## N          -0.174523066  0.2902955 -0.4805457 -0.38103463
## dist.lakes  0.038256788 -0.0867092  0.5157054  0.35285482
## dist.coast  0.007069881  0.1043530 -0.5880433 -0.40621053
## mrvbf       1.000000000 -0.2890518  0.1215610  0.24153216
## tri3       -0.289051831  1.0000000 -0.2965867 -0.24542933
## pop15       0.121560957 -0.2965867  1.0000000  0.76256951
## pop2pt5     0.241532159 -0.2454293  0.7625695  1.00000000

corrplot::corrplot(cor(ne.df[, pred.field.names]),
                   diag = FALSE, type = "upper",
                   method = "ellipse",
                   addCoef.col = "black")
```

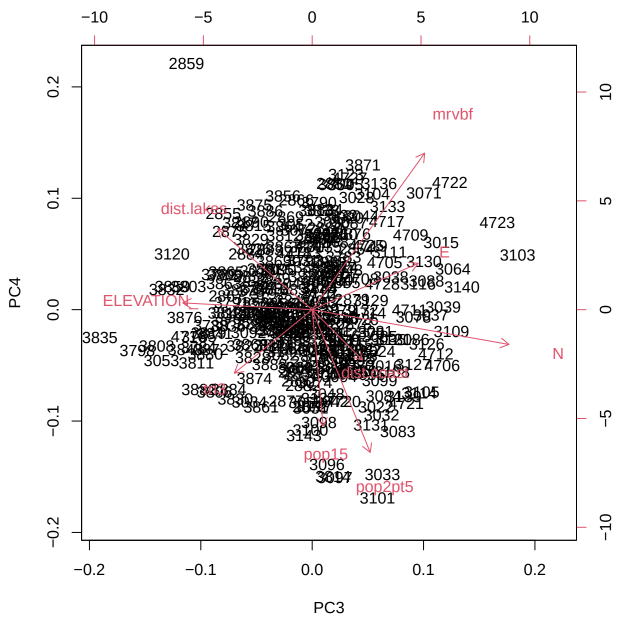


The distances to lakes and coast are inversely related, and these are related to the coordinates, because of the geography. Of course the two population densitities are positively correlated. So there is definite colinearity of predictors.

PCA Examine the **multivariate** relation between predictors with Principal Components Analysis (PCA)

```
pc <- prcomp(ne.df[, pred.field.names], scale. = TRUE, retx=TRUE)
summary(pc)

## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     1.9722 1.3402 1.0421 0.90030 0.84913 0.60906
## Proportion of Variance 0.4322 0.1996 0.1207 0.09006 0.08011 0.04122
## Cumulative Proportion  0.4322 0.6317 0.7524 0.84246 0.92257 0.96379
##                            PC7     PC8     PC9
## Standard deviation     0.44312 0.35714 0.04461
## Proportion of Variance 0.02182 0.01417 0.00022
## Cumulative Proportion  0.98561 0.99978 1.00000

pc$rotation

##                   PC1          PC2         PC3         PC4
## ELEVATION_  0.3790049  0.009868823 -0.40853219  0.02467677
## E          -0.2095170 -0.605201070  0.33442852  0.16910725
## N            0.3109224 -0.314313343  0.62114856 -0.12705918
## dist.lakes -0.4193089 -0.212978286 -0.29841764  0.29538055
## dist.coast  0.4326850  0.280778692  0.15863513 -0.18364982
## mrvbf      -0.1039718  0.409092410  0.35508637  0.57166123
## tri3        0.1774628 -0.432105357 -0.24569463 -0.23203648
## pop15      -0.4209219  0.114890542  0.03467234 -0.42571320
## pop2pt5    -0.3650623  0.208263038  0.18319644 -0.52134788
##                    PC5         PC6         PC7          PC8
## ELEVATION_ -0.162470428  0.79755536  0.12334465  0.105094901
## E           0.009735334  0.25594993  0.07815143  0.057295979
## N           0.012651062  0.16333795 -0.03239904  0.167173955
## dist.lakes -0.026389104 -0.07907947  0.14921339  0.685065908
## dist.coast -0.022078707 -0.21343002 -0.01103666  0.674014967
## mrvbf      -0.587486966  0.10787655 -0.10906923 -0.016268807
## tri3       -0.736527939 -0.31693438 -0.14230730 -0.066428423
## pop15      -0.056171541  0.30501125 -0.70805527  0.171490413
## pop2pt5    -0.285298260  0.12162744  0.64942480  0.005220769
##                      PC9
## ELEVATION_  0.0006230410
## E          -0.6118814943
## N           0.5888366804
## dist.lakes  0.3224711541
## dist.coast -0.4178478488
## mrvbf       0.0033655551
## tri3       -0.0004812559
## pop15      -0.0158959011
## pop2pt5     0.0048274545

biplot(pc)
biplot(pc, choices=3:4)
```





The two populations are closely positively related in PC1 and PC2. Lakes and coasts are almost perfectly inversely related in PC1 and PC2. Population is correlated with distance to lakes (and so inverse to distance from coast); this is a geographic accident because population is denser towards the SE (NYC, NJ). The terrain indices and geography also have a fortuitous relation. We can not easily reduce these PCs to meaningful factors.

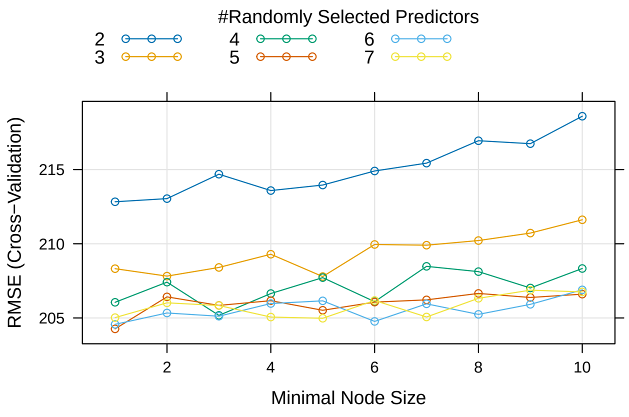
10.6.2 Random forest with additional covariables

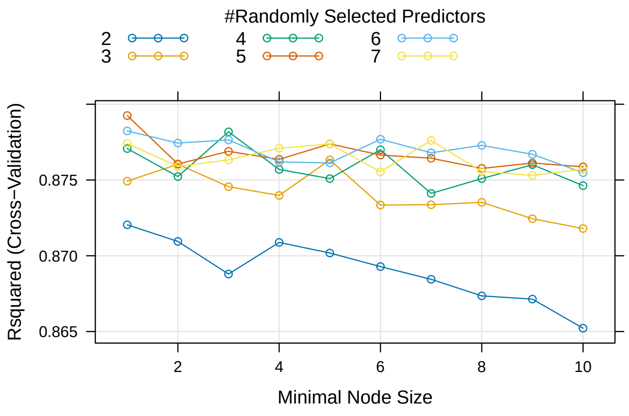
All the previous methods could be extended with this set of variables. Here we look at one, Random Forest.

```
dim(preds <- ne.df[, pred.field.names])

## [1] 305   9

length(response <- ne.df[, "ANN_GDD50"])

## [1] 305

system.time(
  ranger.tune <- train(x = preds, y = response,
                       method="ranger",
                       tuneGrid = expand.grid(.mtry = 2:7,
                                              .splitrule = "variance",
                                              .min.node.size = 1:10),
                       trControl = trainControl(method = 'cv'))
)

##    user  system elapsed
##  66.632   4.630  13.846
```

View the training results:

```
ix <- which.min(ranger.tune$result$RMSE)
ranger.tune$result[ix, c(1,3,4)]

##    mtry min.node.size     RMSE
## 31    5              1 204.2634

ix <- which.max(ranger.tune$result$Rsquared)
ranger.tune$result[ix, c(1,3,5)]

##    mtry min.node.size  Rsquared
## 31    5              1 0.8792542

ix <- which.min(ranger.tune$result$MAE)
ranger.tune$result[ix, c(1,3,6)]

##    mtry min.node.size      MAE
## 31    5              1 162.0682

plot.train(ranger.tune, metric="RMSE")
plot.train(ranger.tune, metric="Rsquared")
```





All three methods agree on five predictors to try at each split, and node size of three. There is not much improvement past `mtry=4`, and `min.node.size` from 1 to 3 give similar results.

Build a model with the optimal parameters:

```
rf.ext <- ranger(model.formula,
             data=ne.df, importance="impurity", mtry=4, min.node.size=3,
             oob.error=TRUE, num.trees=1024)
print(rf.ext)

## Ranger result
##
## Call:
```

```
##  ranger(model.formula, data = ne.df, importance = "impurity",      mtry = 4, min.node.size = 3, oob
##
## Type:                             Regression
## Number of trees:                  1024
## Sample size:                      305
## Number of independent variables:  9
## Mtry:                             4
## Target node size:                 3
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       41773.81
## R squared (OOB):                  0.8731823
```

```
str(rf.ext, max.level=1)
```

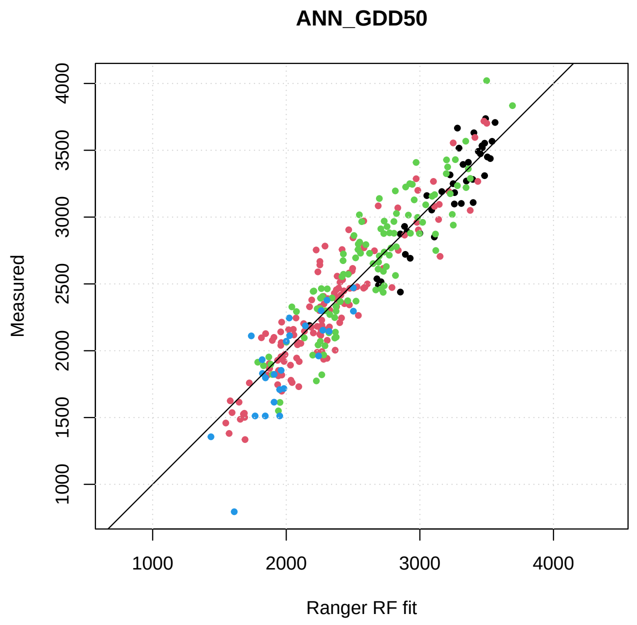
```
## List of 16
##  $ predictions              : num [1:305] 3484 3465 3563 3505 2891 ...
##  $ num.trees                : num 1024
##  $ num.independent.variables: num 9
##  $ mtry                     : num 4
##  $ min.node.size            : num 3
##  $ variable.importance      : Named num [1:9] 14568004 4201191 24527912 27200127 5687762 ...
##   ..- attr(*, "names")= chr [1:9] "ELEVATION_" "E" "N" "dist.lakes" ...
##  $ prediction.error         : num 41774
##  $ forest                   :List of 7
##   ..- attr(*, "class")= chr "ranger.forest"
##  $ splitrule                : chr "variance"
##  $ treetype                 : chr "Regression"
##  $ r.squared                : num 0.873
##  $ call                     : language ranger(model.formula, data = ne.df, importance = "impurity",
##  $ importance.mode          : chr "impurity"
##  $ num.samples              : int 305
##  $ replace                  : logi TRUE
##  $ dependent.variable.name  : chr "ANN_GDD50"
##  - attr(*, "class")= chr "ranger"
```

```
round(rf.ext$prediction.error/rf.ext$num.samples)
```

```
## [1] 137
```

```
plot(rf.ext$predictions, ne.df$ANN_GDD50, asp=1, col=ne.df$STATE, pch=20,
     main="ANN_GDD50", ylab="Measured", xlab="Ranger RF fit")
abline(0,1); grid()
round(ranger::importance(rf.ext)/sum(ranger::importance(rf.ext)),3)
```

```
## ELEVATION_          E          N dist.lakes dist.coast      mrvbf
##      0.147      0.042      0.247      0.274      0.057      0.013
##       tri3      pop15    pop2pt5
##      0.024      0.146      0.049
```

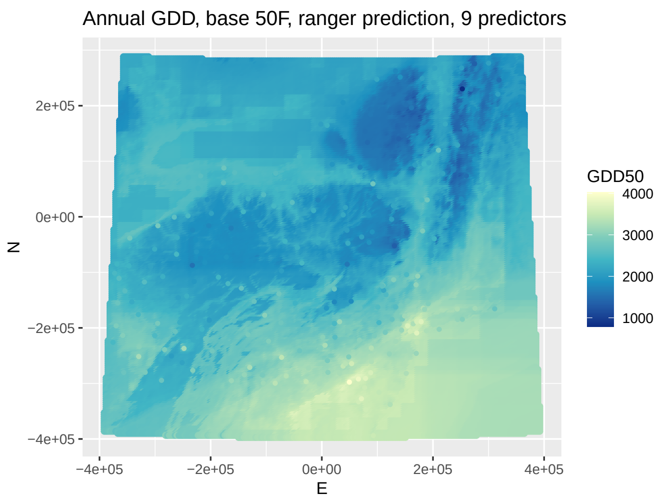


Surprisingly, distance to lakes is the most important; this may be partially substituting for Northing (also very important), but it also accounts for the lake effect we noticed in the linear model residuals. Population at 15' is somewhat important; this may be reflecting the urban heat island effect in the NY City area, but also the ocean nearby. The other areas of the coastline may not have much effect.

The mean out-of-bag error is 137 annual GDD50.

Mapping:

```
dem.ne.m.df$pred.rf.ext <- predict(rf.ext, data=dem.ne.m.df)$predictions
summary(dem.ne.m.df$pred.rf.ext)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1232    2073    2359    2460    2856    3750

display.prediction.map("pred.rf.ext",
  "Annual GDD, base 50F, ranger prediction, 9 predictors",
  "GDD50")
```



The effect of the population grid is clear in the NYC area.

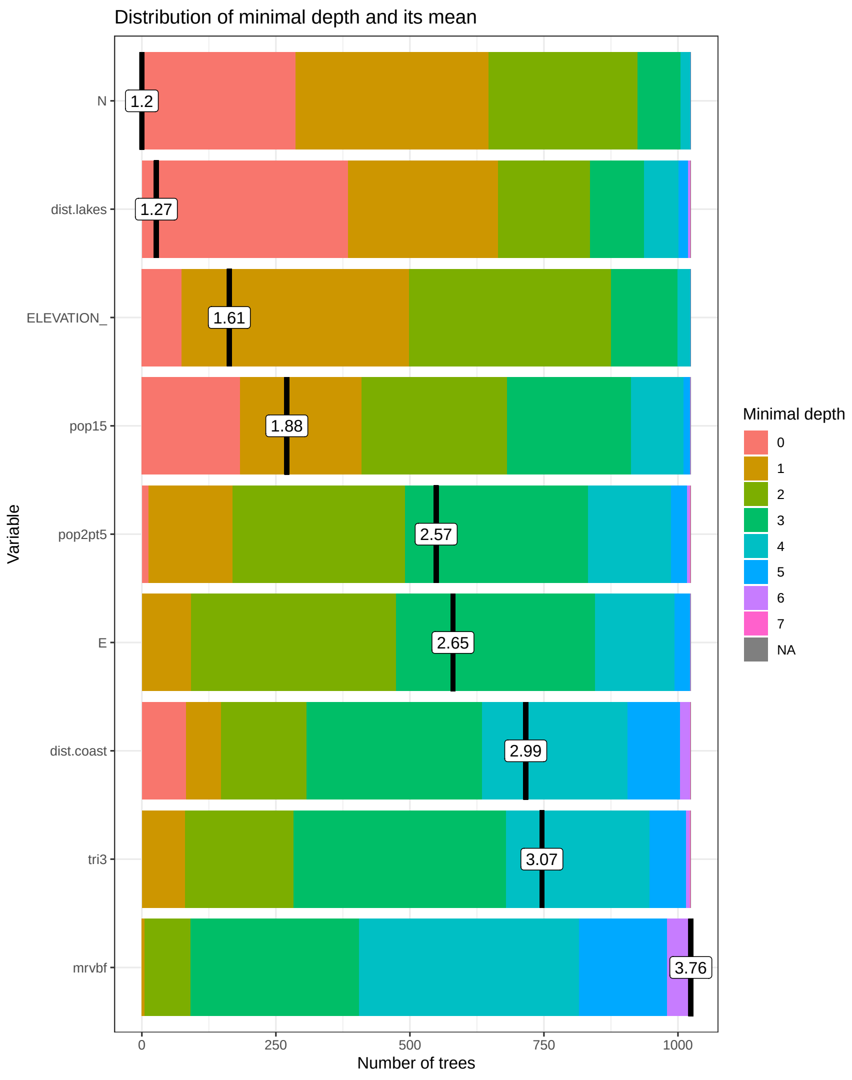
10.6.3 Variable importance in the extended model

A good way to visualize variable importance in random forests is the "minimum depth distribution", i.e., at which depth in the trees each variable is used. This is displayed by the `plot_min_depth_distribution` of the `randomForestExplainer` package.

Task 100 : Compute and display the minimum depth distribution of the 9-predictor model. •

The effect of variable substitution is clear: when one predictor is not used at a particular split, a correlated predictor can partially substitute for it. Here we have `N` and `dist.lakes` as one pair; another is `pop15` and `dist.coast` as a substitute for `ELEVATION` (highest populations in the NYC area).

```
require(randomForestExplainer)
tmp <- min_depth_distribution(rf.ext)
plot_min_depth_distribution(tmp)
```



10.7 Shapley values

The extended model is sufficiently complex to make interpretation difficult. In the past several years there have been developments in so-called "Interpretable Machine Learning"[18].

One method of interpreting models are the **Shapley values**. As explained by Christoph Molnar[18] "[There is a] method from coalitional game theory named [the] Shapley value. Assume that for one data point, the feature values play a game together, in which they get the prediction as a payout. The Shapley value tells us how to fairly distribute the payout among the feature values.

"The 'game' is the prediction task for a single instance of the dataset. The 'gain' is the actual prediction for this instance minus the average prediction for all instances. The 'players' are the feature values of the instance that collaborate to receive the gain …."

Thus the Shapley value for a predictor is its average marginal contribution over all possible coalitions.

10.7.1 *Theory

The Shapley value is defined via a value function val of players in the "game" S.

The Shapley value $\phi_j(val)$ of a feature j's value is its contribution to the payout, weighted and summed over all possible feature value combinations:

$$\phi_j(val) = \sum_{S\subseteq\{1,\ldots,p\}\setminus\{j\}} \frac{|S|!\,(p-|S|-1)!}{p!}\left(val\left(S\cup\{j\}\right) - val(S)\right)$$

where S is a subset of the features used in the model, x is the vector of feature values of the instance to be explained and p is the number of features.

Thus $val_x(S)$ is the prediction for feature values in set S that are marginalized over features that are not included in set S:

$$val_x(S) = \int \hat{f}(x_1,\ldots,x_p)d\mathbb{P}_{x\notin S} - E_X(\hat{f}(X))$$

10.7.2 Practice

This is implemented in the "Interpretable Machine Learning" package `iml`. This uses a `Predictor` object which holds any machine learning model and the data to be used to analyze it.

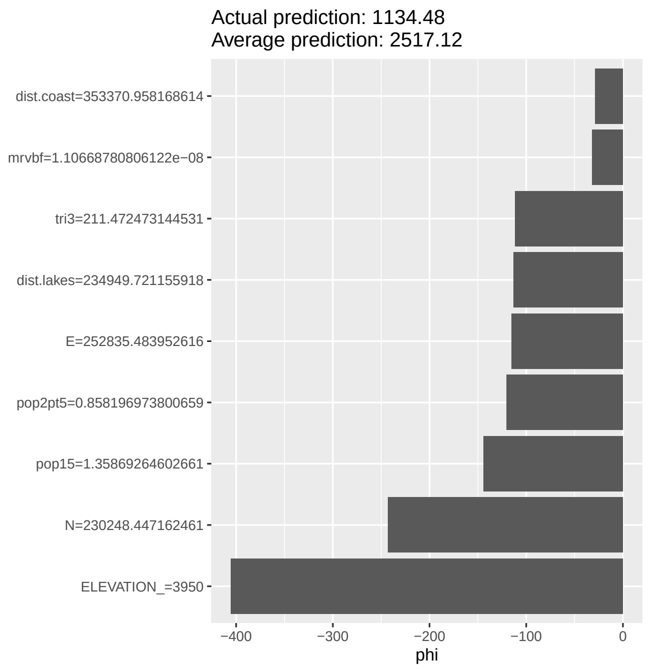
Task 101 : Build a `Predictor` object for use with `iml` functions. •

```
require(iml)
# matrix of predictors to be evaluated
X <-  ne.df[, pred.field.names]
# the
predictor <- Predictor$new(model = rf.ext, data = X, y = ne.df[, "ANN_GDD50"])
```

Task 102 : Compute the Shapley values for the predictors at some interesting stations. •

Now we can compute the Shapley values. For example, the values for the station with the fewest GDD50:

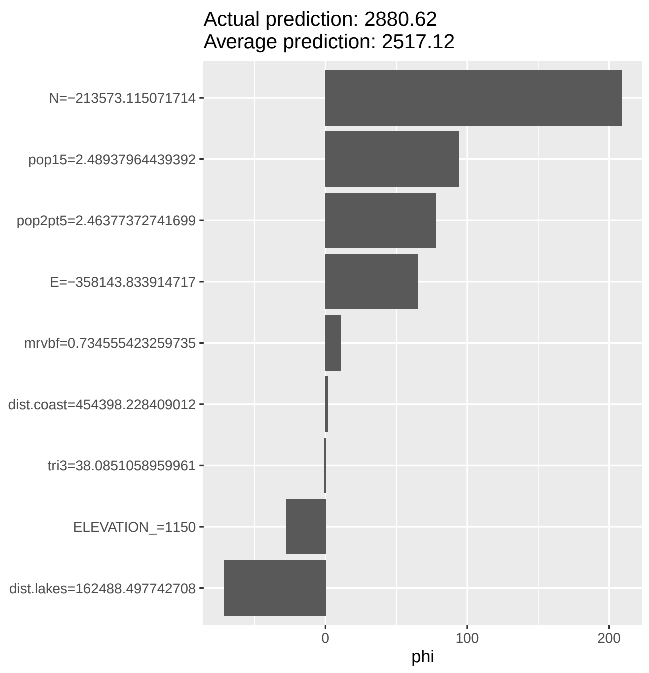
```
ix <- which.min(ne.df[, "ANN_GDD50"])
ne.df[ix, 2:3]

##      STATE      STATION_NA
## 4716    VT MOUNT MANSFIELD

X[ix,]

##      ELEVATION_        E        N dist.lakes dist.coast        mrvbf
## 4716       3950 252835.5 230248.4   234949.7     353371 1.106688e-08
##          tri3    pop15  pop2pt5
## 4716 211.4725 1.358693 0.858197

shapley <- iml::Shapley$new(predictor, x.interest = X[ix, ])
shapley$plot()
```



The figure shows the actual values of each predictor at the observation point, and the ϕ value, i.e., numerical contribution to the difference between actual and average prediction. These sum to the difference. At this climate station all contributions are negative, i.e., they assist in lowering the prediction from the average. The elevation contributes the most; Northing is also important.

Compare to the Shapley values at a more southerly station, Pittsburgh (PA) airport:

```
ix <- which(ne.df[, "STATION_NA"] == "PITTSBURGH INTL AP")
X[ix,]

##      ELEVATION_         E         N dist.lakes dist.coast     mrvbf
## 3861       1150 -358143.8 -213573.1   162488.5   454398.2 0.7345554
##          tri3   pop15  pop2pt5
## 3861 38.08511 2.48938 2.463774

shapley <- iml::Shapley$new(predictor, x.interest = X[ix, ])
shapley$plot()
```



Here the prediction is greater than the average, with positive contributions by the negative Northing and the two population densities.

So we can see for each climate station the reason for its prediction. Molnar emphasizes: "[B]e careful to interpret the Shapley value correctly: [it] is the *average contribution of a feature value to the prediction in different coalitions.* [It] is **not** the difference in prediction when we would remove the feature from the model."

10.7.3 Shapley Additive exPlanations (SHAP)

A slightly different approach to Shapley values for model interpretation are the **SH**apley **A**dditive ex**P**lanations (SHAP), see Lundberg and Lee [14] for details. Several packages have implemented SHAP, here we use the `fastshap` package[22]. The `explain` function in this package requires a prediction function, which in this case is just `predict` from the `ranger` package. This is automatically called from the generic predict when the object is a ranger model.

Task 103 : Compute the SHAP values for all predictors annd stations. •

Note the use of the `nsim` argument: "To obtain the most accurate results, `nsim` should be set as large as feasibly possible."

```
require(fastshap)
# a prediction function
pfun <- function(object, newdata) {
  predict(object, data = newdata)$predictions
}
# matrix of predictors to be evaluated
X <- ne.df[ , pred.field.names]
```

[22] https://github.com/bgreenwell/fastshap

```
fshap <- fastshap::explain(object = rf.ext,
                           X = X,
                           shap_only = FALSE, # also return feature and baseline values
                           pred_wrapper = pfun,
                           nsim = 24)
names(fshap)

## [1] "shapley_values" "feature_values" "baseline"

head(fshap$shapley_values)

##      ELEVATION_          E         N dist.lakes dist.coast      mrvbf
## [1,]   173.6749 -34.058458 280.88285 219.021918   20.88760  -3.508396
## [2,]   187.3322 -17.876824 294.64880 378.386759   57.64471  -6.101101
## [3,]   217.8836  -0.258884 396.63361 397.426697   56.17958  12.274651
## [4,]   166.7820   8.117839 321.85278 295.251817   46.86593 -10.436517
## [5,]   130.4956 -23.104757  42.30555  -3.280741   11.48722   4.181186
## [6,]   109.4734 -30.335904  37.62242  62.927985   16.55914  -2.318895
##            tri3      pop15    pop2pt5
## [1,]  0.8316515  41.952440   6.760234
## [2,]  2.5612183  72.251417   3.389438
## [3,] 10.0883111 123.351508  34.706638
## [4,] 14.8474460  22.442173 -27.788425
## [5,]  1.4598796   9.515917  25.815321
## [6,]  0.8486328  76.132392  66.479336

ne.df[1, 2:3]

##      STATE       STATION_NA
## 2852    NJ ATLANTIC CITY AP

ne.df[1, pred.field.names]

##      ELEVATION_        E         N dist.lakes dist.coast   mrvbf
## 2852         60 123044.6 -337962.4   467665.8   9457.429 0.09103
##          tri3   pop15  pop2pt5
## 2852 4.444497 2.15009 2.339589
```

Each observation has a set of SHAP values. These are the contribution to the difference between the observed value at that point and the average value of all observations, i.e., how much this predictor affects the single prediction, away from the “null” model of the overall average. For the first observation we see that Northing and distance to the Great Lakes have the most effect; elevation is also important.

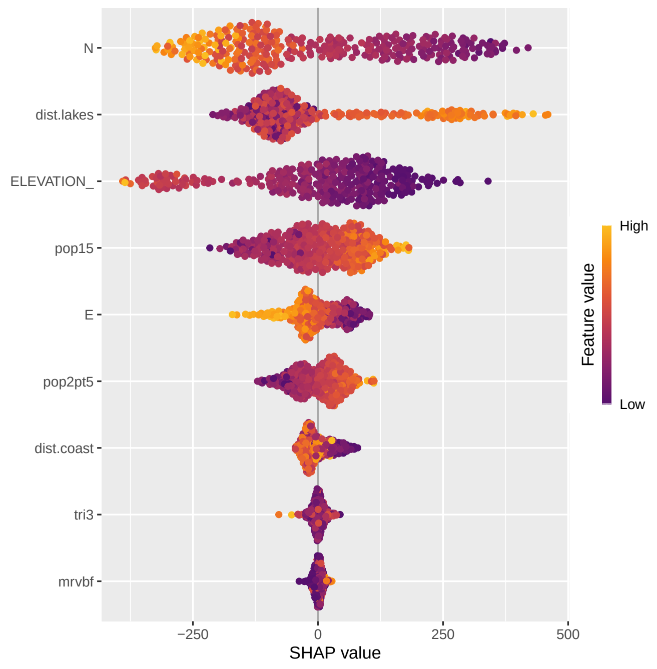
To visualize all the SHAP values for all the predictors and observations we can use the **shapviz** package. The **shapviz** function from this package requires (1) a matrix of computed SHAP values, (2) a set of features for which to display the values, (2) a list of observations.

Once the **shapviz** visualization object is set up, we can display the SHAP values with the `sv_importance` function. We have a fairly small dataset and number of predictors, so we can display all of them together in a so-called ”beehive” plot.

Task 104 : Show all the SHAP values as a “beehive” plot. •

```
require(shapviz)
sv.fshap <- shapviz(fshap,
                    X = ne.df[ , pred.field.names],
                    bg_X = ne.df) # small dataset, can see all of them
class(sv.fshap)
```

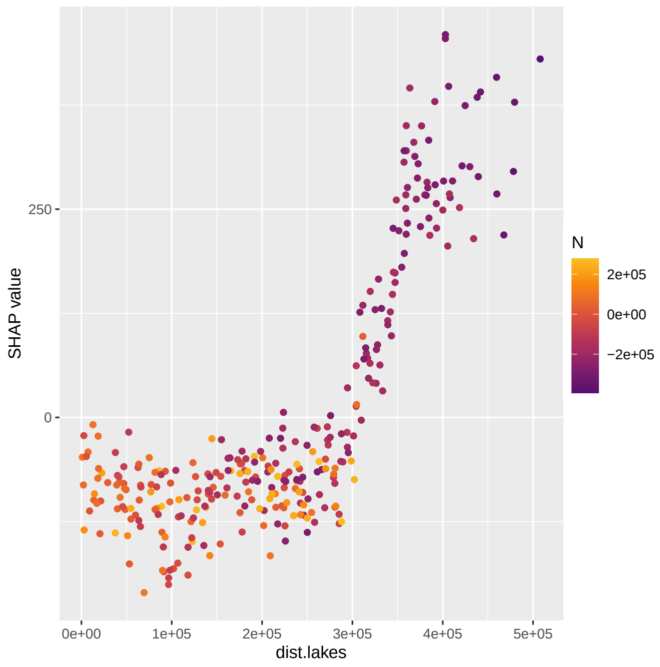
```
## [1] "shapviz"

sv_importance(sv.fshap, kind = "bee")
```



The dependence of the predictions on any predictor, vs. its value, and its relation to any other single predictor, can be shown with the `sv_dependence` function:

Task 105 : Show the SHAP values for distance to lakes, coloured by Northing. •

```
sv_dependence(sv.fshap, v = "dist.lakes", color_var = "N")
ix <- which(fshap$shapley_values[, "dist.lakes"] > 350)
ne.df[ix, 2:5]

##      STATE             STATION_NA LATITUDE_D LONGITUDE_
## 2853    NJ     ATLANTIC CITY MARINA      39.38     -74.43
## 2854    NJ                  AUDUBON      39.88     -75.08
## 2859    NJ              CAPE MAY 2 NW      38.95     -74.93
## 2869    NJ           INDIAN MILLS 2 W      39.80     -74.78
## 2875    NJ    MILLVILLE MUNICIPAL AP      39.37     -75.08
## 2883    NJ              SANDY HOOK      40.45     -73.98
## 2884    NJ          SEABROOK FARMS      39.50     -75.23
## 2890    NJ    WOODSTOWN PITTSGROV 4E      39.55     -75.17
## 3097    NY NEW YORK CITY CENTRAL PK      40.78     -73.97
## 3099    NY    NEW YORK LA GUARDIA AP      40.78     -73.88
## 3842    PA              MARCUS HOOK      39.82     -75.42
## 3858    PA      PHILADELPHIA INTL AP      39.87     -75.23
```



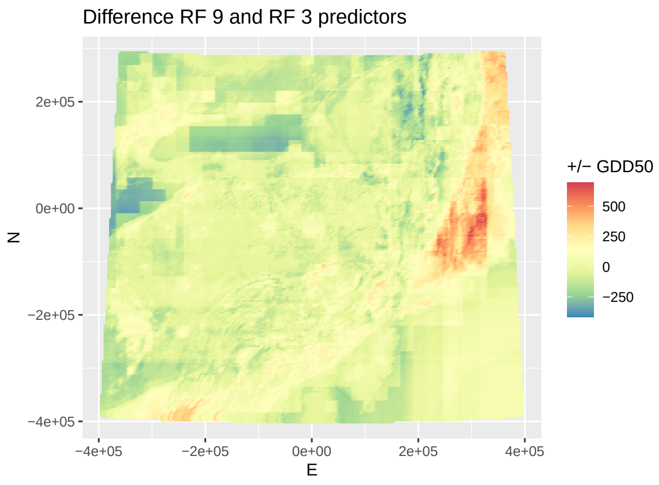
The strongly positive contributions of distance to lakes are all far from the lakes and increase with negative Northing; these are mostly in New Jersey.

10.8 Extended vs. base model

Is the extended random forst model (with more covariates) more successful than the base model (using only Northing and elevation) in modelling the growing degree days?

Task 106 : Compute and display the difference between the 9-predictor and 2-predictor maps. •

```
summary(dem.ne.m.df$diff.rf.9.3 <-
          dem.ne.m.df$pred.rf.ext - dem.ne.m.df$pred.rf)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -413.097  -72.057   -8.703    1.956   61.509  694.914

display.difference.map("diff.rf.9.3",
  "Difference RF 9 and RF 3 predictors",
  "+/- GDD50")
```



The largest differences are outside the study area, mostly (?) due to distance from the Great Lakes. Within it, small negative adjustments on Long Island, but also near Lake Ontario (not correct).

11 Thin-plate spline interpolation

A quick way to see the distribution of a variable in space as a surface is with an empirical method that adjusts locally to the data. A common empirical method is **thin-plate splines** (TPS), also referred to as "minimum curvature", which are implemented in the `fields` package.

11.1 * Theory

Hastie et al. [7, §5.7] explains the mathematics of multi-dimensional smoothing splines. A more thorough mathematical treatment is given by Wood [28] and Mitasova and Mitas [17]; these are developments from the "minimum curvature" methods of Briggs [3]. Applications include Hutchinson [9] and Mitasova and Hofierka [16].

TPS is the mathematical equivalent of a thin (so, flexible) plate that is warped to fit the data. This can range from very "rigid", i.e., just a single surface (the usual least-squares plane of a first-order trend surface) to very "flexible", i.e., perfectly fitting every observation. In general we want something in between: if we think there is an overall surface we just fit it as one polynomial (first, second …order polynomials on the coördinates), but if we want to fit more locally, we must expect local noise which should be somehow locally averaged-out.

Fitting a TPS depends on the k data points with known coördinates and attribute values. They can be described by $2(k+3)$ parameters, six of which are overall affine transformation parameters (to center the function in 2D) and $2k$ of which link to the control points.

The general method is to minimize the residual sum of squares (RSS) of the fitted function, subject to a constraint that the function be "smooth" in some sense; this is expressed by a **roughness penalty** which balances the fit to the observations with smoothness. This is a minimization problem. If $\mathrm{x_i}$ is one point in 2D space (i.e., it has two coördinates) and y_i is the attribute value at the same points, the aim is to minimize:

$$\min_f \sum_{i=1}^{N} \{y_i - f(\mathrm{x_i})\}^2 + \lambda J[f] \tag{23}$$

where J is the penalty function and λ controls how important it is; $\lambda = 0$ means there is no roughness penalty and the data will be fit exactly; as $\lambda \to \omega$ the solution approximates the least-squares plane, i.e., the trend surface averaged over all the points.

In 2D an appropriate penalty is:

$$J[f] = \int_{\mathbb{R}}\int_{\mathbb{R}} \left[\left(\frac{\partial^2 f(\mathrm{x})}{\partial x_1^2} \right)^2 + 2 \left(\frac{\partial^2 f(\mathrm{x})}{\partial x_1 \partial x_2} \right)^2 + \left(\frac{\partial^2 f(\mathrm{x})}{\partial x_2^2} \right)^2 \right] \mathrm{d}x_1 \mathrm{d}x_2 \tag{24}$$

where (x_1, x_2) are the two coördinates of the vector x. In practice the double integral is discretized over some grid known as **knots**; these may be defined by the observations or may be a different set, maybe an evenly-spaced grid.

This penalty can be interpreted as the "bending energy" of a thin plate represented by the function $f(x)$; by minimizing this energy the spline function in over the 2D plane is a thin (flexible) plate which, according to the first term of Equation 23 would be forced to pass through data points, with minimum bending. However the second term of Equation 23 allows some smoothing: the plate does not have to bend so much, since it is allowed to pass "close to" but not necessarily through the data points. The higher the λ, the less exact is the fit. This has two purposes: (1) it allows for measurement error; the data points are not taken as exact; (2) it results in a smoother surface. So cross-validation is used to determine the degree of smoothness.

The solution to Equation 24 is a linear function:

$$f(\mathrm{x}) = \beta_0 + \beta^T \mathrm{x} + \sum_{j=1}^{N} \alpha_j h_j(\mathrm{x}) \tag{25}$$

where the β account for the overall trend and the α are the coefficients of the warping.

The set of functions $h_j(\mathrm{x})$ is the **basis kernel**, also called a **radial basis function** (RBF), for thin-plate splines:

$$h_j(\mathrm{x}) = \|\mathrm{x} - \mathrm{x}_j\|^2 \log \|\mathrm{x} - \mathrm{x}_j\| \tag{26}$$

where the norm distance $r = \|\mathrm{x} - \mathrm{x}_j\|$ is also called the **radius** of the basis function. The norm is usually the Euclidean (straight-line) distance.

11.2 Practice

Here we only uses the Northing and Easting as predictors.

Task 107 : Set up for thin-plate splines and compute the minimum-curvature spline, subject to roughness constraint determined by generalized cross-validation. •

The `Tps` function of the `fields` package compute this; however the coörd-inates must be formatted as a matrix field in the dataframe, using the `matrix` function.

```
require(fields)
ne.tps <- ne.df
ne.tps$coords <- matrix(c(ne.df$E, ne.df$N), byrow=F, ncol=2)
surf.1 <-Tps(ne.tps$coords, ne.tps$ANN_GDD50)
class(surf.1)

## [1] "Krig" "Tps"

summary(surf.1)

## CALL:
## Tps(x = ne.tps$coords, Y = ne.tps$ANN_GDD50)
##
##  Number of Observations:                305
##  Number of unique points:               305
##  Number of parameters in the null space 3
##  Parameters for fixed spatial drift     3
##  Effective degrees of freedom:          88.5
##  Residual degrees of freedom:           216.5
##  MLE tau                                220.7
##  GCV tau                                228.6
##  MLE sigma                              129700000
##  Scale passed for covariance (sigma)    <NA>
##  Scale passed for nugget (tau^2)        <NA>
##  Smoothing parameter lambda             0.0003755
##
## Residual Summary:
##        min      1st Q     median      3rd Q        max
## -882.40000 -113.60000   -0.01774  127.30000  574.90000
##
## Covariance Model: Rad.cov
##   Names of non-default covariance arguments:
##        p
##
## DETAILS ON SMOOTHING PARAMETER:
##  Method used:   GCV    Cost:  1
##    lambda       trA       GCV   GCV.one GCV.model    tauHat
## 3.755e-04 8.846e+01 7.359e+04 7.359e+04        NA 2.286e+02
##
##  Summary of all estimates found for lambda
##                lambda   trA   GCV tauHat -lnLike Prof converge
## GCV         0.0003755 88.46 73587  228.6         2149       13
## GCV.model          NA    NA    NA     NA           NA       NA
## GCV.one     0.0003755 88.46 73587  228.6           NA       13
## RMSE               NA    NA    NA     NA           NA       NA
## pure error         NA    NA    NA     NA           NA       NA
## REML        0.0009690 59.35 74532  245.0         2146        7
```

Task 108 : Set up a grid covering the four States at approximately 6 x 6 km resolution, and convert to a dataframe with the coördinates as a matrix field. This last is because the `fields` package works with coördinate matrices. •

The `spsample` function of the `sp` package can make various sampling plans, including a regular grid, within a study area.

We compute the approximate area of the four states, in km, from its bounding box in the US Census state shapefile; these are in m^2, and so must be converted to km^2. We then ask for a grid with each cell covering about (9 km)^2.

> **Note:** Recall, this is not for exact prediction, just to get an overview of the regional distribution of the variable of interest.

```
resolution <- 9
st_bbox(state.ne.m)

##      xmin      ymin      xmax      ymax
## -387135.2 -396311.1  357707.2  288684.4

(ne.sq.km <- diff(st_bbox(state.ne.m)[c(1,3)]) * diff(st_bbox(state.ne.m)[c(2,4)])/10^6)

##     xmax
## 510213.8

(approx.n.grid.cells <- ceiling(ne.sq.km/(resolution^2)))

## xmax
## 6299

states.grid <- st_sample(state.ne.m, size = approx.n.grid.cells, type="regular")
class(states.grid)

## [1] "sfc_POINT" "sfc"

states.grid.df <- as.data.frame(states.grid)
states.grid.df$coords <- as.matrix(st_coordinates(states.grid))
str(states.grid.df)

## 'data.frame': 6285 obs. of  2 variables:
##  $ geometry:sfc_POINT of length 6285; first list element:  'XY' num  94145 -390279
##  $ coords  : num [1:6285, 1:2] 94145 100922 94145 100922 100922 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "X" "Y"
```

Task 109 : Predict over the four-states grid using the fitted thin-plate spline. •

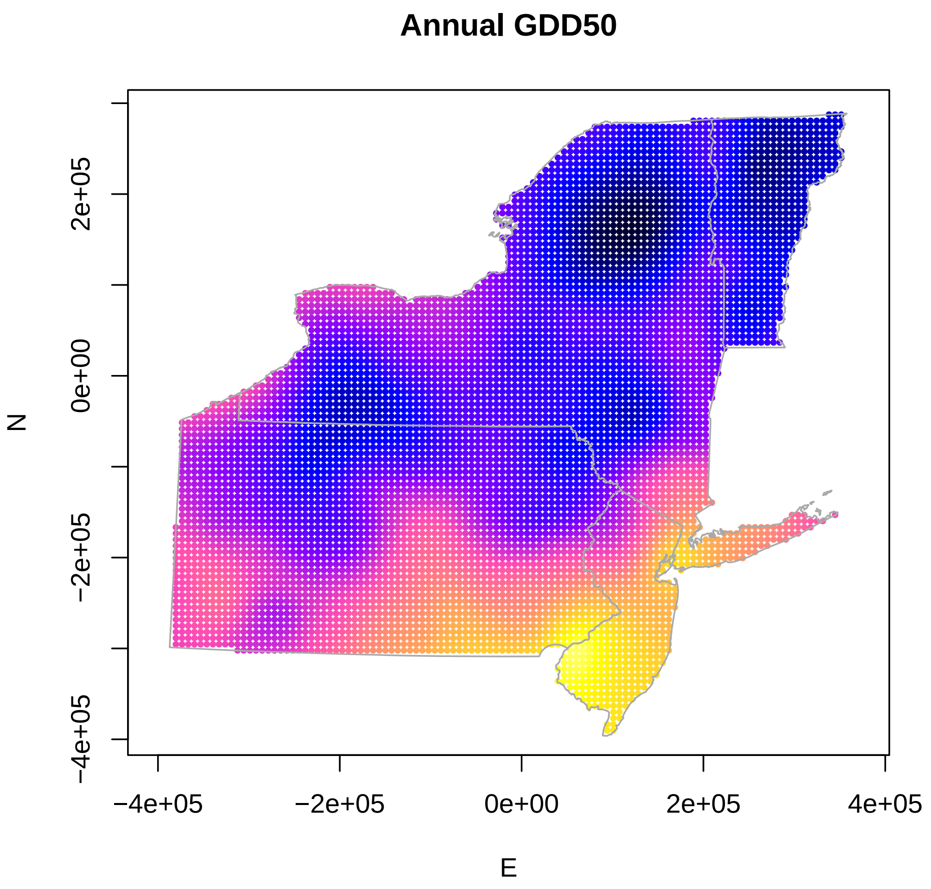
The `predict.Krig` method of the `fields` package computes the prediction as a matrix.

```
surf.1.pred <- predict.Krig(surf.1, states.grid.df$coords)
class(surf.1.pred)

## [1] "matrix" "array"

dim(surf.1.pred)

## [1] 6285    1
```

```
summary(as.vector(surf.1.pred))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1460    2058    2288    2389    2680    3695
```

Task 110 : Display the gridded prediction. •

```
plot(states.grid.df$coords, pch=20, asp=1, cex=.6,
     col=sp::bpy.colors(256)[cut(surf.1.pred, 256)],
     xlab="E", ylab="N",
     main="Annual GDD50")
plot(state.ne.m.boundary, add = TRUE, col = "darkgray")
```



This map captures the main features of the annual GDD50 fairly well, even though elevation was not used in the thin-plate spline empirical model. In particular, it captures the high-GDD areas along the Lake Ontario and Lake Erie plains, the low-GDD cold spots in the Adirondack, Catskill and Green Mountains, as well as in the Allegheny State Park area of SW NY/NE PA, and the very high GDD-area around the Delaware Bay. It does not account for local variations in GDD because of elevation.

> **Note:** It is also possible to include the elevation in the thin-plate spline model, but but because it varies at such short ranges, the result is nonsense.

To determine the predicted value at any location, just predict at that point.

To make the target point as a `sfc_POINT` we first create the point geometry with `st_point` and then make it a spatial object with Simple Features geometry with the `st_sfc`ethod. And of course we need to specify its CRS. Since we first specify geographic coördinates, we then transform to the CRS used in the grid.

```
pt <- st_sfc(st_point(x=c(-76.402175, 42.453271)))
st_crs(pt) <- 4326
pt <- st_transform(pt, st_crs(states.grid))
st_coordinates(pt)

##              X         Y
## [1,] -33055.67 -5118.213

(pt.pred <- surf.1.pred <- predict.Krig(surf.1, st_coordinates(pt)))

##          [,1]
## [1,] 2205.347
```

The thin-plate spline interpolation predicts 2205 annual GDD50 for this location.

12 Local interpolators

A purely **local** approach to prediction is to ignore any causative factors (in this case, northing and elevation) and just use "nearby" known observations to predict at any location. This is an operational realization of the well-known Tober's First Law of Geography: "everything is related to everything else, but near things are more related than distant things" [23][23]. In the current case this is not advisable, because of the strong and useful relation of the target variable `ANN_GDD50` with the covariables, which we have seen in earlier sections. However, for completeness we illustrate this method.

Local approaches can be **model-based** or **model-free**.

The best-known model-based method is Ordinary Kriging, which relies on a model of local spatial dependence of the target variable. In this case the universal model of spatial distribution shown in 1 is simplified to:

$$Z(\mathrm{s}) = \varepsilon(\mathrm{s}) + \varepsilon'(\mathrm{s}) \tag{27}$$

(s) : a location in space, designated by a **vector** of coördinates;

$Z(\mathrm{s})$: **true** (unknown) value of some property at the location;

- when modelled, expressed as **most likely** value and some **uncertainty**, or as a **probability distribution**

$\varepsilon(\mathrm{s})$: locally **spatially-autocorrelated stochastic** component;

$\varepsilon'(\mathrm{s})$: pure ("white") **noise**, no structure.

In the purely local model there is no deterministic component, i.e., global coördinates or covariables.

[23] Tobler goes on to point out that "the ...model used is thus very parochial, and ignores most of the world"

We model $\varepsilon(s)$ with an **authorized model of spatial dependence**, usually with an authorized variogram model. This model is then used in Ordinary Kriging (OK). This also reveals the magnitude of $\varepsilon'(s)$, i.e., the pure noise that can not be modelled nor predicted.

Good explanations of Ordinary Kriging are from Webster and Oliver [24] and Goovaerts [6].

12.1 Computing the empirical variogram

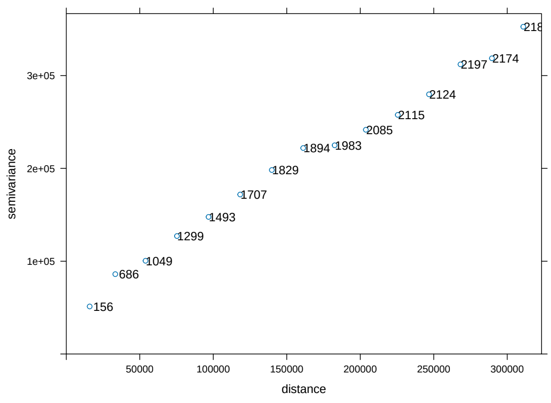
The definition of the empirical variogram was explained in §4.4.1. Here we apply it to the original values, not the trend residuals as in that section.

Task 111 : Display an empirical variogram to 400 km maximum separation. •

We use the `variogram` function of the `gstat` package. The default bin width for this function is 1/15 the cutoff. By default the cutoff is 1/3 of the diagonal across the bounding box of the point-set. These are both 'rules of thumb' and should be adjusted by the analyst until (1) the range of spatial dependence (if any) is found; (2) each bin has enough point-pairs; (3) the structure is clear.

First with the defaults. The cutoff is then 3.21841×10^5 km, and the bin widths about 2.1456×10^4 km.

```
library(gstat)
st_bbox(ne.m)

##      xmin      ymin      xmax      ymax
## -375745.2 -393922.9  318960.4  276614.1

(cutoff.default <- (sqrt(diff(st_bbox(ne.m)[c(1,3)])^2 +
                           diff(st_bbox(ne.m)[c(2,4)])^2))/3)

##     xmax
## 321841.2

(binwidth.default <- cutoff.default/15)

##     xmax
## 21456.08

plot(v.o <- variogram(ANN_GDD50 ~ 1, locations=ne.m),
     plot.numbers=TRUE)
```

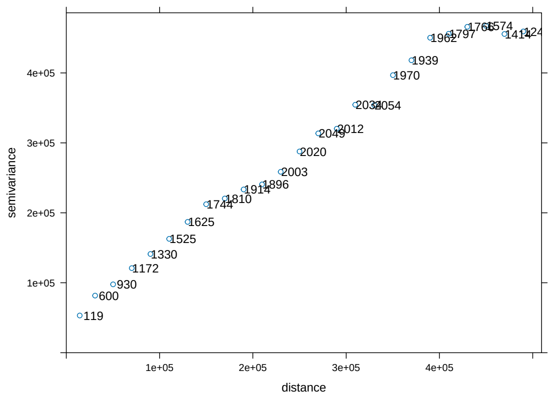


Within this range the variogram is unbounded, which means the maximum variation has not been reached even between observations at the largest separations. This is because of the strong regional effect of Northing.

> **Note:** Notice the fairly large nugget variance. In the residual variogram used in KED and RK there was no nugget. How can this be explained?

Extend the cutoff to see if we can find a bound:

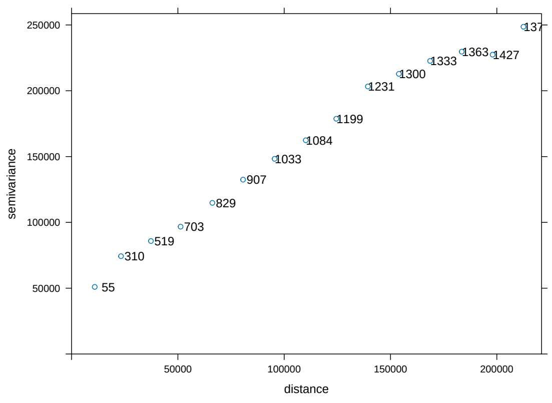
```
plot(v.o <- variogram(ANN_GDD50 ~ 1, locations=ne.m, cutoff=500000, width=20000),
     plot.numbers=TRUE)
```



Here we finally do get a bound, at about 430 km. There appears to be two structures: one to about 180 km and one to about 430 km. Since OK is a local predictor, and the nearer points receive most of the weight, this short-range variation is what we want to model for local prediction.

Task 112 : Recompute and display the empirical variogram to 220 km cutoff. •

```
v.o <- variogram(ANN_GDD50 ~ 1, locations=ne.m,
                 cutoff=220000)
plot(v.o, plot.numbers=TRUE)
```



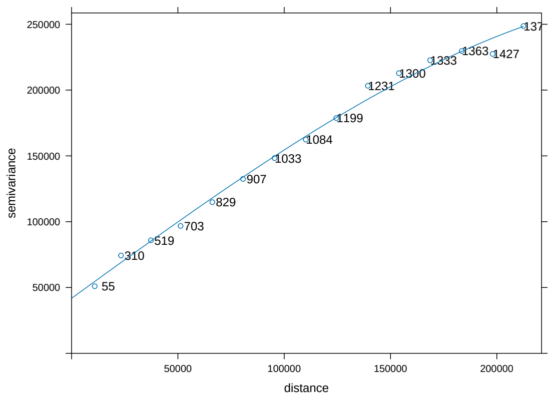
12.2 Fitting an authorized variogram model

We require a continuous function of semivariance vs. separation, so that for any separation we can compute the semivariance to be used in the kriging equations. This must be an **authorized** variogram model that ensures that the kriging system will be postive semi-definite and thus invertible.

This empirical variogram can be well-modelled with a spherical variogram model; at short ranges this is almost linear. The nugget appears to be about 40 000 GDD^2, the partial sill at the maximum semivariance of about 250 000 GDD^2 less the nugget, and the range can be approximated by the maximum separation.

The `fit.variogram` function fits variogram models specified with the `vgm` "variogram model" function. We initialize the weighted least-squares (WLS) fit to the empirical variogram with our eyeball estimates.

```
(vmf.o <- fit.variogram(v.o,
                        vgm(psill=210000, model="Sph",
                            range=220000, nugget=40000)))

##   model     psill    range
## 1   Nug  41816.39      0.0
## 2   Sph 232353.41 297706.6

plot(v.o, plot.numbers=TRUE, model=vmf.o)
```



This fits fairly well; our original estimates were not too far off. The range is fitted to be about 35% longer than our estimate.

12.3 Predicting by Ordinary Kriging

Once we have the fitted model and the data points, we can predict at any location, by solving the OK system (§7.1) for the weights λ_i to be used in the weighted average (Eqn. 14). The mathematics of OK were presented in (§7.1).

Task 113 : Predict over this grid with OK. •

For Ordinary Kriging, we can use stations that we know are in the "local" region, for example the nearest "few" stations, or we can use all points within the range of the empirical variogram. The latter will have a strong averaging effect; the former will show artefacts where some stations come into or leave the range from a prediction point. Theoretically we should use all the stations; but if we have reason to suspect first-order non-stationarity, we may prefer to just use the closest stations.

For the local OK we specify the optional `nmax` to use only the twenty-four closest stations.

We also specify the optional `block` argument, to average the prediction over a 1 km^2 block.

```
# dem.ne.m.sp was set up in \S6.2 "Adjusting the grid for prediction"
k.ok <- krige(ANN_GDD50 ~ 1, locations=ne.m, newdata=dem.ne.m.sf,
              model=vmf.o, nmax=24, block=c(1000,1000))

## [using ordinary kriging]

summary(k.ok)

##    var1.pred       var1.var              geometry
##  Min.   :1483   Min.   : 12201   POINT        :40052
##  1st Qu.:2189   1st Qu.: 28249   epsg:NA      :    0
##  Median :2534   Median : 37303   +proj=aea ...:    0
##  Mean   :2548   Mean   : 76574
##  3rd Qu.:2845   3rd Qu.:105569
##  Max.   :3740   Max.   :337793
```

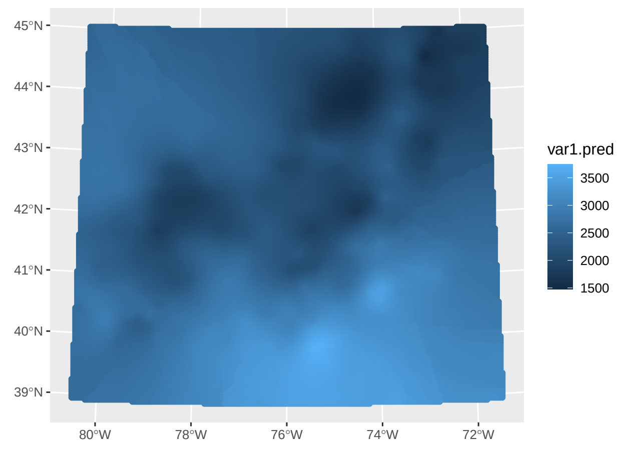
Task 114 : List the 24 stations that were used for the local prediction at the block containing the Ithaca weather station, in order of their separations from Ithaca. •

```
e.ith.pt <- subset(ne.m, (ne.m$STATION_NA=="ITHACA CORNELL UNIV"))
dist.pt <- st_distance(ne.m, e.ith.pt)
(ix <- order(dist.pt)[1:24])

##  [1] 100 148  72  49 154  54  81 167  82  48 133  85 119  40  53 128
## [17] 271 152  62 241 145  51  76 147

print(cbind(ne.df[ix,c('STN_NAME','STATE','ELEVATION_','ANN_GDD50')],
      dist=round(dist.pt[ix,]/1000,1)))

##                        STN_NAME STATE ELEVATION_ ANN_GDD50     dist
## 3074      ITHACA CORNELL UNIV    NY        960      2160  0.0 [m]
## 3122              SPENCER 2 N    NY       1050      1988 22.6 [m]
## 3046                 CORTLAND    NY       1129      2329 27.7 [m]
## 3023     AURORA RESEARCH FARM    NY        830      2590 35.2 [m]
## 3128     TULLY HEIBERG FOREST    NY       1899      1746 46.8 [m]
## 3028  BINGHAMTON BROOME CO AP    NY       1600      2141 47.7 [m]
## 3055                   ELMIRA    NY        844      2395 48.4 [m]
## 3141                  WAVERLY    NY        845      2304 50.5 [m]
## 3056                ENDICOTT    NY        827      2245 51.2 [m]
## 3022                  AUBURN    NY        744      2352 52.9 [m]
## 3107                PENN YAN    NY        830      2430 55.0 [m]
## 3059     GENEVA RESEARCH FARM    NY        718      2396 67.4 [m]
## 3093          MORRISVILLE 6 SW    NY       1300      1818 72.6 [m]
## 3014                 ADDISON    NY        980      2123 74.0 [m]
## 3027                    BATH    NY       1120      2055 74.9 [m]
## 3102                 NORWICH    NY       1020      2203 76.1 [m]
## 3881            TOWANDA 1 ESE    PA        750      2447 77.8 [m]
## 3126 SYRACUSE HANCOCK INTL AP    NY        410      2467 79.8 [m]
## 3036          CANANDAIGUA 3 S    NY        720      2531 81.3 [m]
## 3851                MONTROSE    PA       1420      2064 81.3 [m]
## 3119            SHERBURNE 2 S    NY       1080      2157 82.7 [m]
## 3025           BAINBRIDGE 2 E    NY        994      2063 84.4 [m]
## 3050                 DEPOSIT    NY       1000      2245 94.1 [m]
## 3121            SODUS CENTER    NY        420      2558 95.5 [m]
```

Notice the wide range of elevations in this group of stations, from 410' to almost 1900'.

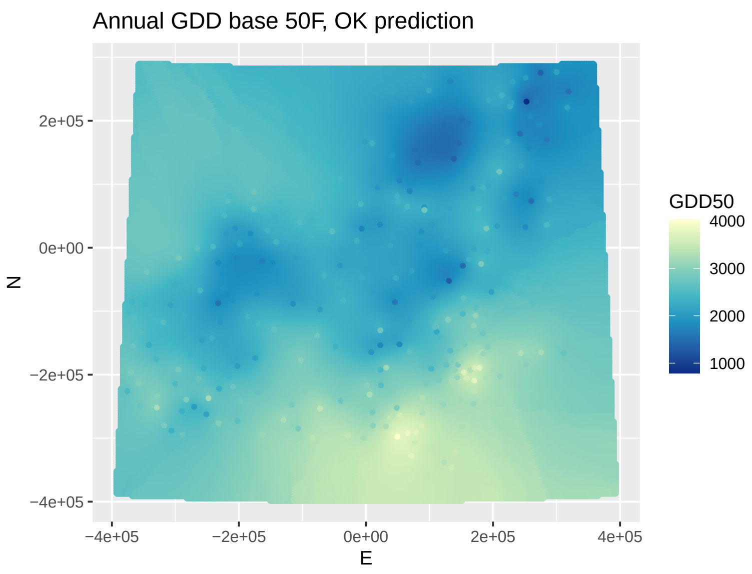
Task 115 : Display a map of the `ANN_GDD50` predicted by OK. •

```
ggplot(k.ok) +
  geom_sf(aes(col=var1.pred))
```



For comparison with the maps made with other techniques, we also add the results to the `data.frame` covering the same area and then display it with `ggplot`:

```
dim(dem.ne.m.df)

## [1] 40052    32

dem.ne.m.df$pred.ok <- k.ok$var1.pred
display.prediction.map("pred.ok",
                       "Annual GDD base 50F, OK prediction",
                       "GDD50")
```



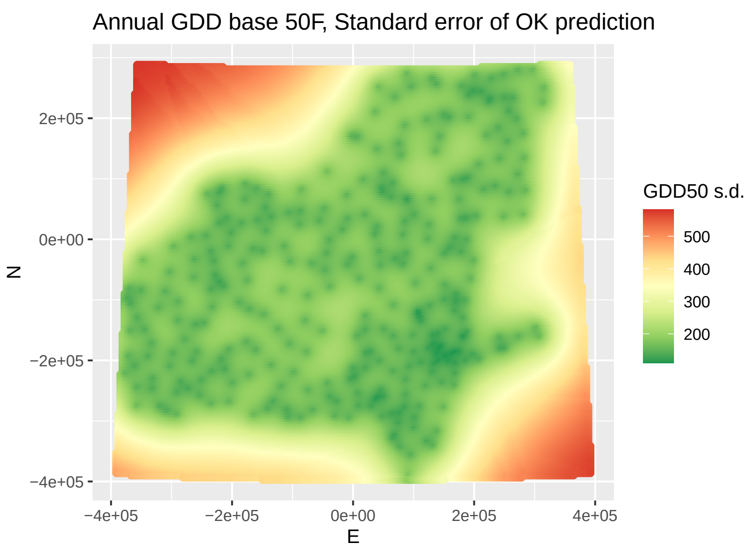
We see this is much smoother than other methods, mainly because it does not take elevation into account.

12.3.1 Accuracy assessment

An advantage of OK is that it gives a direct estimate of its prediction variance at each prediction location – this is because the variance must be minimized, so it must be computed.

Task 116 : Display a map of the OK prediction standard deviations. •

```
dem.ne.m.df$sd.ok <- sqrt(k.ok$var1.var)
ggplot() +
    geom_point(aes(x=E, y=N, colour=sd.ok), data=dem.ne.m.df) +
    xlab("E") + ylab("N") + coord_fixed() +
    ggtitle("Annual GDD base 50F, Standard error of OK prediction") +
    scale_colour_distiller(name="GDD50 s.d.", space="Lab", palette="RdYlGn",
                           direction=-1)
```



Clearly, the further from local information, the more the prediction is uncertain, e.g., northwest of Lake Ontario. The prediction uncertainty is least near stations, especially near a cluster of stations, e.g., the NYC area. We could choose to not predict at areas with too much uncertainty, based on user requirements.

The minimum and mean prediction standard deviations:

```
summary(dem.ne.m.df$sd.ok)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   110.5   168.1   193.1   252.0   324.9   581.2

ix <- which.min(dem.ne.m.df$sd.ok)
k.ok[ix,"var1.pred"]

## Simple feature collection with 1 feature and 1 field
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 164275.7 ymin: -193434.4 xmax: 164275.7 ymax: -193434.4
## Projected CRS: +proj=aea +lat_0=42.5 +lat_1=39
##          +lat_2=44 +lon_0=-76 +ellps=WGS84 +units=m
##       var1.pred                   geometry
## 27807  3415.665 POINT (164275.7 -193434.4)

round(100*dem.ne.m.df[ix,"sd.ok"]/k.ok$var1.pred[ix],1)

## [1] 3.2
```

The minimum is quite small, only about 3% of the prediction at that point, and the mean is only about 2.5 times as poor.

A better way to evaluate the predictive power OK is by **Leave-one-out cross-validation** (LOOCV). Here each point is removed from the dataset in turn, and predicted by the others, using the fitted variogram model. If the observation points well represent the total population, as they do here by design of the weather station network, this gives a good estimate of the prediction error.

Task 117 : Compute and summarize the LOOCV for this OK prediction. •

The `krige.cv` function of the `gstat` package computes this:

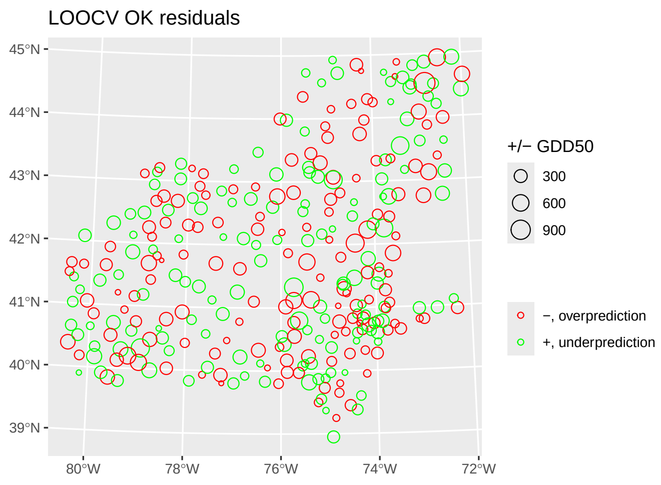
```
kcv.ok <- krige.cv(ANN_GDD50 ~ 1, locations=ne.m, model=vmf.o)
summary(kcv.ok$residual)
```

Overall the results are fairly good, but there are some extremely bad predictions. An overall measure is the **root of the mean squared error**, RMSE:

```
(loocv.ok.rmse <- sqrt(sum(kcv.ok$residual^2)/length(kcv.ok$residual)))

## [1] 268.1796
```

Task 118 : Display a bubble plot of the LOOCV residuals. •

```
bubble.sf("kcv.ok", "residual", "GDD50", "LOOCV OK residuals")
```



There are several regions with intermixed fairly large under- and over-predictions; this means that in these regions there are local factors, most notable elevation.

Task 119 : Find the worst predictions, try to explain in geographic terms. •

```
ne.m[which.min(kcv.ok$residual),2:6]

## Simple feature collection with 1 feature and 5 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 252835.5 ymin: 230248.4 xmax: 252835.5 ymax: 230248.4
## Projected CRS: +proj=aea +lat_0=42.5 +lat_1=39
##          +lat_2=44 +lon_0=-76 +ellps=WGS84 +units=m
##      STATE      STATION_NA LATITUDE_D LONGITUDE_ ELEVATION_
## 4716    VT MOUNT MANSFIELD      44.53     -72.82       3950
##                         geometry
## 4716 POINT (252835.5 230248.4)

ne.m[which.max(kcv.ok$residual),2:6]

## Simple feature collection with 1 feature and 5 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 22581.52 ymin: -130037 xmax: 22581.52 ymax: -130037
## Projected CRS: +proj=aea +lat_0=42.5 +lat_1=39
##          +lat_2=44 +lon_0=-76 +ellps=WGS84 +units=m
##      STATE               STATION_NA LATITUDE_D LONGITUDE_ ELEVATION_
## 3888    PA WILKES BRE SCTN AP AVOCA      41.33     -75.73        930
##                        geometry
## 3888 POINT (22581.52 -130037)
```

The largest under-predictions (i.e., most negative LOOCV residual) is Mt. Mansfield (VT). This station is the only one on the higher parts of the Green Mountains, and is geographically near to Lake Champlain to its west. The largest over-prediction is the AVP (Wilkes-Barre/Scranton) airport in Avoca, near Wilkes-Barre (PA). This station is on an exposed plateau above the nearby Lackawanna Valley stations.

Challenge: Find the optimal local neighbourhood for OK by testing several numbers of nearest neighbours and/or maximum distance, and comparing their cross-validation statistics.

Challenge: Much of the LOOCV errors from OK could be because local elevation is not taken into account. There can be a substantial elevation variation within the nearest 24 stations. Apply KED (§8) with elevation (or its square root) as the covariable and see how much this improves the LOOCV error.

12.4 Inverse-distance interpolation

In case a model of spatial dependence can not be built, a fallback is a model-free empirical interpolation. There are many such, none of which have any theoretical basis, but which can be tested by cross-validation:

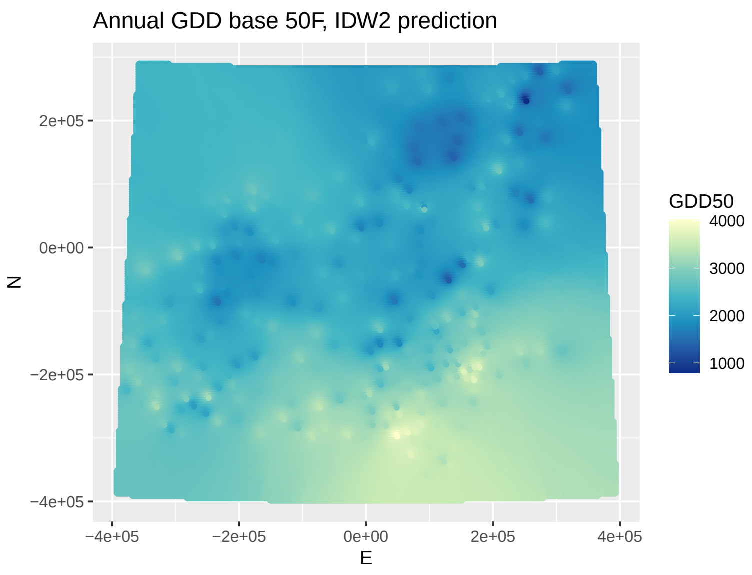
- inverse distance weighting, to some power (generally one or two);
- average of observations within some radius;
- average of some number of nearest observations.

The `krige` function with a null model computes inverse-distance interpolation; another way is with the convenience function `idw` function, which has an optional `idp` "inverse distance power" argument gives the decay power; the default is two, i.e., quadratic decay. Since there is no model, and hence no optimization of the prediction variance, there is here no internal estimate of uncertainty.

```
k.idw <- idw(ANN_GDD50 ~ 1, locations=ne.m, newdata=dem.ne.m.sf,
             idp=2, nmax=24)

## [inverse distance weighted interpolation]

summary(k.idw$var1.pred)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   851.8  2158.9  2418.0  2527.0  2888.9  4013.0

summary(k.ok$var1.pred)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1483    2189    2534    2548    2845    3740
```

Plot the result:

```
dem.ne.m.df$pred.idw <- k.idw$var1.pred
display.prediction.map("pred.idw",
                       "Annual GDD base 50F, IDW2 prediction",
                       "GDD50")
```



LOOCV also can be applied to IDW:

```
kcv.idw <- krige.cv(ANN_GDD50 ~ 1, locations=ne.m, model=NULL)
```

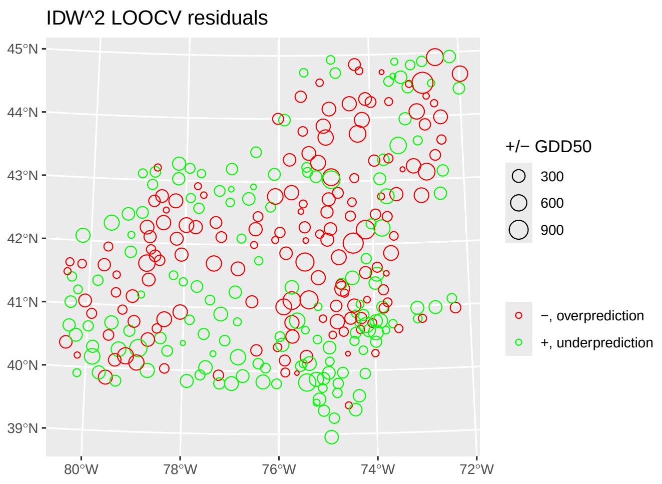
```
summary(kcv.idw$residual)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1183.393  -204.227    -2.284   -23.102   176.446   745.680

summary(kcv.ok$residual)

##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -1126.6606  -154.4425    -2.0428    -0.8718   176.5507   813.0746

(loocv.idw.rmse <- sqrt(sum(kcv.idw$residual^2)/length(kcv.idw$residual)))
```

```
## [1] 300.457

(loocv.ok.rmse)

## [1] 268.1796
```

The OK cross-validation is somewhat better than that from IDW, which makes sense, because of the fitted model of spatial dependence used in OK, vs. the *ad hoc* inverse-distance model.

Task 120 : Display a bubble plot of the LOOCV residuals. •

```
bubble.sf("kcv.idw", "residual", "GDD50", "IDW^2 LOOCV residuals")
```



These show much stronger residual spatial structure than do the OK residuals.

Challenge: Find the optimal power for IDW by testing several inverse-distance powers, and comparing their cross-validation statistics.

12.5 Thiessen polygons

The simplest way to predict a variable at one position is to use the value of that variable at the **nearest neighbour** in **geographic** space. For climate variables with a fairly dense network, as in this example, this is a common procedure: look at the climate record for the nearest station and consider that the local climate can not differ "too much".

A spatial expression of this is to divide the prediction area into **Thiessen polygons**[24], where each location is in a polygon whose centroid is its nearest neighbour. The advantage of this approach is that it requires no statistical

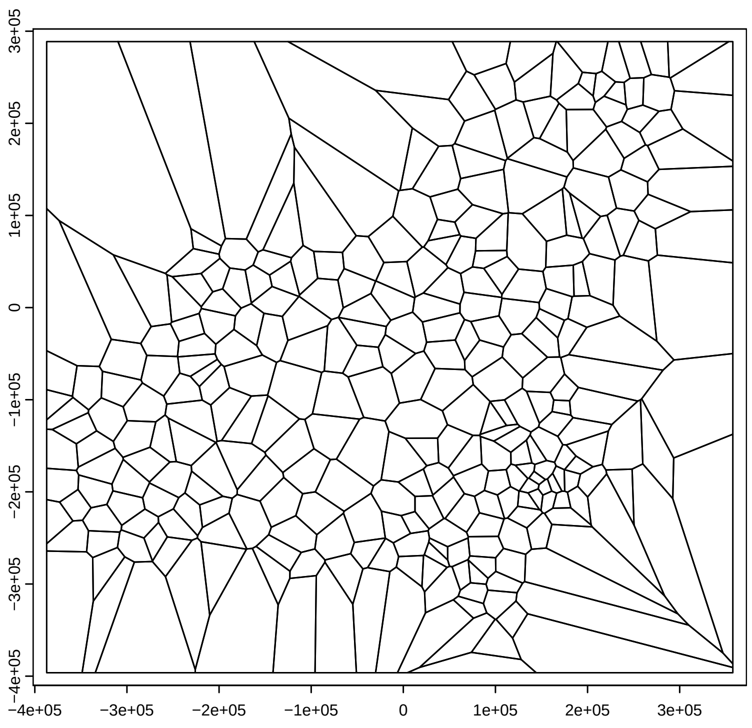
[24] also known as a Voronoi tessellation or a Dirichlet tessellation

model; in particular, there is no assumption of second-order stationarity as required by kriging.

Task 121 : Compute and display the Thiessen polygons over the study area. •

The computation is with the `voronoi` "Voronoi tesselation" function of the `sf` package. The optional `bnd` argument specifies the bounding box for the tesselation. We use the bounding box of the four States, converted to a `SpatVector`.

```
v <- terra::voronoi(vect(ne.m), bnd=vect(state.ne.m))
class(v)

## [1] "SpatVector"
## attr(,"package")
## [1] "terra"

plot(v)
```

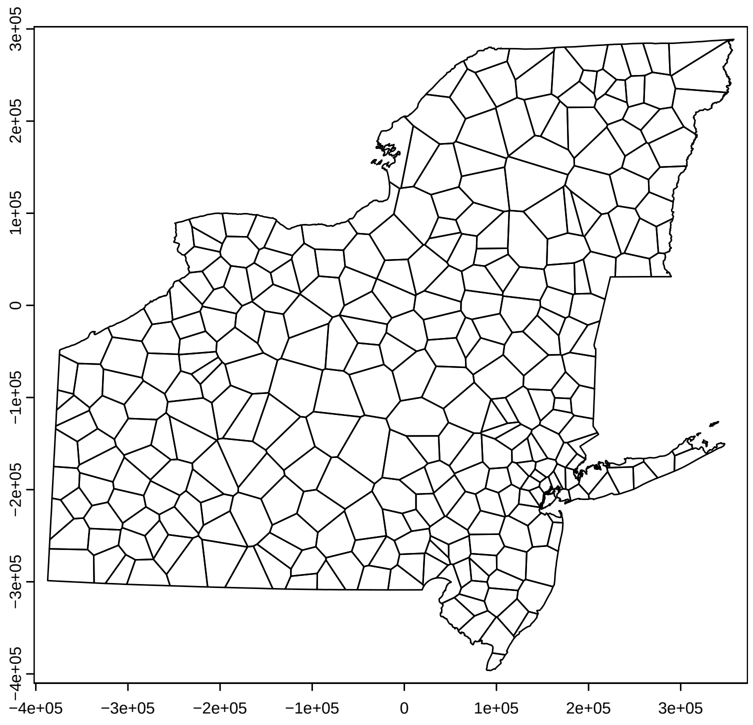


The Voronoi polygons extend to the boundary of the bounding box.

Task 122 : Clip the tesselation to the four states. •

For this we use the `crop` method with two `SpatVect` objects.

```
v <- crop(v, vect(state.ne.m))
plot(v)
```



This map now can be used for prediction. Simply, the entire area of each polygon is predicted with the value from the nearest station, i.e., its centroid.

Task 123 : Predict GDD50 over the study area by assigning the GDD50 from the centroid weather station to the entire polygon. •

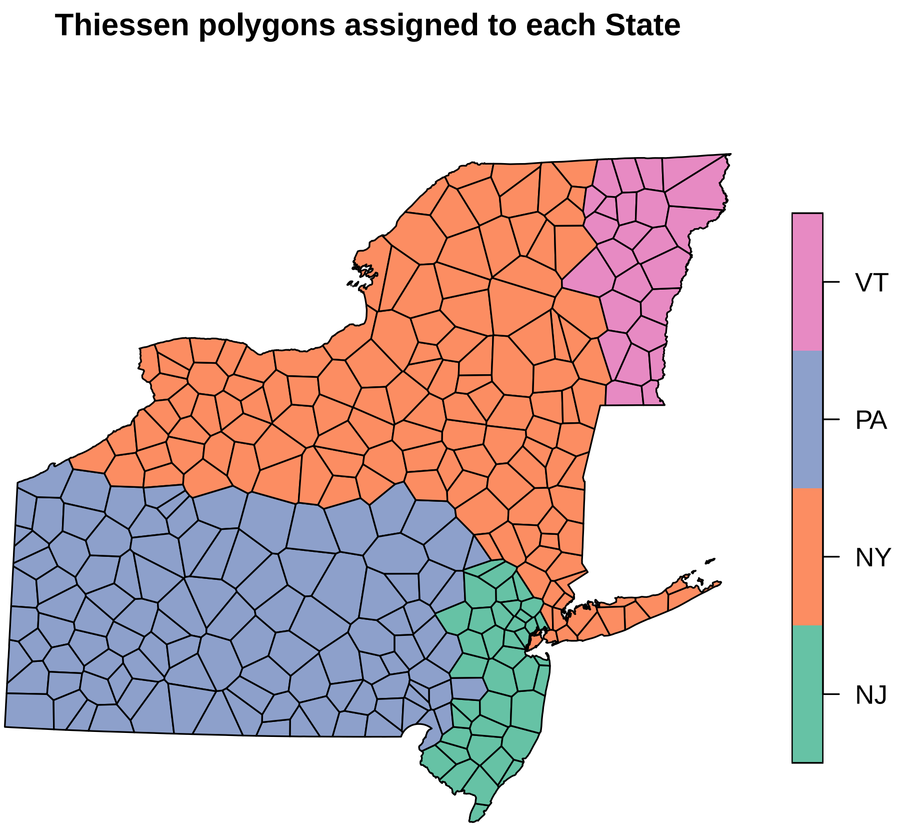
The `st_join` "spatial join" method of the `sf` package queries its second argument (layer from which attributes are queries) and assigns attribute values to the geometries in the first argument. In this case each polygon covers an area; within that area is only one climate station in the `sf` object `ne.m`, so the value of the attributes at that point will be assigned to the polygon.

> **Note:** For overlays where several points are in the same polygon, a user-specified function must be applied to return a single value.

```
class(v); dim(v)

## [1] "SpatVector"
## attr(,"package")
## [1] "terra"
## [1] 305  39

# convert from SpatVector to sf
v.sf <- st_as_sf(v)
summary(v.sf$geometry)

##  MULTIPOLYGON       POLYGON       epsg:NA +proj=aea ...
##             12          293            0            0
```

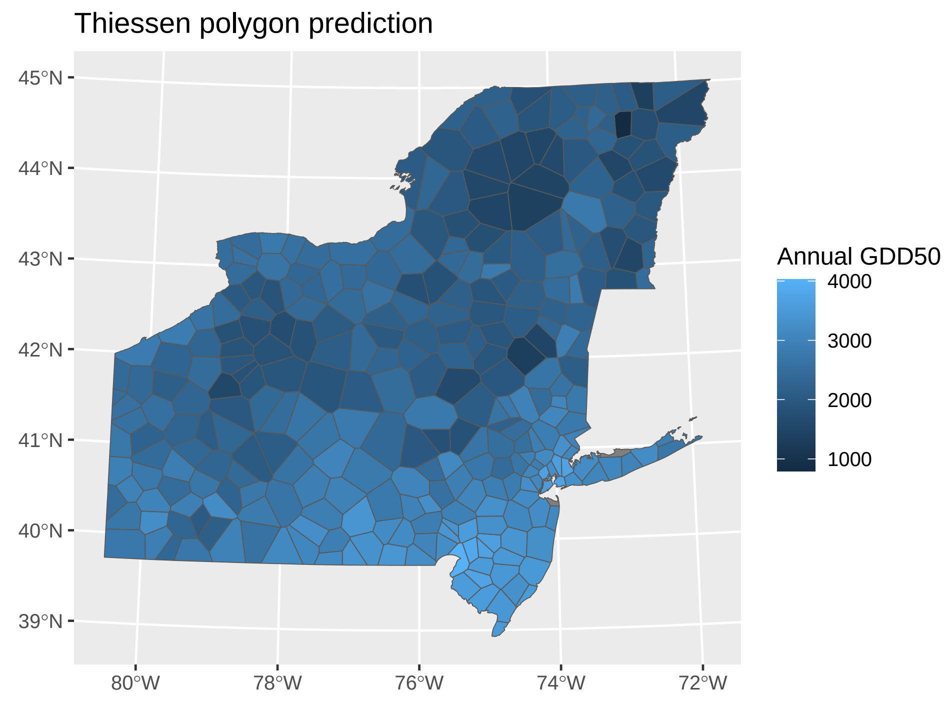
```
plot(v.sf["STATE"], main="Thiessen polygons assigned to each State")
# spatial join
v3 <- st_join(v.sf, ne.m)
class(v3)
```

```
## [1] "sf"         "data.frame"
```



Task 124 : Plot the Thiessen polygons with their predicted values. •

```
ggplot(v3) +
  geom_sf(aes(bg = ANN_GDD50.y)) +
  labs(bg = "Annual GDD50", title = "Thiessen polygon prediction")
```



To determine the predicted value at any location, just predict at that point.

To make the target point as a `sfc_POINT` we first create the point geometry with `st_point` and then make it a spatial object with Simple Features geometry with the `st_sfc` method. And of course we need to specify its CRS. We first specify the point location using geographic coördinates, and then we then transform to the CRS used in the grid. For illustration, we use the `geocode_OSM` function of the `tmaptools` package to retrieve the coördinates of a known address from the Open Street Map Nominatim database. Here we choose Cornell University's Musgrave Research Farm.

```
# an arbitrary point of interest
require(tmaptools)
(query.pt <- geocode_OSM("1256 Poplar Ridge Rd, Aurora, NY 13026"))

## $query
## [1] "1256 Poplar Ridge Rd, Aurora, NY 13026"
##
## $coords
##         x         y
## -76.65689  42.73498
##
## $bbox
##      xmin      ymin      xmax      ymax
## -76.65694  42.73493 -76.65684  42.73503

require(sf)
(pt <- st_sfc(st_point(query.pt$coords)))
```

```
## Geometry set for 1 feature
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -76.65689 ymin: 42.73498 xmax: -76.65689 ymax: 42.73498
## CRS:           NA

## POINT (-76.65689 42.73498)

class(pt)

## [1] "sfc_POINT" "sfc"

# pt <- st_sfc(st_point(x=c(-76.65689, 42.73498)))
st_crs(pt) <- 4326
pt <- st_transform(pt, st_crs(states.grid))
st_coordinates(pt)

##              X       Y
## [1,] -53753.43 26326.6

pt <- st_as_sf(pt)
class(pt)

## [1] "sf"         "data.frame"
```

Now predict at that point by a spatial join:

```
class(pt)

## [1] "sf"         "data.frame"

class(v3)

## [1] "sf"         "data.frame"

pt.pred <- st_join(pt, v3)
print(pt.pred["ANN_GDD50.y"])

## Simple feature collection with 1 feature and 1 field
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -53753.43 ymin: 26326.6 xmax: -53753.43 ymax: 26326.6
## Projected CRS: +proj=aea +lat_0=42.5 +lat_1=39
##          +lat_2=44 +lon_0=-76 +ellps=WGS84 +units=m
##   ANN_GDD50.y                         x
## 1        2590 POINT (-53753.43 26326.6)
```

The Thiessen polygon interpolation predicts 2590 annual GDD50 for this location.

12.5.1 Accuracy assessment

How accurate is a Thiessen polygon map? Ideally we would have a randomly-distributed sample of independent evaluation stations, but since we do not, we can apply the Thiessen polygon version of Leave-One-Out cross-validation (LOOCV). This is simply finding the nearest neighbour of each known station, and predicting at the known station from its neighbour.

Task 125 : Build a matrix of inter-station distances. •

This can be done with the `st_distance`‘spatial distances” function of the `sf` package.

```
dm <- st_distance(ne.m)
str(dm)

##  Units: [m] num [1:305, 1:305] 0 14343 64767 34083 159361 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:305] "1" "2" "3" "4" ...
##   ..$ : chr [1:305] "1" "2" "3" "4" ...
```

Task 126 : Find the nearest neighbour of each station, i.e., the one at closest distance. •

We use the `apply` function to apply the `which.min` "which minimum?" function across the rows of the distance matrix. However, all the diagonals are zero (distance between a station and itself), so we first have to put a large distance on diagonal so the stations themselves won't come out as minima.

```
diag(dm) <- max(dm)*1.1
nn <- apply(dm, 1, which.min)
str(nn)

##  Named int [1:305] 2 1 248 24 267 26 11 4 38 31 ...
##  - attr(*, "names")= chr [1:305] "1" "2" "3" "4" ...
```

This gives the index of each station's nearest neighbour. For example, station 1's nearest neighbour is station 2, and vice-versa.

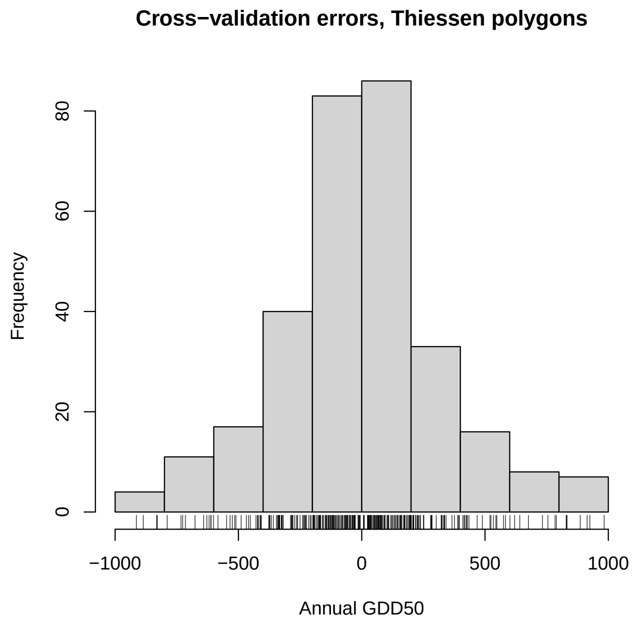
Task 127 : Predict each station from its nearest neighbour, and plot the regional predictions. •

```
nn.gdd <- ne.m[nn,"ANN_GDD50"]
str(nn.gdd)

## Classes 'sf' and 'data.frame': 305 obs. of  2 variables:
##  $ ANN_GDD50: num  3534 3310 3834 3553 2652 ...
##  $ geometry :sfc_POINT of length 305; first list element:  'XY' num  135231 -345526
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA
##   ..- attr(*, "names")= chr "ANN_GDD50"
```

Task 128 : Compare to the actual value and compute evaluation statistics. •

```
obs <- ne.m[,"ANN_GDD50"]
summary(diff <- st_drop_geometry(obs - nn.gdd))

##    ANN_GDD50           geometry
##  Min.   :-914.000   POINT   :305
##  1st Qu.:-193.000   epsg:NA:   0
##  Median :  -9.000
##  Mean   :  -4.393
##  3rd Qu.: 175.000
##  Max.   : 983.000

hist(diff$ANN_GDD50,
     xlab = "Annual GDD50",
     main="Cross-validation errors, Thiessen polygons")
rug(diff$ANN_GDD50)
(me <- mean(diff$ANN_GDD50))

## [1] -4.393443
```

```
(rmse <- sqrt(sum(diff$ANN_GDD50^2)/length(diff$ANN_GDD50)))

## [1] 332.5643
```



The mean error is close to zero, but the RMSE is quite large, 333 degree-days.

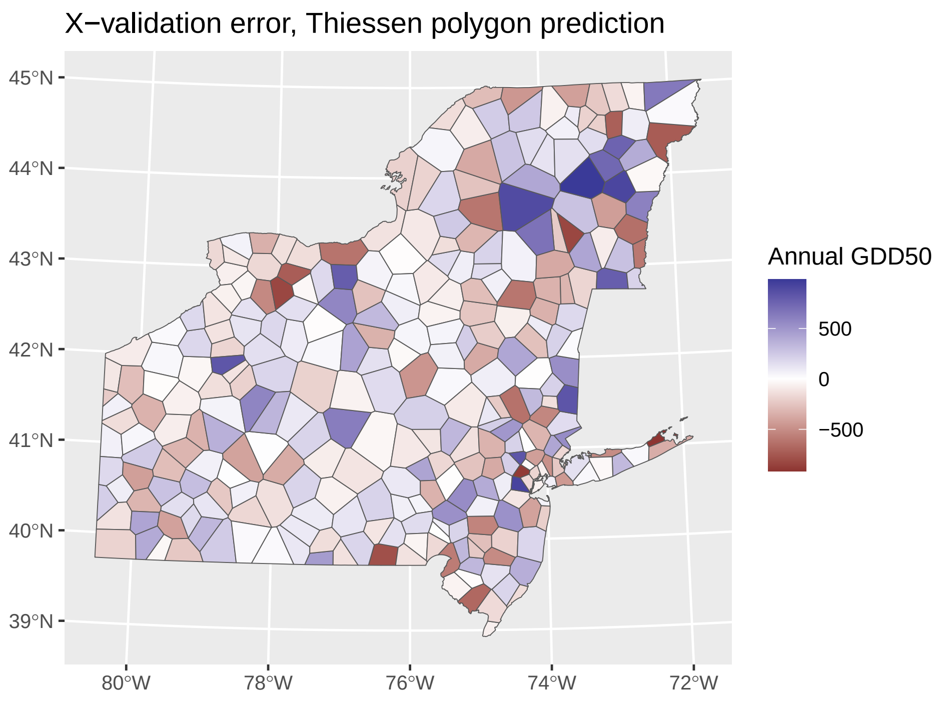
We can compare these to the evaluation statistics at the same points from the Random Forest out-of-bag computed above (§10.2.1).

```
rf.oob.me; rf.oob.rmse

## [1] -2.488285
## [1] 11.59619
```

Clearly, the predictions from Thiessen polygons, with this density of climate stations, are quite poor, compared to the data-driven model using coörd-inates and elevation.

Task 129 : Plot the LOOCV errors. •

```
v3$resid.thiessen <- diff$ANN_GDD50
ggplot(v3) +
  geom_sf(aes(bg = resid.thiessen)) +
  scale_fill_gradient2() +
  labs(bg = "Annual GDD50", title = "X-validation error, Thiessen polygon prediction")
```



The largest errors are where the topography changes rapidly between stations, for example at and next to Mt. Mansfield (VT) – these two polygons have the largest over- and under-predictions, respectively. The smallest errors are where elevations do not change much between neighbour stations, for example on the Lake Ontario plain.

13 Comparing the spatial patterns of two climate variables

An interesting question for geographers is how the spatial patterns of two climate variables are related. In the previous part of this exercise we analyzed growing degree days, base 50°F (GDDG50), which is related to the amount of heat available for crop development.

The source dataset from which the GDD50 records were taken includes other climate variables as 30-year records over the 1971-2000 time period. These are all provided as ESRI shapefiles, included in the compressed file `weather_stn_sums_1971_2000.zip`.

Task 130 : Locate this file, unpack it in a convenient working directory. •

Task 131 : List the shapefiles in this directory. •

The `list.files` function lists the files to the console output; you can also look at the directory in a file manager.

- Its first argument is the directory in which to look.
- The optional `pattern` argument gives a *regular expression* to match the file names. Here we just want the base shape files; there are other "helper" files with the same name but different file extensions, so we specify a pattern of the `shp` extension, at the end of the string, as symbolized by the special `$` regular expression character.

If the files are in the directory where you are connected[25], you can just specify `"."`, which is a Unix and R abbreviation for "the current directory". If they are in a subdirectory, you need to name that, using the forward slash `"/"` character to show you are descending the directory tree.

List the unpacked files:

```
list.files(".", pattern=".shp$")
```

```
## [1] "extmin_7100j.shp" "frz28_7100j.shp"  "frz32_7100j.shp"
## [4] "gdd40_7100j.shp"  "gdd50_7100j.shp"  "maat7100.shp"
## [7] "map7100.shp"
```

Each of these shapefiles has associated metadata, with extension `.xml` which can be read in several viewers.

```
list.files(".", pattern=".xml$")
```

```
## [1] "dem_ne_4km_TRI3_IDW2.sdat.aux.xml"
## [2] "extmin_7100j.shp.xml"
## [3] "frz28_7100j.shp.xml"
## [4] "frz32_7100j.shp.xml"
## [5] "gdd40_7100j.shp.xml"
## [6] "gdd50_7100j.shp.xml"
## [7] "maat7100.shp.xml"
## [8] "map7100.shp.xml"
## [9] "mrvbf_ne_4km.sdat.aux.xml"
```

There are several variables related to heat, in the following files:

gdd40_7100j :	growing degree days, base 40°F (applies to C3 crops such as spring wheat and barley)
maat7100 :	mean air temperature °F
frz32_7100j :	length of frost-free period, consecutive days above 32 °F (for frost-sensitive crops)
frz28_7100j :	length of frost-free period, consecutive days above 28 °F (for frost-tolerant crops)
extmin_7100j :	extreme minimum temperature °F

These all have annual and monthly records, averages over 1971-2000.

In addition, one variable is related to precipitation:

[25] use `getwd` to find this

map7100 : mean annual precipitation, inches[26].

13.1 Choose a variable to compare with annual GDD50

In this section we see how to compare regional maps to understand the spatial patterns of these differences.

We choose to compare mean annual temperature, °F, which is an overall measure of heat over the year, with annual GDD50. Since this is related to heat, the overall geographic pattern should be similar to annual GDD50, but there are physical differences between them.

Challenge: It is interesting to compare other variables, either annual or monthly. To do this, change the following code to substitute another variable for `maat7100`. If you use `map7100` (precipitation) you may want to convert inches to cm or mm.

Task 132 : Import the temperature records for the entire USA into a temporary data frame. •

This is the `maat7100` variable.

```
varname <- "maat7100"
tmp <- st_read(dsn=".", layer=varname,
               int64_as_string = FALSE, quiet = TRUE)
head(tmp)
```

```
## Simple feature collection with 6 features and 19 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -88.13 ymin: 31.3 xmax: -85.82 ymax: 34.8
## Geodetic CRS:  NAD27
##   STATION_ID STATE        STATION_NA LATITUDE_D LONGITUDE_ ELEVATION_
## 1      10160    AL    ALEXANDER CITY      32.95     -85.95        640
## 2      10178    AL        ALICEVILLE      33.13     -88.13        240
## 3      10252    AL      ANDALUSIA 3 W      31.30     -86.52        250
## 4      10272    AL ANNISTON METRO AP      33.58     -85.85        594
## 5      10369    AL       ASHLAND 3 SE      33.23     -85.82       1010
## 6      10395    AL           ATHENS 2      34.80     -86.98        720
##   JAN_TEMP FEB_TEMP MARCH_TEMP APRIL_TEMP MAY_TEMP JUNE_TEMP
## 1     43.0     46.6       54.3       61.1     68.9      76.0
## 2     42.7     46.8       55.0       61.6     70.0      77.4
## 3     48.4     51.5       58.4       64.5     72.1      78.6
## 4     43.3     47.3       54.9       61.7     69.4      76.6
## 5     42.7     46.6       53.9       60.9     68.3      75.0
## 6     38.2     42.1       50.7       59.1     67.5      74.7
##   JULY_TEMP AUG_TEMP SEP_TEMP OCT_TEMP NOV_TEMP DEC_TEMP ANN_TNORM_
## 1      79.5     78.6     73.4     62.8     53.7     45.7       62.0
## 2      80.9     79.8     74.3     63.0     53.3     45.4       62.5
## 3      81.2     80.7     76.8     66.3     57.7     50.8       65.6
## 4      80.1     79.5     73.8     62.9     53.4     46.1       62.4
## 5      78.4     77.7     72.8     62.5     54.0     45.8       61.6
## 6      78.5     77.6     71.2     59.9     49.9     41.5       59.2
##               geometry
## 1 POINT (-85.95 32.95)
## 2 POINT (-88.13 33.13)
## 3  POINT (-86.52 31.3)
## 4 POINT (-85.85 33.58)
## 5 POINT (-85.82 33.23)
## 6  POINT (-86.98 34.8)
```

[26] 1 inch = 2.54 cm

Task 133 : Restrict this dataframe to the the four selected states. •

```
ix <- (tmp$STATE %in% c("NY","NJ","PA","VT"))
tmp <- tmp[ix,]
names(tmp)

##  [1] "STATION_ID" "STATE"      "STATION_NA" "LATITUDE_D" "LONGITUDE_"
##  [6] "ELEVATION_" "JAN_TEMP"   "FEB_TEMP"   "MARCH_TEMP" "APRIL_TEMP"
## [11] "MAY_TEMP"   "JUNE_TEMP"  "JULY_TEMP"  "AUG_TEMP"   "SEP_TEMP"
## [16] "OCT_TEMP"   "NOV_TEMP"   "DEC_TEMP"   "ANN_TNORM_" "geometry"
```

Task 134 : Transform this frame to to the same metric CRS `ne.crs` loaded in §3. •

```
st_crs(tmp)$proj4string

## [1] "+proj=longlat +datum=NAD27 +no_defs"

tmp.m  <- st_transform(tmp, ne.crs)
st_crs(tmp.m)$proj4string

## [1] "+proj=aea +lat_0=42.5 +lon_0=-76 +lat_1=39 +lat_2=44 +x_0=0 +y_0=0 +ellps=WGS84 +units=m +no_d
```

13.2 Add the variable to compare

Task 135 : Transfer the temperature records to the same object with the GDD50, and then remove the temporary objects. •

We use the `cbind` "bind columns" function for this.

```
names(tmp.m)

##  [1] "STATION_ID" "STATE"      "STATION_NA" "LATITUDE_D" "LONGITUDE_"
##  [6] "ELEVATION_" "JAN_TEMP"   "FEB_TEMP"   "MARCH_TEMP" "APRIL_TEMP"
## [11] "MAY_TEMP"   "JUNE_TEMP"  "JULY_TEMP"  "AUG_TEMP"   "SEP_TEMP"
## [16] "OCT_TEMP"   "NOV_TEMP"   "DEC_TEMP"   "ANN_TNORM_" "geometry"

ne.m$ANN_TNORM_ <- tmp.m$ANN_TNORM_
rm(tmp, tmp.m)
```

Add these coördinates as regular fields, for linear models which use them as predictors.

```
ne.coords <- st_coordinates(ne.m)
ne.m <- cbind(ne.m, E = ne.coords[,1], N = ne.coords[ , 2])
rm(ne.coords)
```

13.3 Standardize variables for comparison

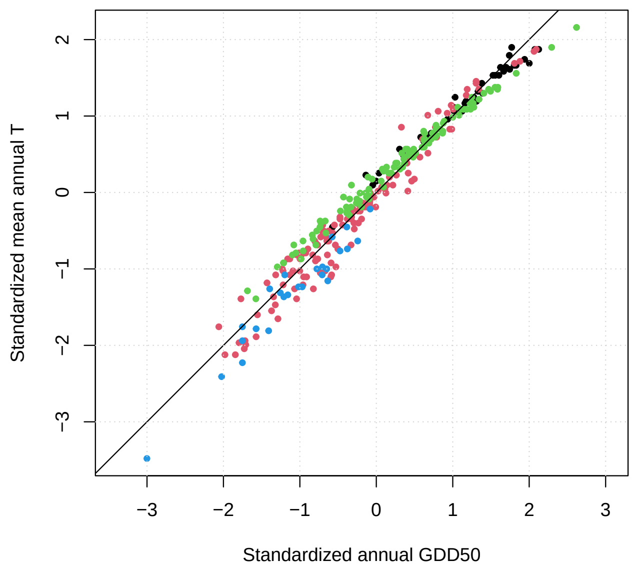
To compare two variables on the same scale, it is necessary to **standardize** them, by subtracting the mean and dividing by the standard deviation.

Task 136 : Compute standardized versions of the `GDD50` and `ANN_TNORM_` fields. •

```
summary(ne.m$ANN_GDD50)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     795    2100    2463    2518    2930    4021

summary(ne.m$ANN_TNORM_)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   34.70   45.20   47.80   48.03   51.00   56.30

gdd50.std <- (ne.m$ANN_GDD50 - mean(ne.m$ANN_GDD50))/sd(ne.m$ANN_GDD50)
t.ann.std <- (ne.m$ANN_TNORM_ - mean(ne.m$ANN_TNORM_))/sd(ne.m$ANN_TNORM_)
summary(gdd50.std)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.00125 -0.72747 -0.09499  0.00000  0.71869  2.61961

summary(t.ann.std)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -3.4800 -0.7388 -0.0600  0.0000  0.7754  2.1591
```

Task 137 : Compare these to see how closely they are correlated. •

```
cor(ne.m$ANN_GDD50, ne.m$ANN_TNORM_)

## [1] 0.9811989

cor(gdd50.std, t.ann.std)

## [1] 0.9811989

plot(gdd50.std, t.ann.std, asp=1,
     col = ne.m$STATE, pch = 20,
     xlab="Standardized annual GDD50",
     ylab="Standardized mean annual T")
abline(0,1); grid()
```



Note that the correlation is the same for both the standardized and unstandardized variables. It's interesting that the GDD50 are a bit higher than the mean annual T at both extremes of the 1:1 plot. So they are closely correlated, but not identical.

We choose to use RK-GLS (§7) and Random Forests (§10.2) as the prediction methods; the other methods could all be used.

13.4 Comparing variables with RK-GLS

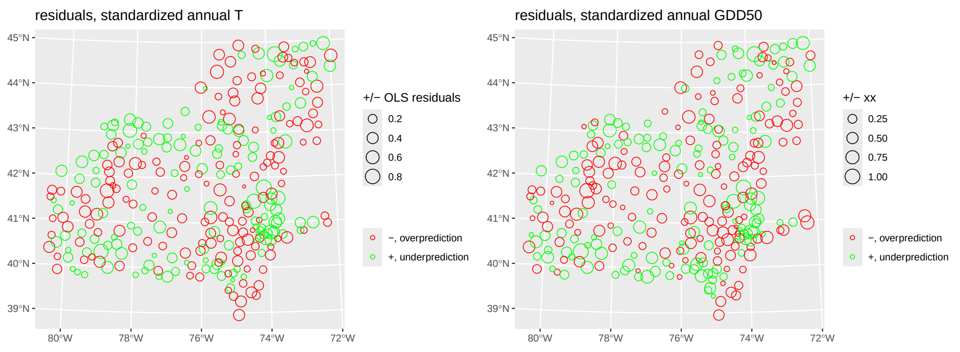
Task 138 : Build OLS models of both standardized variables from the two coördinates and square root of elevation. Compare their adjusted R^2 and coefficients. •

```
summary(m.ols.t <- lm(t.ann.std~ sqrt(ELEVATION_)+N+E, data=ne.m))

##
## Call:
## lm(formula = t.ann.std ~ sqrt(ELEVATION_) + N + E, data = ne.m)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6861 -0.1944 -0.0391  0.1842  0.8651
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.096e+00  5.505e-02  19.913  < 2e-16 ***
## sqrt(ELEVATION_) -5.321e-02  1.829e-03 -29.088  < 2e-16 ***
## N                -3.614e-06  1.199e-07 -30.148  < 2e-16 ***
## E                -9.150e-07  1.134e-07  -8.071 1.68e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2821 on 301 degrees of freedom
## Multiple R-squared:  0.9212,Adjusted R-squared:  0.9204
## F-statistic:  1173 on 3 and 301 DF,  p-value: < 2.2e-16

summary(m.ols.gdd <-  lm(gdd50.std ~ sqrt(ELEVATION_)+N+E, data=ne.m))

##
## Call:
## lm(formula = gdd50.std ~ sqrt(ELEVATION_) + N + E, data = ne.m)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9447 -0.2365 -0.0221  0.2303  1.0474
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.374e+00  6.709e-02  20.483  < 2e-16 ***
## sqrt(ELEVATION_) -6.184e-02  2.230e-03 -27.735  < 2e-16 ***
## N                -2.895e-06  1.461e-07 -19.814  < 2e-16 ***
## E                -9.386e-07  1.382e-07  -6.793 5.88e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3439 on 301 degrees of freedom
## Multiple R-squared:  0.8829,Adjusted R-squared:  0.8818
## F-statistic: 756.6 on 3 and 301 DF,  p-value: < 2.2e-16

coef(m.ols.t)

##      (Intercept) sqrt(ELEVATION_)                N                E
##     1.096150e+00    -5.320974e-02    -3.613649e-06    -9.149859e-07

coef(m.ols.gdd)

##      (Intercept) sqrt(ELEVATION_)                N                E
##     1.374233e+00    -6.183727e-02    -2.894689e-06    -9.386282e-07

coef(m.ols.t)/coef(m.ols.gdd)

##      (Intercept) sqrt(ELEVATION_)                N                E
##        0.7976448        0.8604801        1.2483723        0.9748119
```

Q45 : *Do the two models explain the same amount of spatial variability by the same predictors? If the two variables have the same spatial structure, what should be the ratio of the coefficients? Is that the case here?* *Jump to A45* •

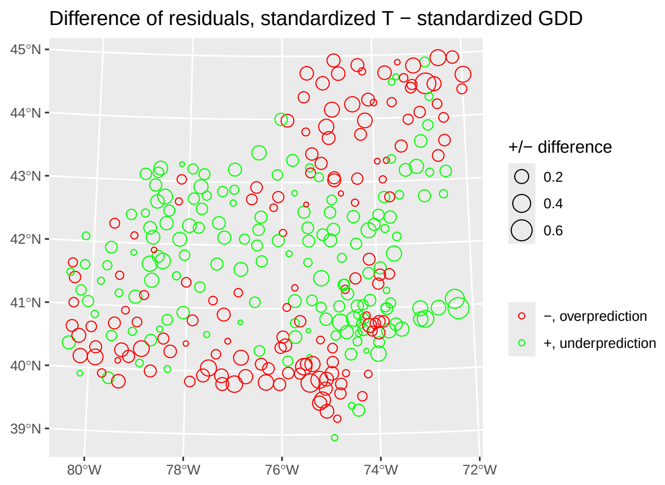
Task 139 : Display bubble plots of the residuals, and 1:1 plots of actual vs. fitted, for both variables. •

```
summary(ne.m$ols.resid.t <- residuals(m.ols.t))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.6861 -0.1944 -0.0391  0.0000  0.1842  0.8651

summary(ne.m$ols.resid.gdd <- residuals(m.ols.gdd))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.9447 -0.2365 -0.0221  0.0000  0.2303  1.0474

p1 <- bubble.sf("ne.m", "ols.resid.t", "OLS residuals",
             "residuals, standardized annual T")
p2 <- bubble.sf("ne.m", "ols.resid.gdd", "xx",
             "residuals, standardized annual GDD50")
require(gridExtra)
grid.arrange(p1, p2, ncol = 2)
```



Task 140 : Compare the residuals with a bubble plot of their differences. •

```
summary(ne.m$ols.resid.diff <- ne.m$ols.resid.t - ne.m$ols.resid.gdd)

##       Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.583312 -0.103252  0.005628  0.000000  0.101299  0.610121

bubble.sf("ne.m", "ols.resid.diff", "difference",
       "Difference of residuals, standardized T - standardized GDD")
```



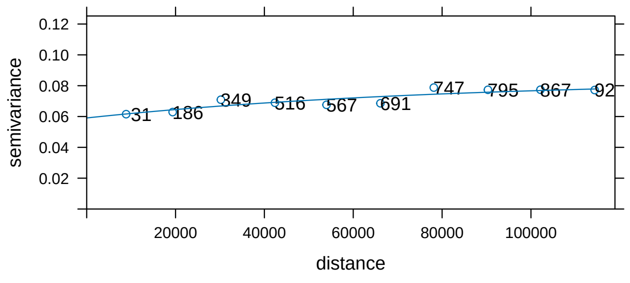
Now we begin to see the pattern, where the regional trend of annual mean temperature is higher or lower than that for GDD50.

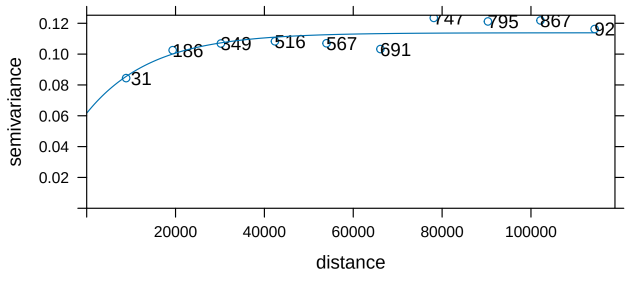
Q46 : *What is the pattern of differences between the residuals from the models for the two variables? Identify the most interesting areas. What could be an explanation?* *Jump to A46* •

Task 141 : Model the residual spatial dependence for both variables, i.e., fit variogram models to the OLS residuals, for both variables. •

```
require(gstat)
v.r.ols.t <- variogram(ols.resid.t ~ 1,
                       locations=ne.m, cutoff=120000, width=12000)
(vmf.r.ols.t <- fit.variogram(v.r.ols.t,
                              vgm(psill=0.1, model="Exp",
                                  range=20000, nugget=0.02)))

##   model      psill   range
## 1   Nug 0.05907886     0.0
## 2   Exp 0.02479735 80781.2

#
v.r.ols.gdd <- variogram(ols.resid.gdd ~ 1,
                       locations=ne.m, cutoff=120000, width=12000)
(vmf.r.ols.gdd <- fit.variogram(v.r.ols.gdd,
                              vgm(psill=0.12, model="Exp",
                                  range=20000, nugget=0.02)))

##   model      psill    range
## 1   Nug 0.06167406     0.00
## 2   Exp 0.05216480 14545.31

#
# a common y-axis scale
ymax <- max(sum(vmf.r.ols.t[,"psill"]), sum(vmf.r.ols.gdd[,"psill"]))*1.1
p1 <- plot(v.r.ols.t, pl=T, model=vmf.r.ols.t, ylim = c(0, ymax))
```

```
p2 <- plot(v.r.ols.gdd, pl=T, model=vmf.r.ols.gdd, ylim = c(0, ymax))
print(p1, split=c(1,1,1,2), more=T)
print(p2, split=c(1,2,1,2), more=F)
```





Q47 : *How strong is the local spatial dependence of the residuals from the two trend surfaces? Do the two trend surfaces have the same residual local spatial structure? If not, what is the difference? What does that imply about the trend surface model and spatial structure of the variables? Jump to A47* •

Task 142 : Use this estimated spatial dependence among the residuals to re-fit the models with GLS. •

We estimate starting values for the proportional nugget from the variogram fit.

```
# require(nlme)
require(nlme)
(p.nugget <- vmf.r.ols.t[1,"psill"]/sum(vmf.r.ols.t[,"psill"]) + 0.001)

## [1] 0.7053578

m.gls.t <- gls(model=t.ann.std ~ sqrt(ELEVATION_) + N + E,
               data=ne.m,
               correlation=corExp(
                 value=c(vmf.r.ols.t[2,"range"], p.nugget),
                 form=~E + N,
                 nugget=TRUE))
#
(p.nugget <- vmf.r.ols.gdd[1,"psill"]/sum(vmf.r.ols.gdd[,"psill"]) + 0.001)

## [1] 0.5427663

m.gls.gdd <- gls(model=gdd50.std ~ sqrt(ELEVATION_) + N + E,
             data=ne.m,
             correlation=corExp(
               value=c(vmf.r.ols.gdd[2,"range"], p.nugget),
               form=~E + N,
               nugget=TRUE))
summary(m.gls.t)

## Generalized least squares fit by REML
##   Model: t.ann.std ~ sqrt(ELEVATION_) + N + E
##   Data: ne.m
##        AIC      BIC    logLik
##   159.9663 185.9161 -72.98317
##
## Correlation Structure: Exponential spatial correlation
##  Formula: ~E + N
##  Parameter estimate(s):
##        range       nugget
## 9.451200e+04 6.735544e-01
##
## Coefficients:
##                       Value  Std.Error   t-value p-value
## (Intercept)       1.0352866 0.07976232  12.97964   0e+00
## sqrt(ELEVATION_) -0.0510409 0.00217065 -23.51411   0e+00
## N                -0.0000035 0.00000027 -12.71021   0e+00
## E                -0.0000010 0.00000025  -4.02844   1e-04
##
##  Correlation:
##                  (Intr) s(ELEV N
## sqrt(ELEVATION_) -0.708
## N                 0.323 -0.184
## E                -0.232  0.246 -0.335
##
## Standardized residuals:
##         Min          Q1         Med          Q3         Max
## -2.46015918 -0.62877289 -0.04851122  0.67873651  2.94128997
##
## Residual standard error: 0.2964007
## Degrees of freedom: 305 total; 301 residual

summary(m.gls.gdd)

## Generalized least squares fit by REML
##   Model: gdd50.std ~ sqrt(ELEVATION_) + N + E
##   Data: ne.m
##        AIC     BIC    logLik
##   274.5262 300.476 -130.2631
##
## Correlation Structure: Exponential spatial correlation
##  Formula: ~E + N
##  Parameter estimate(s):
##        range       nugget
## 7.741961e+04 6.175653e-01
##
## Coefficients:
##                       Value  Std.Error    t-value p-value
## (Intercept)       1.2724192 0.09520248  13.365400   0e+00
## sqrt(ELEVATION_) -0.0579702 0.00265657 -21.821480   0e+00
## N                -0.0000027 0.00000033  -8.161348   0e+00
## E                -0.0000011 0.00000030  -3.696353   3e-04
##
##  Correlation:
##                  (Intr) s(ELEV N
## sqrt(ELEVATION_) -0.734
## N                 0.333 -0.188
## E                -0.246  0.254 -0.346
##
## Standardized residuals:
##          Min            Q1          Med           Q3          Max
## -2.847476935 -0.591834392  0.001698496  0.684143298  2.811395719
```

```
##
## Residual standard error: 0.3611266
## Degrees of freedom: 305 total; 301 residual
```

Task 143 : Compare the model coefficients. •

```
coefficients(m.gls.t)

##     (Intercept) sqrt(ELEVATION_)               N               E
##    1.035287e+00    -5.104088e-02   -3.489755e-06   -1.018102e-06

coefficients(m.gls.gdd)

##     (Intercept) sqrt(ELEVATION_)               N               E
##    1.272419e+00    -5.797018e-02   -2.688525e-06   -1.117387e-06

coefficients(m.gls.t)/coefficients(m.gls.gdd)

##     (Intercept) sqrt(ELEVATION_)               N               E
##       0.8136364        0.8804679       1.2980184       0.9111453
```

This shows the ratio of the coefficients for the two variables. If they would have the same regional spatial structure (trend surface), these values would all be 1.

Task 144 : Compare the correlation structures fit by REML. •

```
intervals(m.gls.t)$corStruct

##               lower         est.        upper
## range  2.349567e+04 9.451200e+04 3.801773e+05
## nugget 4.280832e-01 6.735544e-01 8.504693e-01
## attr(,"label")
## [1] "Correlation structure:"

intervals(m.gls.gdd)$corStruct

##               lower         est.        upper
## range  1.947769e+04 7.741961e+04 3.077262e+05
## nugget 4.071888e-01 6.175653e-01 7.915103e-01
## attr(,"label")
## [1] "Correlation structure:"

intervals(m.gls.t)$corStruct/intervals(m.gls.gdd)$corStruct

##           lower     est.    upper
## range  1.206286 1.220776 1.235440
## nugget 1.051314 1.090661 1.074489
## attr(,"label")
## [1] "Correlation structure:"
```

Again, all these ratios would be 1 if the structures were identical. The mean annual temperature has proportionally longer range and higher nugget than GDD50. The range parameters are 77–95 km, for an effective range of about 240–300 km; this is quite a bit longer than what we estimated by eye from the residual variograms from the OLS fit.

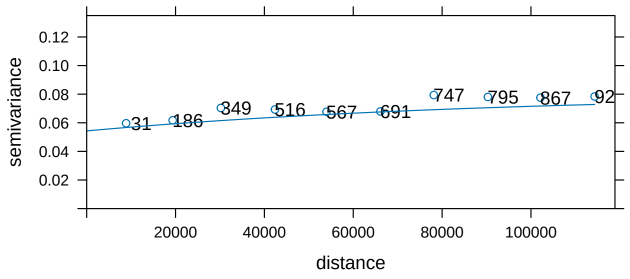
Task 145 : Add the GLS model residuals to the spatial data frame. •

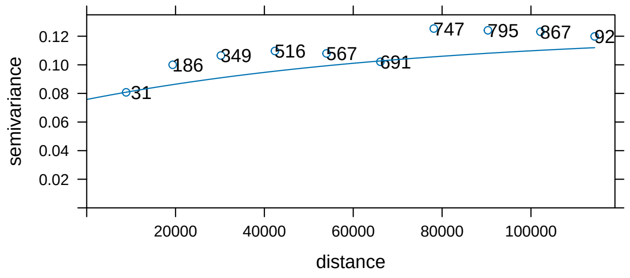
```
ne.m$gls.resid.t <- residuals(m.gls.t)
ne.m$gls.resid.gdd <- residuals(m.gls.gdd)
```

Task 146 : Display the fitted model of spatial correlation with the empirical variogram of the GLS residuals. •

We first convert the correlation structure found by `gls` to a variogram model; the partial sill is estimated as the variance of the residuals, adjusted for the proportional nugget. We estimate the nugget as a proportion of this total sill.

```
(p.nugget <- intervals(m.gls.t)$corStruct["nugget","est."])

## [1] 0.6735544

(t.sill <- var(ne.m$gls.resid.t))

## [1] 0.08065724

(vmf.r.gls.t <- vgm(psill=t.sill*(1-p.nugget), model="Exp",
                    range=intervals(m.gls.t)$corStruct["range","est."],
                    nugget=t.sill*p.nugget))

##   model      psill range
## 1   Nug 0.05432704     0
## 2   Exp 0.02633020 94512

v.r.gls.t <- variogram(gls.resid.t ~ 1,
                         locations=ne.m, cutoff=120000, width=12000)
#
(p.nugget <- intervals(m.gls.gdd)$corStruct["nugget","est."])

## [1] 0.6175653

(t.sill <- var(ne.m$gls.resid.gdd))

## [1] 0.1226937

(vmf.r.gls.gdd <- vgm(psill=t.sill*(1-p.nugget), model="Exp",
                    range=intervals(m.gls.gdd)$corStruct["range","est."],
                    nugget=t.sill*p.nugget))

##   model      psill    range
## 1   Nug 0.07577136     0.00
## 2   Exp 0.04692231 77419.61

v.r.gls.gdd <- variogram(gls.resid.gdd ~ 1,
                         locations=ne.m, cutoff=120000, width=12000)
ymax <- max(sum(vmf.r.gls.t[,"psill"]), sum(vmf.r.gls.gdd[,"psill"]))*1.1
p1 <- plot(v.r.gls.t, model=vmf.r.gls.t, pl=T, ylim = c(0, ymax))
p2 <- plot(v.r.gls.gdd, model=vmf.r.gls.gdd, pl=T, ylim = c(0, ymax))
print(p1, split=c(1,1,1,2), more=T)
print(p2, split=c(1,2,1,2), more=F)
```





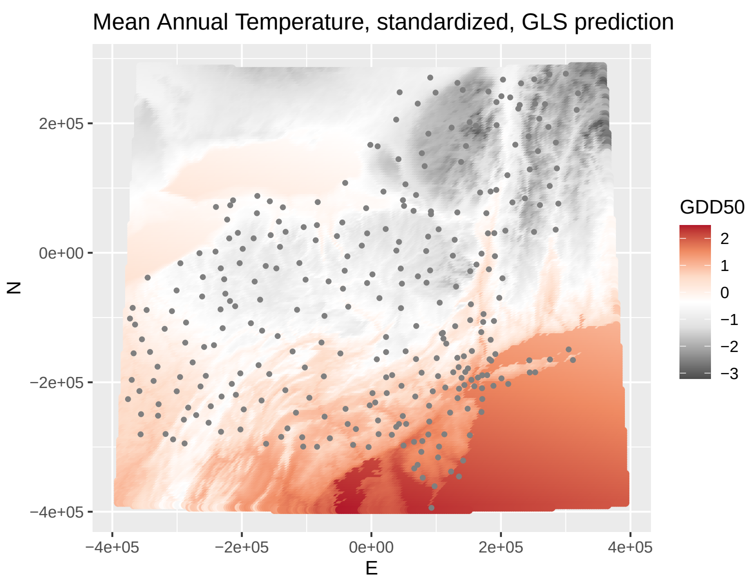
Task 147 : Predict over the regional grid with the GLS model and add the result to the dataframe. •

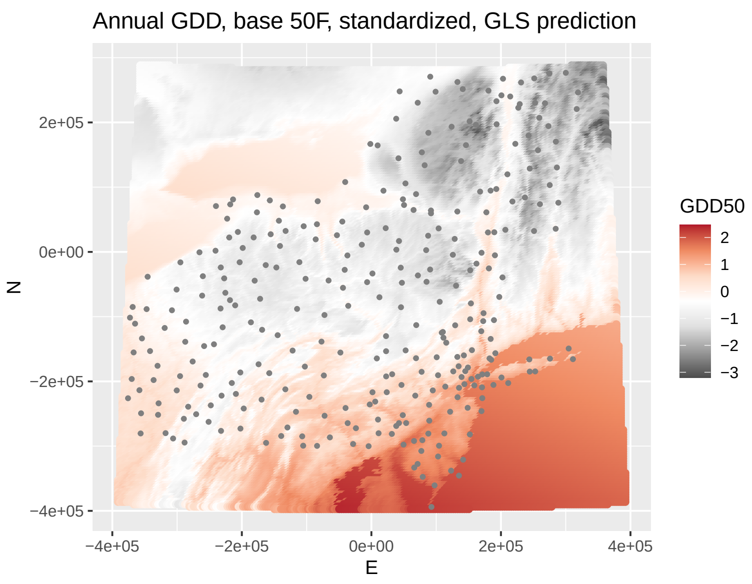
```
dem.ne.m.df$pred.gls.t <- predict(m.gls.t, newdata=dem.ne.m.df)
dem.ne.m.df$pred.gls.gdd <- predict(m.gls.gdd, newdata=dem.ne.m.df)
dem.ne.m.df$diff.gls.t.gdd <-
    dem.ne.m.df$pred.gls.t - dem.ne.m.df$pred.gls.gdd
```

Task 148 : Plot the two regional predictions on the same visual scale. •

We use a different palette from the non-standardized prediction maps, to emphasize that these are standardized values.

```
(std.pred.lim <- c(min(dem.ne.m.df[,c("pred.gls.t","pred.gls.gdd")]),
                   max(dem.ne.m.df[,c("pred.gls.t","pred.gls.gdd")])))

## [1] -3.203491  2.471781

display.prediction.map("pred.gls.t",
                       "Mean Annual Temperature, standardized, GLS prediction",
                       "GDD50", std.pred.lim, .palette="RdGy")
display.prediction.map("pred.gls.gdd",
                       "Annual GDD, base 50F, standardized, GLS prediction",
                       "GDD50", std.pred.lim, .palette="RdGy")
```



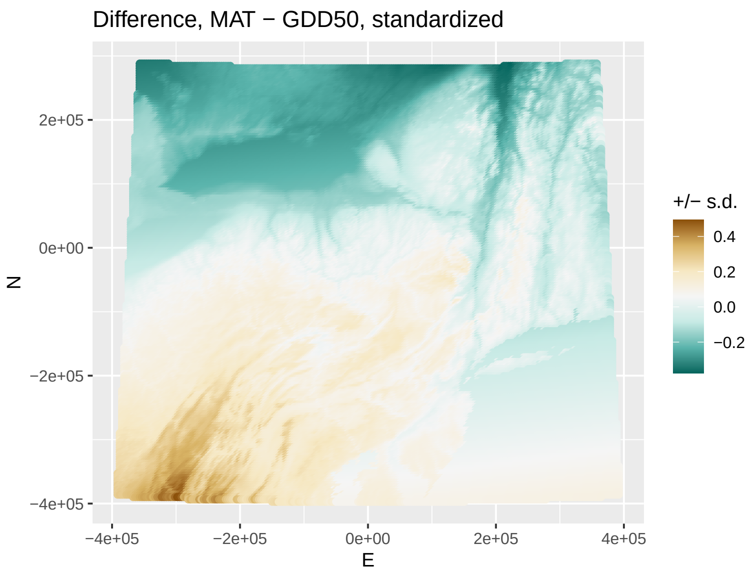


Task 149 : Compute the differences between the two GLS predictions, add them to the data frame, summarize them, and display them as a map. •

```
summary(dem.ne.m.df$diff.gls.t.gdd <-
            dem.ne.m.df$pred.gls.t - dem.ne.m.df$pred.gls.gdd)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.376528 -0.120566  0.016410 -0.007091  0.102386  0.492284
```

```
display.difference.map("diff.gls.t.gdd",
                       "Difference, MAT - GDD50, standardized",
                       "+/- s.d.",
                       .palette="BrBG")
```



Q48 : *Describe the spatial distribution of the differences between the MAT and GDD50 predictions.* *Jump to A48* •

Although the GLS model residuals do not have strong spatial structure, still we can krige them to improve the maps.

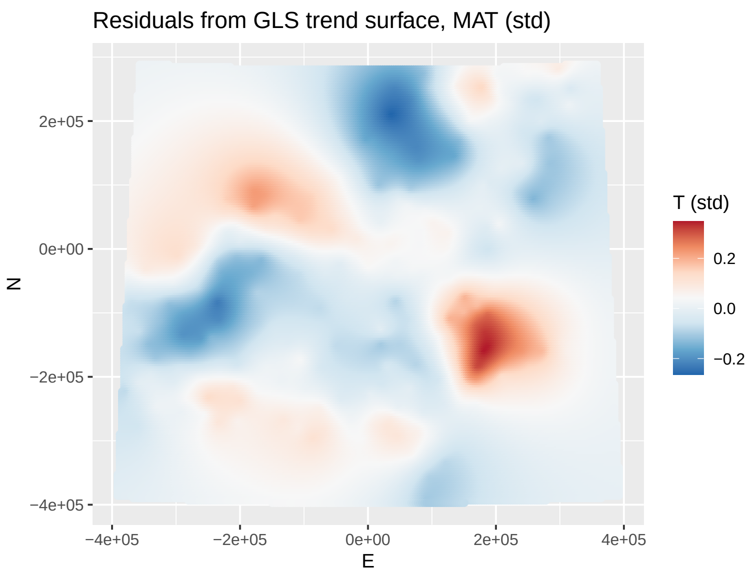
Task 150 : Predict the deviations from the GLS trend surface at each location on the grid, using Ordinary Kriging (OK) of the GLS residuals for both standardized variables; display their summary, and display as maps. •

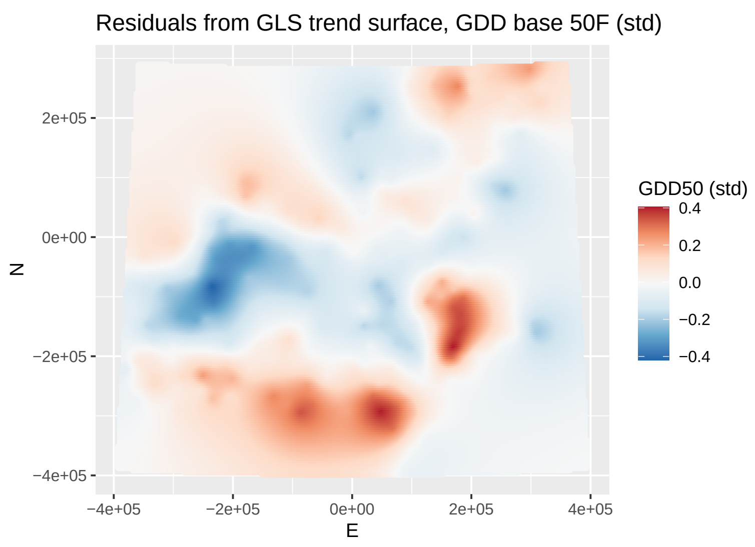
Recall we build variogram models of the residuals from the correlation structures found by `gls`.

```
ok.gls.resid.t <- krige(gls.resid.t ~ 1, loc=ne.m, newdata=dem.ne.m.sf,
                        model=vmf.r.gls.t)

## [using ordinary kriging]

ok.gls.resid.gdd <- krige(gls.resid.gdd ~ 1, loc=ne.m, newdata=dem.ne.m.sf,
                          model=vmf.r.gls.gdd)

## [using ordinary kriging]
```

```
dem.ne.m.df$ok.gls.resid.t <- ok.gls.resid.t$var1.pred
dem.ne.m.df$ok.gls.resid.gdd <- ok.gls.resid.gdd$var1.pred
```

```
ggplot() +
    geom_point(aes(x=E, y=N, colour=ok.gls.resid.t), data=dem.ne.m.df) +
    xlab("E") + ylab("N") + coord_fixed() +
    ggtitle("Residuals from GLS trend surface, MAT (std)") +
    scale_colour_distiller(name="T (std)", space="Lab", palette="RdBu")
ggplot() +
    geom_point(aes(x=E, y=N, colour=ok.gls.resid.gdd), data=dem.ne.m.df) +
    xlab("E") + ylab("N") + coord_fixed() +
    ggtitle("Residuals from GLS trend surface, GDD base 50F (std)") +
    scale_colour_distiller(name="GDD50 (std)", space="Lab", palette="RdBu")
```

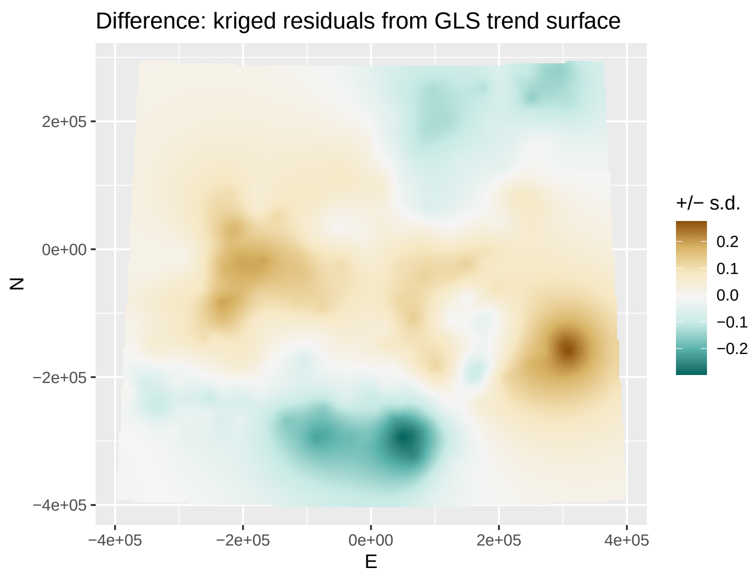




Task 151 : Display the differences in the OK of the standardized residuals. ●

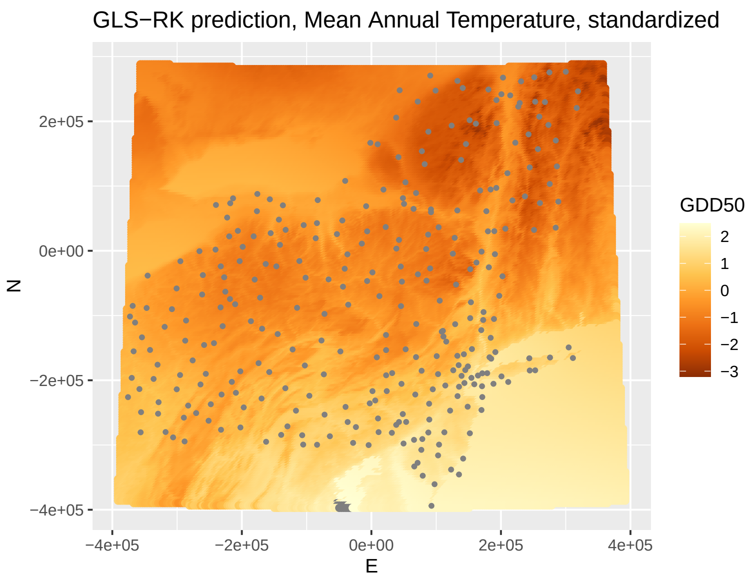
This shows where each variable is more or less adjusted.

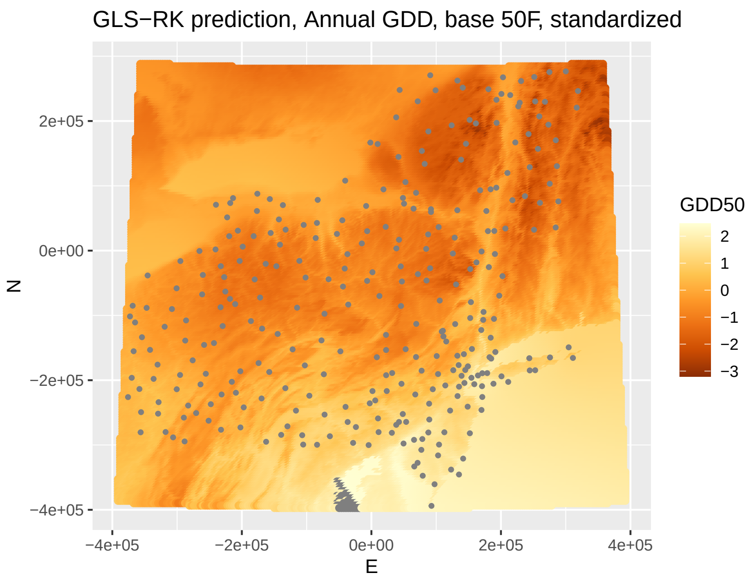
```
summary(dem.ne.m.df$ok.gls.resid.diff.t.gdd <-
            dem.ne.m.df$ok.gls.resid.t - dem.ne.m.df$ok.gls.resid.gdd)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.297577 -0.044848  0.020110  0.009595  0.065114  0.273665

display.difference.map("ok.gls.resid.diff.t.gdd",
                       "Difference: kriged residuals from GLS trend surface",
                       "+/- s.d.",
                       .palette="BrBG")
```



Task 152 : Add the kriged GLS residuals to the trend surfaces for a final GLS-RK prediction. Display the two maps. •

```
summary(dem.ne.m.df$pred.rkgls.std.t <-
            dem.ne.m.df$pred.gls.t + dem.ne.m.df$ok.gls.resid.t)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.16100 -0.89334 -0.20840 -0.08172  0.65080  2.50018

summary(dem.ne.m.df$pred.rkgls.std.gdd <-
            dem.ne.m.df$pred.gls.gdd + dem.ne.m.df$ok.gls.resid.gdd)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.16605 -0.87200 -0.23809 -0.08422  0.53944  2.51504

display.prediction.map("pred.rkgls.std.t",
                       "GLS-RK prediction, Mean Annual Temperature, standardized",
                       "MAAT  (std)", std.pred.lim, .palette="YlOrBr")
display.prediction.map("pred.rkgls.std.gdd",
                       "GLS-RK prediction, Annual GDD, base 50F, standardized",
                       "GDD50 (std)", std.pred.lim, .palette="YlOrBr")
```

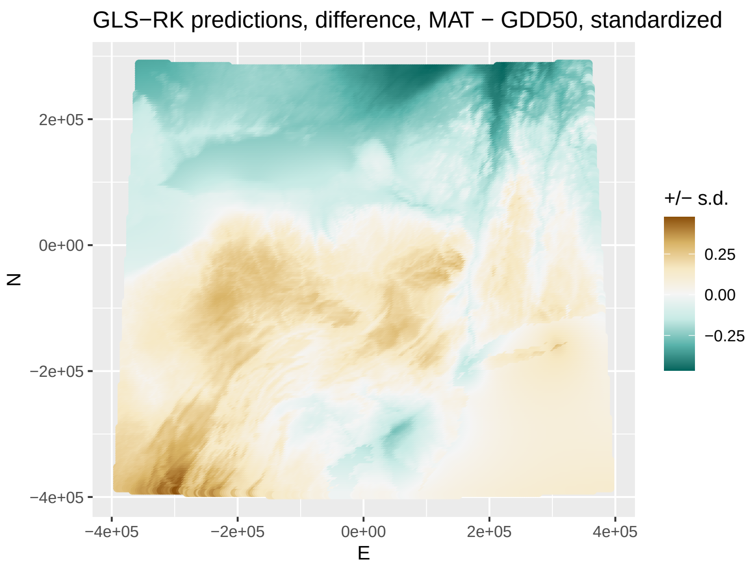




Task 153 : Compute and display the difference between the two GLS-RK predictions. ●

```
#
summary(dem.ne.m.df$diff.rkgls.std.t.gdd <-
              (dem.ne.m.df$pred.rkgls.std.t - dem.ne.m.df$pred.rkgls.std.gdd))
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.462388 -0.120160  0.037457  0.002504  0.124340  0.477189

display.difference.map("diff.rkgls.std.t.gdd",
                       "GLS-RK predictions, difference, MAT - GDD50, standardized",
                       "+/- s.d.",
                       .palette="BrBG")
```



There is a considerable difference between the two normalized variables. Normalized MAT is higher than normalized GDD50 in the Appalachian, Taconic and Allegheny mountains, and on eastern Long Island; the reverse is the case in Adirondacks and Green Mountains, and especially in the Lake Champlain valley. To the north, the mean annual temperature is lowered by the very cold winter months. If the comparison were with growing season mean temperature, perhaps the results would be more similar.

13.5 Comparing variables with Random Forests

Another way to compare the variables is with their RF models. This does not show linear model coefficients or local spatial correlation structure, but does show variable importance, and also produces two maps.

Task 154 : Build RF models of the two normalized variables. •

```
require(randomForest)
m.rf.std.t <- randomForest(t.ann.std ~ ELEVATION_ + N + E + dist.lakes + dist.coast,
                     data=ne.df, ntree=1200,
                     importance=TRUE)
m.rf.std.gdd <- randomForest(gdd50.std ~ ELEVATION_ + N + E,
                     data=ne.df, ntree=1200,
                     importance=TRUE)
```

Task 155 : Compare the variable importance of the two models. •

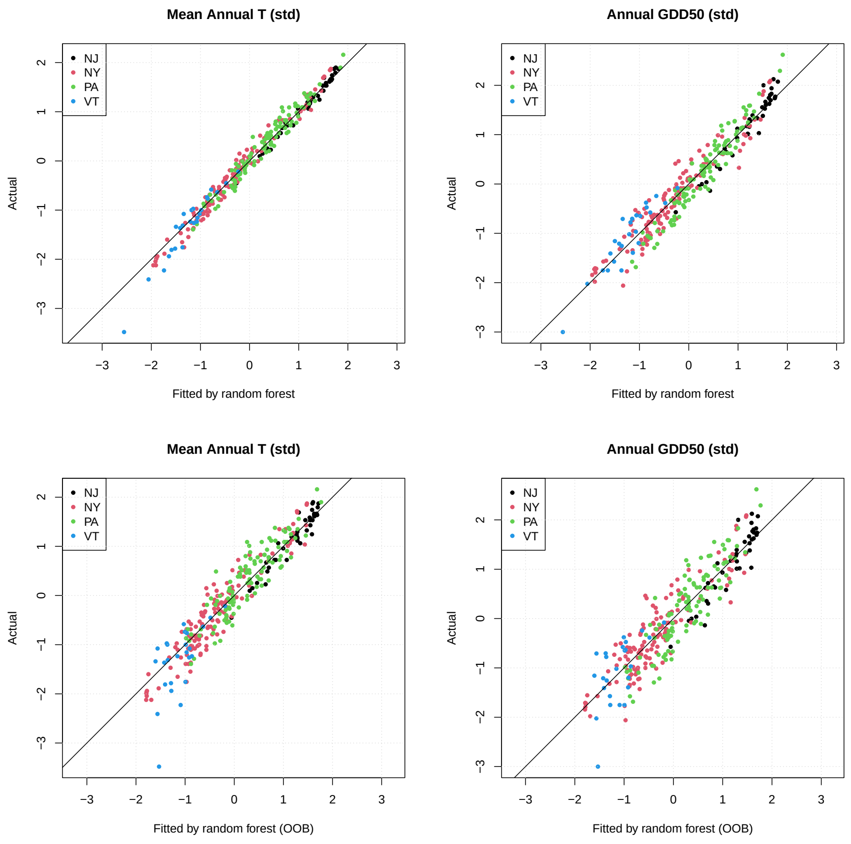
We use the `importance` function of the `randomForest` package:

```
randomForest::importance(m.rf.std.t)

##              %IncMSE IncNodePurity
## ELEVATION_ 44.40325      58.15060
## N          45.70317      80.86923
## E          44.97693      36.19765
## dist.lakes 38.95843      65.83409
## dist.coast 29.96851      56.13357

randomForest::importance(m.rf.std.gdd)

##              %IncMSE IncNodePurity
## ELEVATION_ 71.93775     120.02959
## N          82.20071     130.47469
## E          55.55530      45.31475
```

The Northing is more influential in the MAT model than in the GDD50 model, whereas the elevation is slightly more influential in the GDD50 model. This agrees with the results of the GLS model (§13.4), where the absolute Northing coefficient was larger for the MAT model, and the absolute elevation coefficient larger for the GDD50 model.

Task 156 : Plot the two fits, and the two OOB fits, side-by-side. •

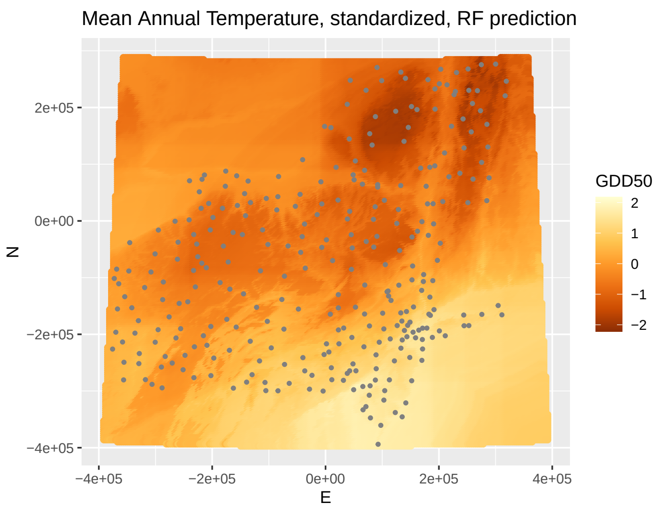
```
plot(t.ann.std ~ predict(m.rf.std.t, newdata=ne.m),
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by random forest", ylab="Actual",
     main="Mean Annual T (std)")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid(); abline(0,1)
plot(gdd50.std ~ predict(m.rf.std.t, newdata=ne.m),
     col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by random forest", ylab="Actual",
     main="Annual GDD50 (std)")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid(); abline(0,1)
plot(t.ann.std ~ predict(m.rf.std.t), col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by random forest (OOB)", ylab="Actual",
     main="Mean Annual T (std)")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid(); abline(0,1)
plot(gdd50.std ~ predict(m.rf.std.t), col=ne.m$STATE, pch=20, asp=1,
     xlab="Fitted by random forest (OOB)", ylab="Actual",
     main="Annual GDD50 (std)")
legend("topleft", levels(ne.m$STATE), pch=20, col=1:4)
grid(); abline(0,1)
```

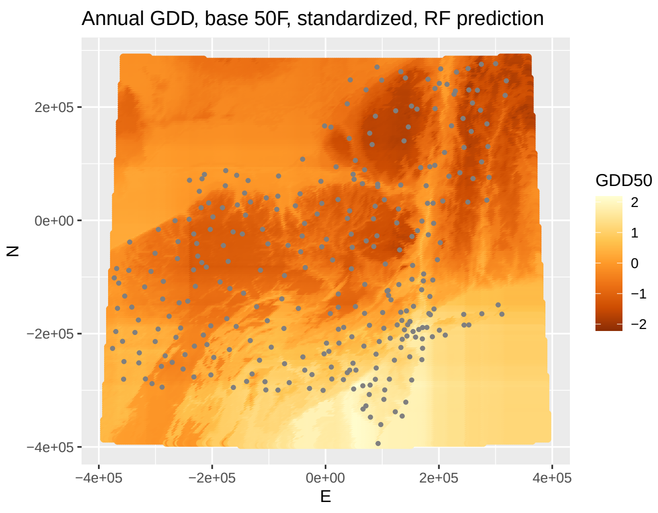


There is not much difference between these. The GDD50 for Mount Mansfield (VT) is better fit than the MAT.

Task 157 : Predict over the regional grid with both RF models, and display the maps. ●

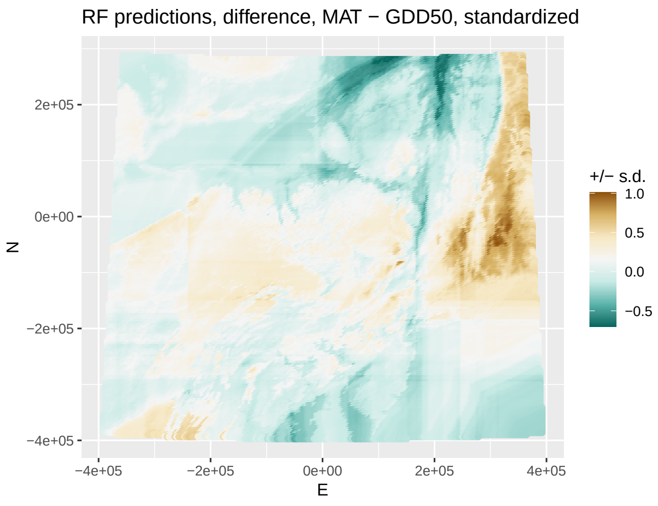
```
dem.ne.m.df$pred.rf.std.t <- predict(m.rf.std.t, newdata=dem.ne.m.df)
dem.ne.m.df$pred.rf.std.gdd <- predict(m.rf.std.gdd, newdata=dem.ne.m.df)
(std.pred.lim <- c(min(dem.ne.m.df[,c("pred.rf.std.t","pred.rf.std.gdd")]),
                       max(dem.ne.m.df[,c("pred.rf.std.t","pred.rf.std.gdd")])))

## [1] -2.207555  2.176656

display.prediction.map("pred.rf.std.t",
                       "Mean Annual Temperature, standardized, RF prediction",
                       "degrees C", std.pred.lim, .palette="YlOrBr")
display.prediction.map("pred.rf.std.gdd",
                       "Annual GDD, base 50F, standardized, RF prediction",
                       "GDD50", std.pred.lim, .palette="YlOrBr")
```





Task 158 : Compute the differences between the two maps and display the difference map. •

```
summary(dem.ne.m.df$diff.rf.std.t.gdd <-
    dem.ne.m.df$pred.rf.std.t - dem.ne.m.df$pred.rf.std.gdd)

##      Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.69765 -0.06407  0.06807  0.08525  0.21273  1.01038

display.difference.map("diff.rf.std.t.gdd",
                       "RF predictions, difference, MAT - GDD50, standardized",
                       "+/- s.d.",
                       .palette="BrBG")
```



The summary show that half the differences between the standardized variables are quite small, about ±0.115 standard deviations. This agrees with the close correlation between the variables.

This map shows similar differences between the variables as the GLS model difference map (§13.4): normalized MAT is higher than normalized GDD50 in the Appalachian, Taconic and Allegheny mountains, and on eastern Long Island; the reverse is the case in Adirondacks and Green Mountains, and especially in the Lake Champlain valley.

14 Answers

A1 : *There is an obvious regional trend in the N-S dimension: degree-days tend to decrease going North, as shown by the smaller symbols in that direction. There does not seem to be a consistent trend in the E-W dimension, There is also a trend with elevation: degree-days tend to decrease as elevation increases, at the same N latitude.* *Return to Q1* •

A2 :

`Northing` : *This relation looks linear at the more southerly portions of the map (NJ and PA except the northern tier of counties) but quite spread out, with a less obvious relation, in the northerly portion (NY and VT, northern PA).*

`Easting` : *Note the much wider spread of GDD in the East, which ranges from southern NJ to northern VT, than in the continental climate of the West. Easting does not appear to be predictive of the expected value of GDD – it appears that a linear regression would have a (near) zero slope.*

`ELEVATION_` : *There is a clear relation (higher elevations have fewer GDD) but it appears inverse-parabolic rather than linear.*

Return to Q2 •

A3 : *Elevation has the strongest correlation; Northing is not much weaker. The correlation coefficients are both negative: GDD50 decreases with increasing Northing and elevation. There is essentially no relation of GDD50 with Easting.* *Return to Q3* •

A4 : *Square root of elevation explains a bit more than half (55.6%) of the total variation in GDD50.* *Return to Q4* •

A5 : *In general there is a good fit: most points are near the 1:1 actual:fit line. However two NY points (red) are poorly fit, one underfit at* $\approx (1300, 1950)$ *(fit, actual) and one overfit at* $\approx (2700, 1700)$. *Return to Q5* •

A6 :

The model diagnostics look pretty good: (1) no relation between fits and residuals; (2) approximately equal variance at all fitted values; (3) residuals mostly normally-distributed; (4) the high-leverage point is consistent with the others. However two points are severely under-predicted (positive residuals) and one severely over-predicted (negative residual). *Return to Q6*
•

A7 :

There is obvious correlation, with some reasonable geographic interpretation. The model under-predicts GDD along the Lake Erie and Ontario plains (this due to the moderating effect of the lake) and over-predicts along the Atlantic coast, southeastern VT and the northern tier of PA. The Atlantic coast may stay cooler in spring than indicated by its very low elevation and southerly position. The model also under-predicts in SE PA and SW NJ (the Philadelphia area) where there may be influence from warm southerly winds. *Return to Q7* •

A8 : *There appears to be some structure up to* ≈ 40 *km.* *Return to Q8* •

A9 : *The confidence interval of the range parameter seems quite wide, almost double from the lower limit* 1.285×10^4 *to the upper limit* 2.3715×10^4. *Return to Q9* •

A10 :

The range of spatial dependence has been adjusted; from the variogram fit we estimated in §4.4, i.e., $1.2107{\times}10^4$; *the REML estimate is somewhat longer,* 1.7457×10^4 *m. These are 1/3 of the effective range, since we fit an exponential model. The effective range found by* `gls` *is thus* 5.2371×10^4 *m.* *Return to Q10* •

A11 : *The GLS fit does not remove spatial correlation, it just takes it into account* *when computing the regression parameters.* Return to Q11 •

A12 : *This fits reasonably well; we conclude that we've detected the true correlation structure of the residuals.* Return to Q12 •

A13 :

The coefficients have changed a small amount; that for Northing was reduced by over 2%. This shows that some clustered far N and far S elevation points had higher leverage in the OLS model than the cluster warranted. Return to Q13 •

A14 : *We see the effect of the reduced coefficient for Northing: the GLS predicts somewhat lower in the N, so the residuals are lower than the OLS residuals (red circles in the bubble plot); the reverse is true in the S.* Return to Q14 •

A15 : *There is quite some adjustment along the Great Lakes plain (NW) and near some cities (additional GDD) and the Atlantic coast (lower GDD). There are several "hotspots" of large adjustments near single climate stations.* Return to Q15 •

A16 : *These areas are beyond the range of spatial correlation, here about 50 km, and so are not affected by "nearby" observations.* Return to Q16 •

A17 : *Compared to the GLS surface this has more detail and is adjusted locally, for example along the Lake Ontario plain. The discrepancies (points that can be seen on the map) are much smaller.* Return to Q17 •

A18 : *Recall that the fitted variogram model shows local spatial dependence only to an effective range of 52 km, so that there is no local adjustment further than this from any point. Hardly any area of the bordering States or Province is within this distance of an observation point.*

The other issue is the extrapolation of the linear trend. All areas are within the elevation range, so that is not a problem. But for areas further North or South, we are assuming the linear trend continues unchanged. Return to Q18 •

A19 : *(1) The global trend on the covariates is adequate to compute residuals and their spatial structure; we do not expect it to change much if re-fit locally. (2) It is not practical because the trend surface would have to be re-computed, the residuals extracted, and the variogram re-fit at 1000's of prediction points; this would have to be done automatically without the possibility of checking for artefacts. In local KED we can use the single fitted residual variogram model, while the trend is adjusted locally and the predictions are dependent only on observations in the local neighbourhood. The model is not changed, the results of applying it are.*

Note that with very dense point networks (e.g., precision agriculture) this procedure *is applicable, and implemented by the VESPER computer program*[27] *for Ordinary Kriging, but not KED.* *Return to Q19* •

A20 : *The largest over-predictions by local KED are extrapolations, outside the area with points, e.g., west side of Lake Ontario, and the south-central Appalachians. Within the interpolation area, global KED is generally a bit higher, especially in north-central PA. This may be due to a stronger Northing effect in the global model, vs. the effect as found in local neighbourhoods.* *Return to Q20* •

A21 : *Local KED with the 61 neighbours is considerably better: Cross-validation RMSE is about 4% lower and the extreme errors are smaller.*

This shows that both the effect of the covariates and any local effects represented by the stations are not uniform across the area, and some form of localized prediction is indicated. *Return to Q21* •

A22 : *The differences are certainly geographically-consistent. The global model gives more positive residuals (excessively high predictions) in the southwest, Lake Ontario plain, and far North Country of NY and VT. It gives more negative residuals (excessively low predictions) in the east. This shows that the local fit substantially changed the predictions.* *Return to Q22* •

A23 : *The marginal relations are not well-fit by linear relations, although the square root of elevation is nearly linear, as we saw in the OLS/GLS modelling. However, in GAM modelling we do not need to select a single transformation over the whole range of a predictor to linearize the relation. One transformation may not be applicable to the whole range. Instead, we allow the smooth fit to determine the local adjustment.* *Return to Q23* •

A24 : *The model fits well; the adjusted R^2 is 0.908, and the residuals are less spread than those from the OLS and GLS models. The effective degrees of freedom, i.e., accounting for the many local regressions in the splines, was 23.53 for the 2D surface, and 8.52 for the 1D relation with elevation.*

Return to Q24 •

A25 : *Both of these plots support the conclusion that there is no spatial dependence of the model residuals. (1) The bubbles appear to be randomly distributed, both in colour and size. (2) The variogram shows the same spatial correlation at all separations.* *Return to Q25* •

A26 : *The GAM 2D trend clearly differs from the linear trend surface, especially in the Lake Ontario plain (towards the right centre in the figure). It is also also higher than a linear trend in the S Hudson valley, but lower along the Atlantic shore (upper left in the figure). These are areas we identified with large OLS and GLS residuals in the linear trend surfaces (§4, §5).* *Return to Q26* •

[27] https://sydney.edu.au/agriculture/pal/software/vesper.shtml

A27 : *This is almost linear now, as suggested by physical theory, once the smooth geographic trend is removed. However at the low elevations there is a wide spread of GDD50, which is well-fit by an almost vertical portion of the marginal smooth function.* *Return to Q27* •

A28 : *The largest differences are to the E and W, out of the calibration area – this illustrates that GAM should not be extrapolated. The much lower GDD50 in New England is because the GAM trend surface was considerably below the GLS surface along the Atlantic coast, and this was extrapolated eastward. The GAM predicts higher values along the Great Lakes plains and lower Hudson valley, as we saw in the GLS residuals.* *Return to Q28* •

A29 : *The largest differences are quite local, around certain weather stations where the local deviation from the trend could be accounted for in RK-GLS, but was somewhat averaged out in GAM.* *Return to Q29* •

A30 : *The first (root) splitting variable is N. The split is at* -1.55967×10^5 *N. The mean value of GDD50 of the whole dataset is 2517.52; the mean value of the observations in the left branch (less than) is 2182.54 and of the right branch (greater than) is3019.99. These branches have 183 and 122 observations, respectively.* *Return to Q30* •

A31 : *Northing is most important; it explains almost 50% of the variance. Then elevation explains about another 40%, and Easting very little. This agrees with the linear correlations computed as preparation for fitting the OLS model.* *Return to Q31* •

A32 : *Each run of the* `rpart` *function will give the same tree (if the same parameters are specified) but slightly different cross-validation statistics. The cross-validation error reaches an effective minimum around 15 splits, CP about* 0.0045*. So building the tree with CP=*0.003 *was overfitting.* *Return to Q32* •

A33 : *The pruned tree has the same root and higher levels, but fewer splits and leaves.* *Return to Q33* •

A34 : *There are 16 unique values predicted by the pruned regression tree. The fit to the actual values is better the OLS or GLS models; the RMSE from the regression tree is 11.16 GDD50; for the GLS model 12.08 In this case the linear model predicts more values but on average they deviate more from the true values.* *Return to Q34* •

A35 : *The regression tree divides the area into "blocks" mostlt with the Northing but in one place with the Easting. It also slices most of the "blocks" according to elevation zones. These give the maximum between-group variance at the leaves of the regression tree, without overfitting.* *Return to Q35* •

A36 : *Permuting either elevation or Northing leads to a large change in the predictions; Easting much less.* *Return to Q36* •

A37 : *The out-of-bag mean errors are from 2 to 3 times that of the fits. This is a typical result for random forests. The OOB errors are indicative of the errors at unknown points, i.e., prediction accuracy.* *Return to Q37* •

A38 : *For most runs we see no or weak residual spatial structure, because of the averaging effect of the repeated bootstrap sampling; in many trees close point-pairs lose one of the points.* *Return to Q38* •

A39 : *The RF surface shows some irregular patches and abrupt transitions, whereas the RK-GLS surface is by construction smooth.* *Return to Q39* •

A40 : *The RF prediction is from the "box" containing the elevation (all the same in the lake), Northing and Easting, which was fit with the most similar points. These are presumably along the Lake shore. There is no extrapolation via a trend surface to modify the effect of the coördinates in the model.* *Return to Q40* •

A41 : *The GLS and GLS-RK models have coefficients for the coördinates, which here are far East, so the predictions change. The RF does not have any information in this area and so puts it all in the "boxes".* *Return to Q41* •

A42 : *There are no points in OH so the prediction by RF is made from the fitted model of the nearby PA stations. These apparently use the Easting.* *Return to Q42* •

A43 : *The RF model has no way to extrapolate to higher or lower elevations than in the calibration set. In the Alleghenies and Catskills there are a lot of higher-elevation area beyond the elevation of weather stations. In the Adirondacks the stations are all at low elevations, but apparently the Northing is here used to predict the GDD. This is why the RF over-predicts in the lowlands around Plattsburgh and Lake Champlain.* *Return to Q43* •

A44 : *The random forest model is clearly not suitable to show a regional trend, especially outside of the model calibration area. It does allow for non-linearities and local combinations of factors, for example on the Lake Ontario and Erie plains.* *Return to Q44* •

A45 : *Slightly more variation of annual temperature is explained by the model. Since these are both standardized, if their regional relation with the predictors is the same, so should the coefficients. They are close but not identical.* *Return to Q45* •

A46 *: There is a clear spatial pattern and local spatial dependence. The residuals from the MAT model are lower than those from the GDD50 model in the N and S edges, and at the higher elevations (Adirondacks, Green Mountains, Allegheny Plateau). The reverse is the case in the centre and especially on Long Island (NY) and northern NJ.*

Return to Q46 •

A47 *:*

The variogram structures are both weak (short range, very high nugget proportion). That is, the regional trend explains most of the variation in both cases.

There is weaker residual spatial structure for mean annual temperature than for GDD50 (lower total sill, higher nugget proportion). This implies that the MAT trend (the predictors N, E and elevation) explains more of the spatial variability. Note however that the MAT trend surface has a lower R^2 than the GDD50 trend surface. This implies that MAT is more explained regionally and less by local variations than annual GDD50.

Return to Q47 •

A48 *: The MAT is proportionally higher than GDD50 in the higher elevations, especially towards the SW. The opposite effect is seen in the lowlands, especially the Lake Champlain valley (NE).*

Return to Q48 •

15 Challenge

Do a similar analysis *either* for:

- Over the same study area as the example:
  - the growing degree days in one of the growing-season months: May through September; or
  - the annual growing degree days at base 40°F
  - one of the other climate variables in one of the other shapefiles.
- Over one of the States within the study area.
- Over some other region of the USA:
  - the growing degree days base 50°F, i.e., the same as used in this example. Note that for this you will have to define a suitable coördinate reference system (CRS) for that area.

If you choose a different study area, you will need to re-build the points database and prediction grid, as explained in the companion tutorial "Tutorial: Regional mapping of climate variables from point samples Data preparation".

If you have to define a suitable CRS:

- For E-W oriented regions, you can uses the same Albers Equal Area projection as was used in this tutorial, but the parameters will be different.
- For N-S oriented regions, you will need to select a different projection. A good choice is Transverse Mercator, PROJ.4 name `tmerc`. The parameters for this are[28]:

```
+proj=tmerc +lat_0=Latitude of natural origin
            +lon_0=Longitude of natural origin
            +k=Scale factor at natural origin
            +x_0=False Easting
            +y_0=False Northing
```

1. Determining which covariables (elevation, Northing, Easting, their transformations or interaction) best model the target variable;
2. Estimating the coefficients of the linear model with OLS;
3. Examining the (non-spatial) OLS model diagnostics;
4. Modelling the spatial structure of the OLS model residuals;
5. Estimating the coefficients of the linear model and the spatial correlation structure (nuisance parameters) with GLS;
6. Fitting a random forest with all three predictors.

[28] `http://geotiff.maptools.org/proj_list/transverse_mercator.html`, see the 'PROJ.4 organization' subheading for the names and meanings of the parameters

7. Mapping the various predictions and their differences.

Then answer these questions:

1. Is the linear model justified?
2. Is there spatial structure in the OLS model residuals? So, is a GLS model needed?
3. How much do the OLS and GLS model coefficients differ?
4. What is the spatial correlation structure fitted by the GLS model? Does this agree with that estimated from the OLS variogram?
5. How does your model compare to that developed in this exercise for annual GDD50?
6. What could be the reason(s) for differences between the model in the exercise and your model?

References

[1] D Bates. Fitting linear mixed models in R. *R News*, 5(1):27–30, 2005. 28

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth, 1983. 80

[3] I. C. Briggs. Machine contouring using minimum curvature. *Geophysics*, 39(1):39–48, January 1974. ISSN 0016-8033, 1942-2156. doi: 10.1190/1.1440410. 131

[4] Bradley Efron and Gail Gong. A leisurely look at the bootstrap, the jackknife & cross-validation. *American Statistician*, 37:36–48, 1983. 91

[5] John C. Gallant and Trevor I. Dowling. A multiresolution index of valley bottom flatness for mapping depositional areas. *Water Resources Research*, 39(12):ESG4–1 – ESG4–13, Dec 2003. doi: 10.1029/2002WR001426. 115

[6] P Goovaerts. *Geostatistics for natural resources evaluation*. Applied Geostatistics. Oxford University Press, New York; Oxford, 1997. 54, 137

[7] T Hastie, R Tibshirani, and J H Friedman. *The elements of statistical learning data mining, inference, and prediction*. Springer series in statistics. Springer, New York, 2nd ed edition, 2009. ISBN 9780387848587. 67, 80, 90, 91, 131

[8] Tomislav Hengl, Gerard B. M. Heuvelink, and David G. Rossiter. About regression-kriging: From equations to case studies. *Computers & Geosciences*, 33(10):1301–1315, 2007. doi: 10.1016/j.cageo.2007.05.001. 43

[9] M. F. Hutchinson. Interpolating mean rainfall using thin plate smoothing splines. *International Journal of Geographical Information Science*, 9(4):385 – 403, 1995. 131

[10] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning: with applications in R*. Number 103 in Springer texts in statistics. Springer, 2013. ISBN 9781461471370. 67, 80, 90

[11] Max Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008. 101

[12] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2013 edition edition, Sep 2013. ISBN 978-1-4614-6848-6. 107

[13] R. M. Lark and B. R. Cullis. Model based analysis using REML for inference from systematically sampled data on soil. *European Journal of Soil Science*, 55(4):799–813, 2004. 26, 27

[14] Scott M. Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in*

Neural Information Processing Systems 30 (Nips 2017), volume 30, La Jolla, 2017. Neural Information Processing Systems (nips). 127

[15] G. S. McMaster and W. W. Wilhelm. Growing degree-days: one equation, two interpretations. *Agricultural and Forest Meteorology*, 87(4): 291–300, 1997. doi: 10.1016/S0168-1923(97)00027-0. 2

[16] H. Mitasova and J. Hofierka. Interpolation by regularized spline with tension: II. Application to terrain modeling and surface geometry analysis. *Mathematical Geology*, 25(6):657–669, 1993. doi: 10.1007/BF00893172. 131

[17] H. Mitasova and L. Mitas. Interpolation by regularized spline with tension: I. Theory and implementation. *Mathematical Geology*, 25(6): 641–655, 1993. doi: 10.1007/BF00893171. 131

[18] Christoph Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. Leanpub. URL https://leanpub.com/interpretable-machine-learning. 124

[19] J C Pinheiro and D M Bates. *Mixed-effects models in S and S-PLUS*. Springer, 2000. ISBN 0387989579. 27, 28

[20] J. R Quinlan. *C4.5: programs for machine learning*. The Morgan Kaufmann series in machine learning. Morgan Kaufmann Publishers, 1993. ISBN 1-55860-238-0. 107

[21] Shawn J. Riley, Stephen D. DeGloria, and Robert Eliot. A terrain ruggedness that quantifies topographic heterogeneity. *Intermountain Journal of Sciences*, 5(1–4):23–27, 1999. 116

[22] Cosma Shalizi. The bootstrap. *American Scientist*, 98(3):186–190, 2010. doi: DOI:10.1511/2010.84.186. URL http://www.americanscientist.org/issues/pub/2010/3/the-bootstrap/3. 91

[23] W. R. Tobler. A computer movie simulating urban growth in the Detroit region. *Economic Geography*, 46:234–240, 1970. ISSN 0013-0095. doi: 10.2307/143141. 136

[24] R. Webster and M. A. Oliver. *Geostatistics for environmental scientists*. John Wiley & Sons Ltd., 2nd edition, 2008. 54, 137

[25] Hadley Wickham. ggplot2. http://ggplot2.org/. URL http://ggplot2.org/. 6

[26] Hadley Wickham. *ggplot2: Elegant graphics for data analysis*. Use R! Springer, August 2009. ISBN 0387981403. 6

[27] Leland Wilkinson. *The grammar of graphics*. Statistics and computing. Springer, New York, 2nd ed edition, 2005. ISBN 9780387286952. 6

[28] S. N. Wood. Thin plate regression splines. *Journal of the Royal Statistical Society Series B-Statistical Methodology*, 65:95–114, 2003. doi: 10.1111/1467-9868.00374. 131

[29] Marvin N. Wright and Andreas Ziegler. ranger: a fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, Mar 2017. doi: 10.18637/jss.v077.i01. 91

[30] Yihui Xie. `knitr`: Elegant, flexible and fast dynamic report generation with R, 2011. URL http://yihui.name/knitr/. Accessed 04-Mar-2016. 2

A * Colour ramps with ggplot2

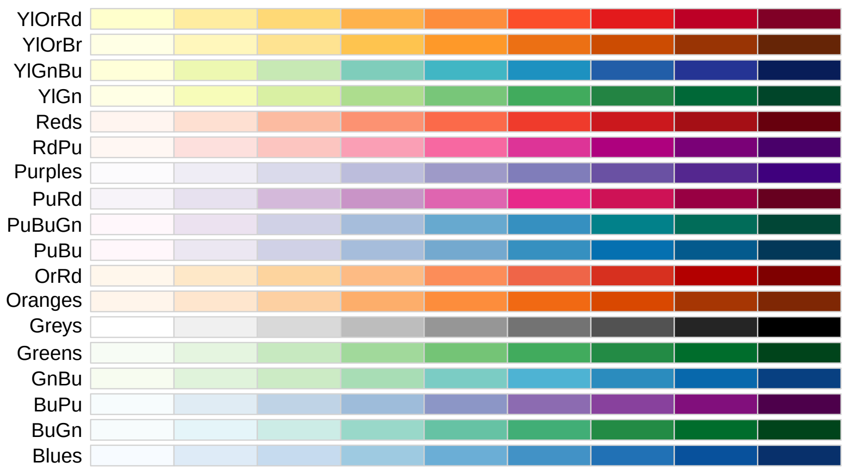
The `colour` argument to the `aes` "aesthetics" function has a default colour ramp.

In `ggplot2` terminology this is called a a **scale** It maps numbers (such as GDD50) to a scale in colour space, in the same way there is a mapping from numbers to a position on an axis of a scatterplot. It is a **function** from a **domain** (data values, which can be classes or continuous) to a **range** (the colour for that value). This is a quite tricky – the user's visual perception must match the change in colour. One package that has dealt with this is `RColorBrewer`[29], which palettes (colour choices) can be accessed with the `scale_colour_brewer` function for categorical variables and `scale_colour_distiller` for continuous variables. This package provides ready-made sequential, diverging, and qualitative palettes.

Task 159 : Load the `RColorBrewer` package and display the ready-made palettes for continuous **sequences**. •

```
library(RColorBrewer)
display.brewer.all(type="seq")
```

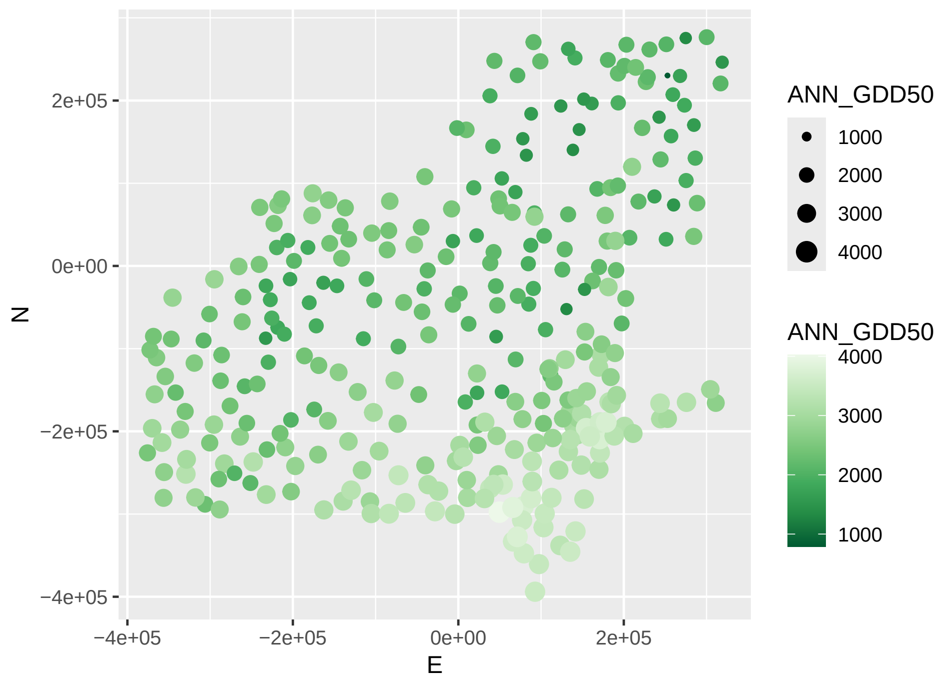
[29] http://colorbrewer2.org/



These should be used when the variable to display is a continuous value from some minimum to some maximum, where increasing values indicate increasing intensity of the variable. For example, the annual GDD50, from low to high.

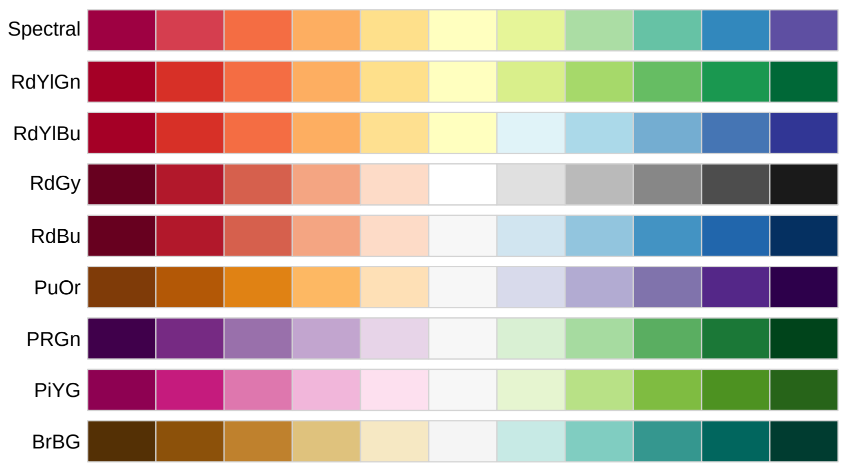
Task 160 : Select one of the palettes and re-display the GDD50 map with it, •

```
ggplot(data=ne.df) +
    aes(x=E, y=N) +
    geom_point(aes(size=ANN_GDD50, colour=ANN_GDD50),
               shape=20) +
    scale_colour_distiller(space="Lab",
                           palette="Greens") +
        xlab("E") +  ylab("N") + coord_fixed()
```



Task 161 : Display the ready-made palettes for **diverging** palettes. •

```
display.brewer.all(type="div")
```



These should be used when the central value is a natural zero and we want to emphasize divergences from it in two directions, e.g., for residuals.

Index of Commands