SpatCourse_SSN2

Stefano Larsen

4/8/2021

Load libraries

```
library(SSN)
```

Tutorial with the ready-available spatial data in SSN R package

Note: part of these examples come from the original SSN vignette!

In this script we will use the Middle Fork river data to fit models for mean summer water T. Then we will predict mean temperature across a range of un-sampled locations across the river network.

Import the ssn file from the SSN R package system file

```
mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn",
   package = "SSN"), predpts = "pred1km", o.write = TRUE)
```

You can also import specific set of locations for which we want to make predictions.

This is done using the *importPredpts* comand.

```
mf04p <- importPredpts(mf04p, "Knapp", "ssn")
mf04p <- importPredpts(mf04p, "CapeHorn", "ssn")
```

We explore the mf04p SSN object.

It shows four groups of variables: the observed, and three sets of prediction locations.

```
mf04p
```

```
## Object of class Spatial Stream Network
##
## Object includes observations on 35 variables across 45 sites within the bounding box
##         min      max
## x -1531385 -1498448
```

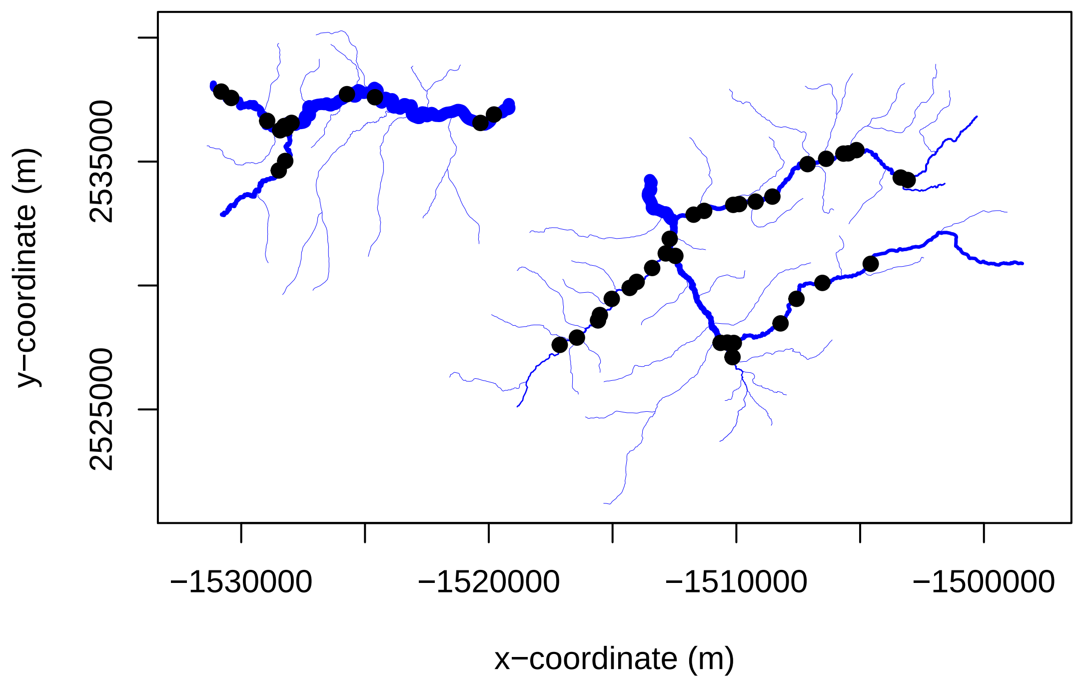
```
## y  2521181  2540274
##
## Object also includes 3 sets of prediction points with a total of 2102 locations
##
## Variables recorded are (found using names(object)):
## $Obs
##  [1] "STREAMNAME" "HUC3"       "HUC4"       "COMID"      "CUMDRAINAG"
##  [6] "AREAWTMAP"  "MAXELEVSMO" "SLOPE"      "NCEASID_"   "ELEV_DEM"
## [11] "Deployment" "SampleYear" "NumberOfDa" "OriginalID" "Source"
## [16] "Summer_mn"  "MaxOver20"  "C16"        "C20"        "C24"
## [21] "FlowCMS"    "AirMEANc"   "AirMWMTc"   "NEAR_FID"   "NEAR_DIST"
## [26] "NEAR_X"     "NEAR_Y"     "NEAR_ANGLE" "rid"        "ratio"
## [31] "afvArea"    "upDist"     "locID"      "netID"      "pid"
##
## $pred1km
##  [1] "COMID"      "GNIS_NAME"  "CUMDRAINAG" "HUC3"       "HUC4"
##  [6] "AREAWTMAP"  "MAXELEVSMO" "SLOPE"      "COMID_"     "ELEV_DEM"
## [11] "FlowCMS"    "AirMEANc"   "AirMWMTc"   "SampleYear" "NEAR_FID"
## [16] "NEAR_DIST"  "NEAR_X"     "NEAR_Y"     "NEAR_ANGLE" "rid"
## [21] "ratio"      "afvArea"    "upDist"     "locID"      "netID"
## [26] "pid"
##
## $Knapp
##  [1] "COMID"      "GNIS_NAME"  "CUMDRAINAG" "HUC3"       "HUC4"
##  [6] "AREAWTMAP"  "MAXELEVSMO" "SLOPE"      "COMID_"     "ELEV_DEM"
## [11] "FlowCMS"    "AirMEANc"   "AirMWMTc"   "SampleYear" "NEAR_FID"
## [16] "NEAR_DIST"  "NEAR_X"     "NEAR_Y"     "NEAR_ANGLE" "rid"
## [21] "ratio"      "afvArea"    "upDist"     "locID"      "netID"
## [26] "pid"
##
## $CapeHorn
##  [1] "COMID"      "GNIS_NAME"  "CUMDRAINAG" "HUC3"       "HUC4"
##  [6] "AREAWTMAP"  "MAXELEVSMO" "SLOPE"      "COMID_"     "ELEV_DEM"
## [11] "FlowCMS"    "AirMEANc"   "AirMWMTc"   "SampleYear" "NEAR_FID"
## [16] "NEAR_DIST"  "NEAR_X"     "NEAR_Y"     "NEAR_ANGLE" "rid"
## [21] "ratio"      "afvArea"    "upDist"     "locID"      "netID"
## [26] "pid"
##
## Generic functions that work with this object include names, plot, print, summary, hist, boxplot and 
```

Creating distance matrices is necessary

```
createDistMat(mf04p, predpts = "Knapp", o.write = TRUE,
    amongpreds = TRUE)
createDistMat(mf04p, predpts = "CapeHorn", o.write = TRUE,
    amongpreds = TRUE)
```

Let's plot our data

```
plot(mf04p, lwdLineCol = "afvArea", lwdLineEx = 6, lineCol = "blue",
   pch = 19, xlab = "x-coordinate (m)", ylab = "y-coordinate (m)",
   asp = 1)
```



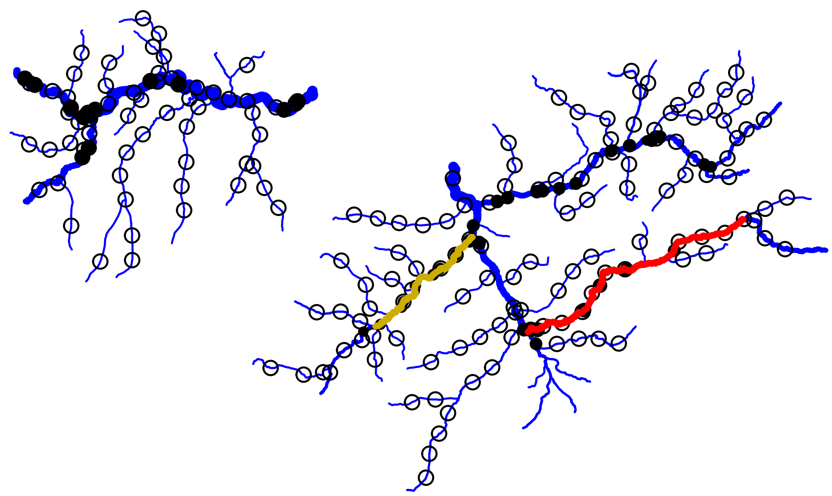
This chunk must be run all at once inside the .Rmd script.

Alternatively, run each line directly in the console.

Here we first plot the rivers as spatial lines object. Then we plot the locations with observed data. Then we plot three sets of prediction points. The first is a diffuse 1-km apart prediction across the networks. The second (Knapp) and third (CapeHorn) are dense sets of prediction points used for *block-kriging'.

```
#plot as spatial lines object
plot(as.SpatialLines(mf04p), col = "blue",
    lwd = 1+ log(1+ mf04p@data$afvArea)*6)

# add the observed locations with size proportional
# to mean summer temperature
plot(as.SpatialPoints(mf04p), pch = 19,
cex = as.SpatialPointsDataFrame(mf04p)$Summer_mn/15 , add = TRUE)
```

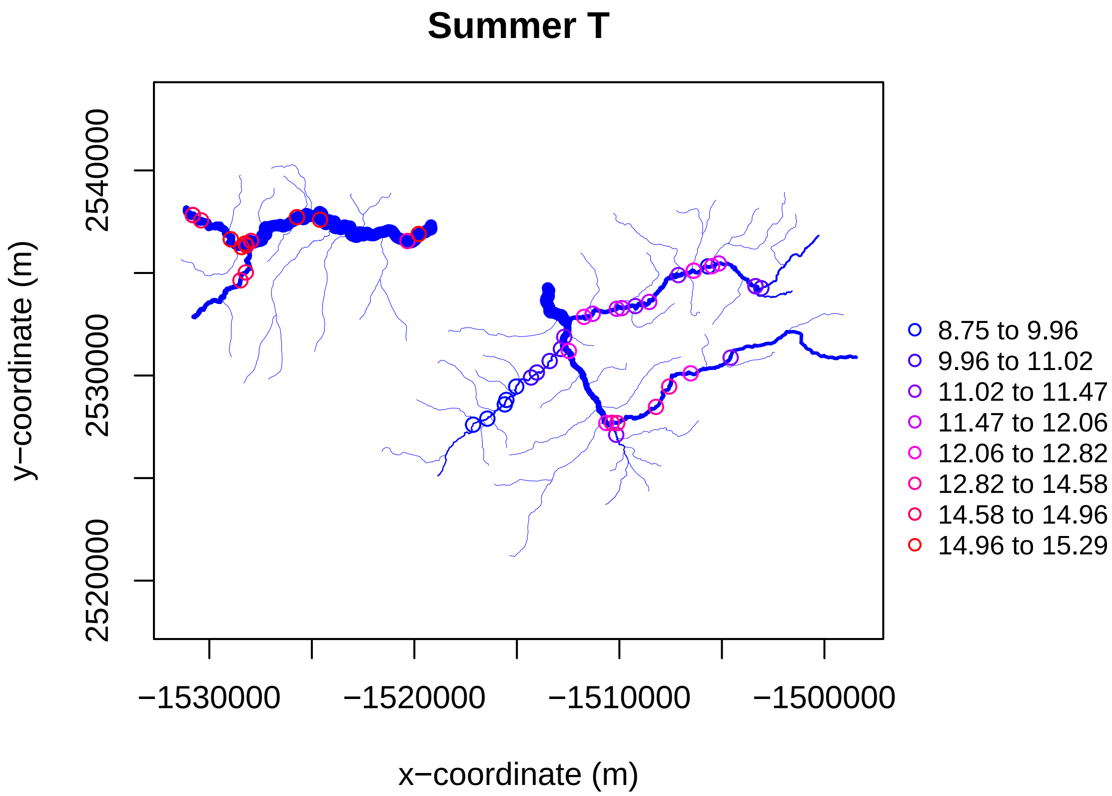
```
# add the prediction locations on the 1 km spacing
plot(as.SpatialPoints(mf04p, data = "pred1km"),
    cex = 1, add = TRUE, pch = 1)

# add the dense set of points for block prediction on Knapp segment
plot(as.SpatialPoints(mf04p, data = "Knapp"), pch = 19, cex = 0.3,
  col = "red", add = TRUE)
# add the dense set of points for block prediction on CapeHorn segment
plot(as.SpatialPoints(mf04p, data = "CapeHorn"), pch = 19, cex = 0.3,
  col = "Gold3", add = TRUE)
```



We can have a look at the values of mean summer T.

Use *nclasses* to create break points for plotting.

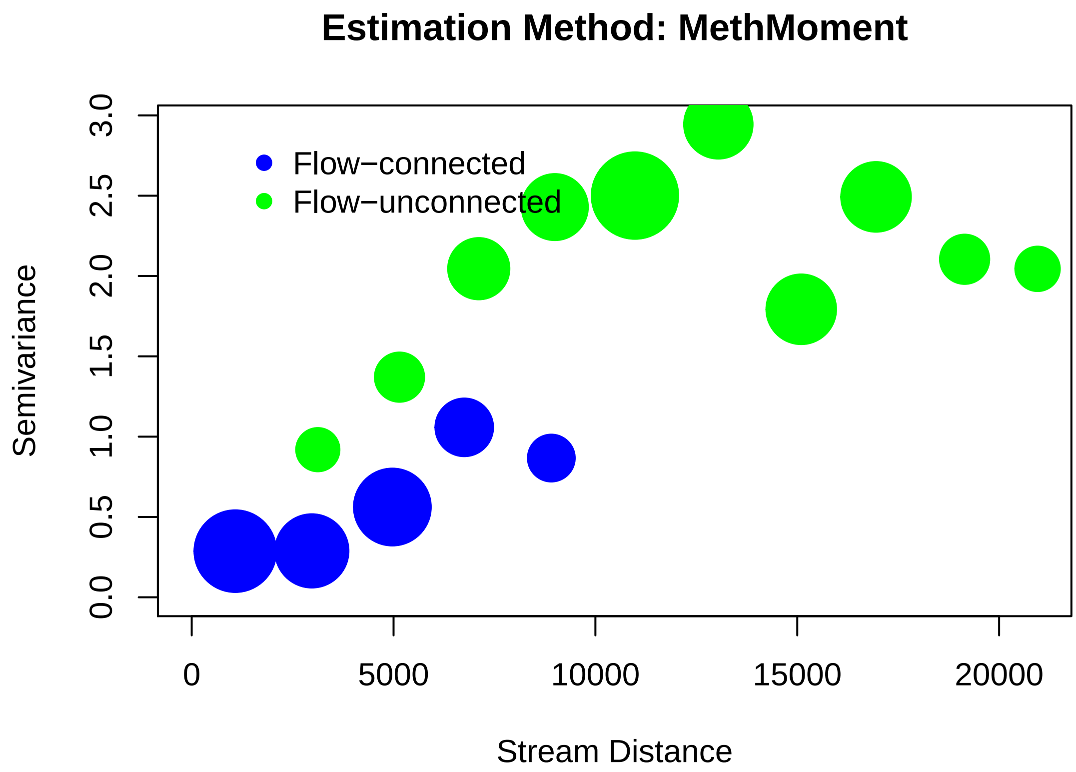
```
plot(mf04p, 'Summer_mn',, lwdLineCol = "afvArea", lwdLineEx = 6, lineCol = "blue",
   pch = 1, xlab = "x-coordinate (m)", ylab = "y-coordinate (m)",
   asp = 1, nclasses=8, main='Summer T')
```



As always is good to have a look at the Torgegram.

We see that autocorrelation in water Tis high among flow-connected locations (lower variances), while flow-unconnected locations have larger variances.

```
tor.summer_mn<- Torgegram(mf04p, "Summer_mn", nlag = 20, maxlag = 40000)
plot(tor.summer_mn)
```



We can now fit some models

We start by fitting a non-spatial models for summer T using elevation and slope as covariates

The summary shows that both elevation and slope are important covariates, explaining summer T.

```
mf04.glmssn0 <- glmssn(Summer_mn ~ ELEV_DEM + SLOPE, mf04p,
   CorModels = NULL, use.nugget = TRUE, EstMeth = "ML")
summary(mf04.glmssn0)
```

```
##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM + SLOPE, ssn.object = mf04p,
##     CorModels = NULL, use.nugget = TRUE, EstMeth = "ML")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.3835 -1.1704  0.6205  0.9088  2.1930
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  62.839372  10.292925   6.105  < 2e-16 ***
## ELEV_DEM     -0.024955   0.005197  -4.802    2e-05 ***
## SLOPE       -88.019985  30.906572  -2.848  0.00678 **
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##  Covariance.Model Parameter Estimate
##            Nugget   parsill     1.69
##
## Residual standard error: 1.298518
## Generalized R-squared: 0.5625148
```

Then we can fit a spatial models including all autocovariance functions.

The summary shows that, in the spatial model, slope is not significant anymore. This happens often that spatial models have less frequent significant predictors.

```
mf04.glmssn1 <- glmssn(Summer_mn ~ ELEV_DEM + SLOPE, mf04p,
   CorModels = c("Exponential.tailup", "Exponential.taildown",
      "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")
summary(mf04.glmssn1)
```

```
##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM + SLOPE, ssn.object = mf04p,
##     CorModels = c("Exponential.tailup", "Exponential.taildown",
##         "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9604 -1.6071 -0.1483  0.5446  1.5256
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  63.109796  12.099159   5.216    1e-05 ***
## ELEV_DEM     -0.025000   0.006042  -4.137  0.00016 ***
## SLOPE       -30.302536  15.982053  -1.896  0.06485 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##      Covariance.Model Parameter    Estimate
##    Exponential.tailup   parsill      1.2410
##    Exponential.tailup     range 117751.1625
##  Exponential.taildown   parsill      0.0799
##  Exponential.taildown     range  54012.2604
##    Exponential.Euclid   parsill      0.2356
##    Exponential.Euclid     range  30700.7140
##              Nugget   parsill      0.0238
##
## Residual standard error: 1.257096
## Generalized R-squared: 0.4158553
```

We can fit another model excluding the effect of slope.

```
mf04.glmssn2 <- glmssn(Summer_mn ~ ELEV_DEM , mf04p,
   CorModels = c("Exponential.tailup", "Exponential.taildown",
      "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")

summary(mf04.glmssn2)
```

```
##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM, ssn.object = mf04p, CorModels = c("Exponential.tailup",
##     "Exponential.taildown", "Exponential.Euclid"), addfunccol = "afvArea",
##     EstMeth = "ML")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4890 -1.9089 -0.4207  0.1934  1.3568
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 70.519387  10.720076   6.578   <2e-16 ***
## ELEV_DEM    -0.028653   0.005356  -5.350   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##      Covariance.Model Parameter     Estimate
##    Exponential.tailup   parsill      1.84958
##    Exponential.tailup     range 117677.23917
##  Exponential.taildown   parsill      0.05093
##  Exponential.taildown     range  49830.62383
##    Exponential.Euclid   parsill      0.00961
##    Exponential.Euclid     range  31458.88862
##                Nugget   parsill      0.00817
##
## Residual standard error: 1.385025
## Generalized R-squared: 0.408599
```

```
varcomp(mf04.glmssn2)
```

```
##                VarComp  Proportion
## 1    Covariates (R-sq) 0.408599020
## 2   Exponential.tailup 0.570216085
## 3 Exponential.taildown 0.015702700
## 4   Exponential.Euclid 0.002962057
## 5               Nugget 0.002520137
```

We fit a third model, this timee exlcuding the Euclidean autocovariance function.

```r
mf04.glmssn3 <- glmssn(Summer_mn ~ ELEV_DEM , mf04p,
   CorModels = c("Exponential.tailup", "Exponential.taildown"), addfunccol = "afvArea", EstMeth = "ML")
summary(mf04.glmssn3)
```

```
##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM, ssn.object = mf04p, CorModels = c("Exponential.tailup",
##     "Exponential.taildown"), addfunccol = "afvArea", EstMeth = "ML")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4599 -1.8444 -0.3773  0.2742  1.3736
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 69.104034  11.313490   6.108   <2e-16 ***
## ELEV_DEM    -0.027972   0.005644  -4.956    1e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##      Covariance.Model Parameter     Estimate
##    Exponential.tailup   parsill      1.79102
##    Exponential.tailup     range 117791.93394
##  Exponential.taildown   parsill      0.18276
##  Exponential.taildown     range  40859.42223
##                Nugget   parsill      0.00217
##
## Residual standard error: 1.405685
## Generalized R-squared: 0.372422
```

We compare the different models in terms of RMSPE, AIC and COV.90

It looks like *mf04.glmssn1* is the best model among these.

```r
InfoCritCompare(list(mf04.glmssn0,mf04.glmssn1,mf04.glmssn2, mf04.glmssn3 ))[,c(3,5,8,12)]
```

```
##                                                          Variance_Components
## 1                                                                     Nugget
## 2 Exponential.tailup + Exponential.taildown + Exponential.Euclid + Nugget
## 3 Exponential.tailup + Exponential.taildown + Exponential.Euclid + Nugget
## 4                     Exponential.tailup + Exponential.taildown + Nugget
##         AIC     RMSPE    cov.90
## 1 159.21471 1.4504309 0.9111111
## 2  77.66517 0.4491926 0.9333333
## 3  74.39852 0.5417670 0.8888889
## 4  71.25492 0.5500877 0.8888889
```

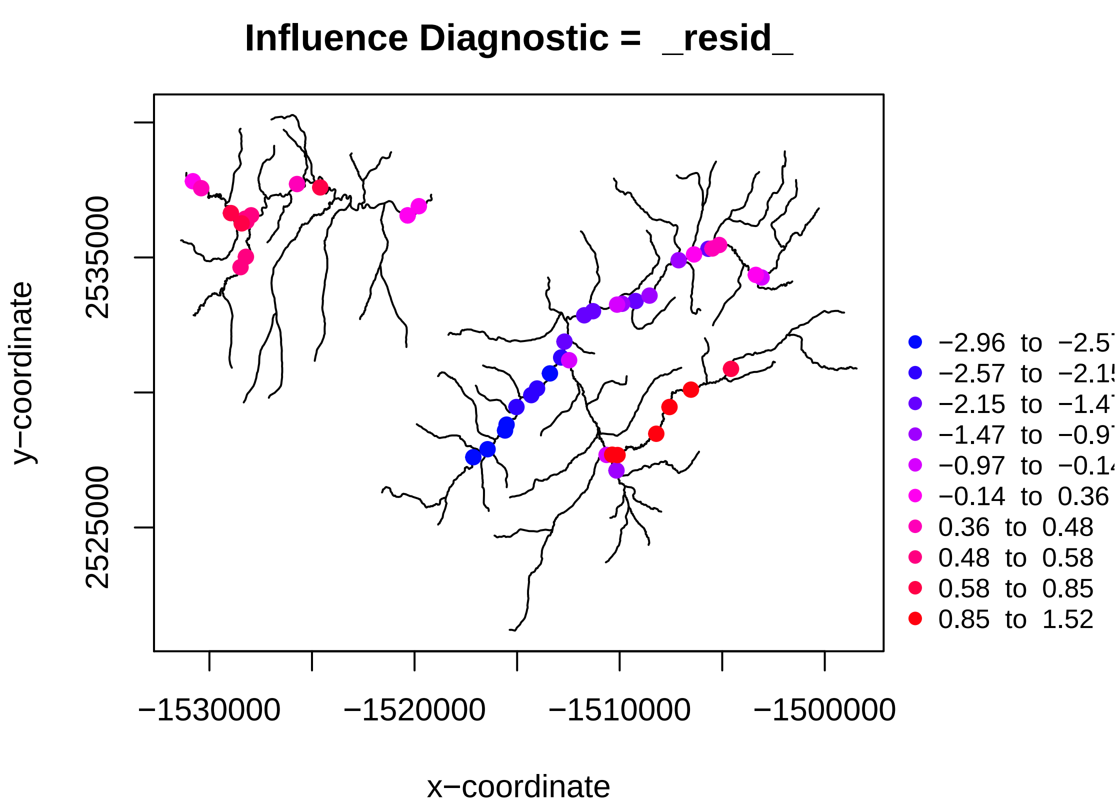
Let's explore the residuals

The result of the residuals function is an `influenceSSN' object, which is an exact copy of the`glmssn' object, except that residual diagnostics are appended as new columns to the data frame point.data containing the observed data. The default plotting method for an 'influenceSSN' object is a map with color-coded raw residuals.

```
# get the residuals from the model
mf04.resid1 <- residuals(mf04.glmssn1)
# Explore the new variables appended to the data
names(getSSNdata.frame(mf04.resid1))
```

```
##  [1] "pid"              "STREAMNAME"       "HUC3"             "HUC4"
##  [5] "COMID"            "CUMDRAINAG"       "AREAWTMAP"        "MAXELEVSMO"
##  [9] "SLOPE"            "NCEASID_"         "ELEV_DEM"         "Deployment"
## [13] "SampleYear"       "NumberOfDa"       "OriginalID"       "Source"
## [17] "Summer_mn"        "MaxOver20"        "C16"              "C20"
## [21] "C24"              "FlowCMS"          "AirMEANc"         "AirMWMTc"
## [25] "NEAR_FID"         "NEAR_DIST"        "NEAR_X"           "NEAR_Y"
## [29] "NEAR_ANGLE"       "rid"              "ratio"            "afvArea"
## [33] "upDist"           "locID"            "netID"            "obsval"
## [37] "_fit_"            "_resid_"          "_resid.stand_"    "_resid.student_"
## [41] "_leverage_"       "_CooksD_"
```

```
#simple plot of residuals
plot(mf04.resid1)
```

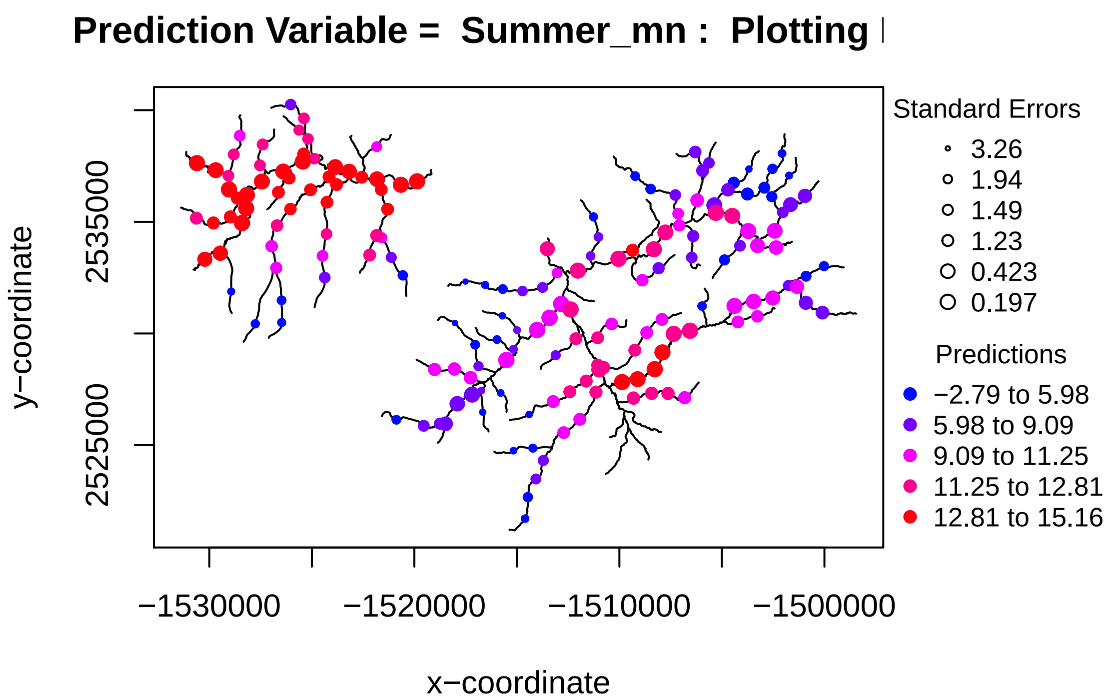


The residual plot shows that some locations have rather large residual values (< -3). We could eventually remove these outliers. For now we carry on using all the data points.

We can now make predictions of summer T, using the model we have fitted.

First we predict over the range of locations across the networks. These are the points data *pred1km*.

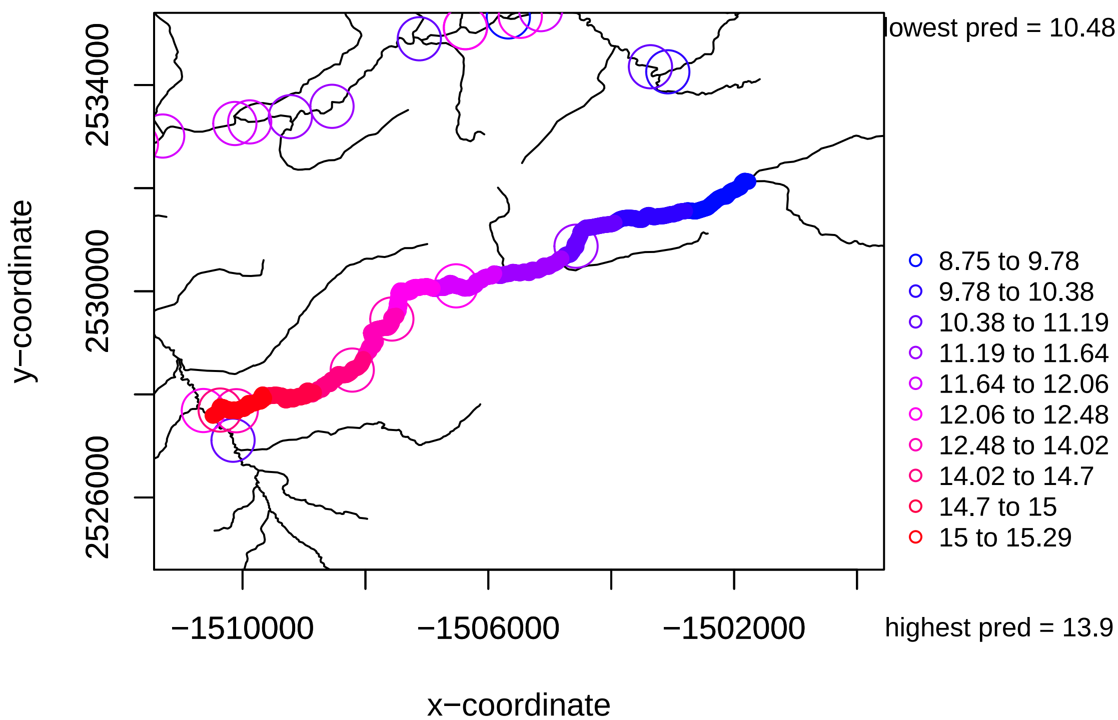
```
mf04.pred1km <- predict(mf04.glmssn1, "pred1km")
plot(mf04.pred1km, SEcex.max = 1, SEcex.min = .3, nclasses=5)
```



Then we can also make block predictions, over a range of adjacent points along a river reach. here are the points along the *Knapp* river.

```
# Plot zooming to the 'Knapp' river where predictions are needed
plot(mf04p, "Summer_mn", pch = 1, cex = 3,
   xlab = "x-coordinate", ylab = "y-coordinate",
   xlim = c(-1511000,-1500000), ylim = c(2525000,2535000))

#Run the predictions using the 'Knapp' prediction locations
mf04.glmssn1.Knapp <- predict(mf04.glmssn1, "Knapp")

#The plot it
```

```
plot(mf04.glmssn1.Knapp, "Summer_mn", add = TRUE,
   xlim = c(-1511000,-1500000), ylim = c(2525000,2535000))
```



Excercise

We have modelled water temperature using gaussian distribution (the default).

However, the glmssn allows different distribution of the response variable, including binomial and poisson (counts).

The mf04p dataset also contains a variable (*C16*) reporting the number of days the water T was above 16C. This is therefore a *count* variable that can be modelled using a poisson distribution.

Here is the variable *C16*

```
mf04p@obspoints@SSNPoints[[1]]@point.data$C16
```

```
##  [1] 24 22 17 11  7  2  2  2  1 36 30 40 37 32 17 25 32 31 34 28 15 33 21 40 33
## [26] 31 11 21  0  0 39 41 40 40 39 40 40 39 38 40 40 38 38 36  5
```

You can try yourself:

- Fit a poisson model for the variable *C16* using again elevation and slope as covariates.
- Exclude non-significant predictors.
- Predict the *C16* over the prediction locations (e.g. Knapp stream) using your model.

TIP

In SSN we can include the line: family = "poisson" in the *glmssn* call, as in standard *glm* models.

Then you can use the examples already provided to make prediction on the Knapp stream.