

# SpatCourse\_SSN1

Stefano Larsen

3/22/2021

## Scripts and examples for visualising and analysing Spatial Stream Network models (SSN)

These scripts provide introductory examples on how to deal with .ssn objects in R. We will work with a ready-available .ssn object in .RData format. You can download this from: <https://github.com/stefanolarsen/SpatCourse> The dataset include macroinvertebrates community data across the Adige River network (NE Italy). Environmental descriptors, diversity and feeding-trait metrics are included. For details see: **Larsen et al 2019 - Testing the river continuum concept with geo-statistical stream network models. Ecological Complexity.** <https://doi.org/10.1016/j.ecocom.2019.100773>

### Load the key libraries.

```
library(SSN)
library(tidyverse)
#library(gtools)
```

### Load the RData with the SSN object (stream invertebrates from Adige River network)

You must first download this .RData object from the <https://github.com/stefanolarsen/SpatCourse>. Then modify the code below to include your path (where you downloaded the file).

```
load("~/Documents/SpatCoursePT/SpatCourse/spat.course.ssn1.RData")
#gitUrl='https://github.com/stefanolarsen/SpatCourse/blob/main/bent.ssn.logit.RData'
```

### Create the distance matrices.

These are non-orthodox distance matrices. See help for details. Distance matrix is important for modelling later!

```
createDistMat(spat.course.ssn)
```

```
## Distance matrices already existed while o.write was set to FALSE. Not overwriting existing matrices
```

Now that you have the `.ssn` object imported, you can explore it.

Spatial objects can be explored with *summary*.

```
summary(spat.course.ssn)
```

```
## Object of class Spatial Stream Network
##
## Object includes observations on 49 variables across 195 sites within the bounding box
##      min      max
## x  601178.4  753197.5
## y  5042879.0 5219401.0
##
## Variables recorded are (found using names(object)):
## $Obs
##   [1] "code"      "X"          "Y"          "richness"   "shannon"
##   [6] "simpson"   "density"    "Star_icmi"  "T_wat"      "pH"
##  [11] "Ox_sat"    "Ox_dis"     "N_tot"      "NO3"        "NH4"
##  [16] "P"         "T_air"      "upDist"     "h2oArea"    "h2oAgric"
##  [21] "h2oWood"   "Z"          "buf_agric_" "buf_wood_a" "mean_slope"
##  [26] "afvArea"   "NEAR_FID"   "NEAR_DIST"  "NEAR_X"     "NEAR_Y"
##  [31] "NEAR_ANGLE" "rid"        "ratio"      "locID"      "netID"
##  [36] "pid"       "PCA1.long"  "LIMeco"     "buf.agr.p"  "buf.wood.p"
##  [41] "h2oAgric_p" "h2oWood_p" "feed_gra"   "feed_xyl"   "feed_shr"
##  [46] "feed_gat"  "feed_pff"   "feed_pre"   "feed_oth"
##
## Generic functions that work with this object include names, plot, print, summary, hist, boxplot and c
```

## Explore the variables in the data only.

Here there are diversity metrics (richness, shannon). Other descriptors e.g. `T_wat` (water T), `LIMeco` (water quality index). Other variables (`netID`, `ratio`) come from the process leading to the SSN file, from GIS. `##` An important variable is *afvArea* (the additive function based on catchment area), use for weighting the correlation function at river junctions.

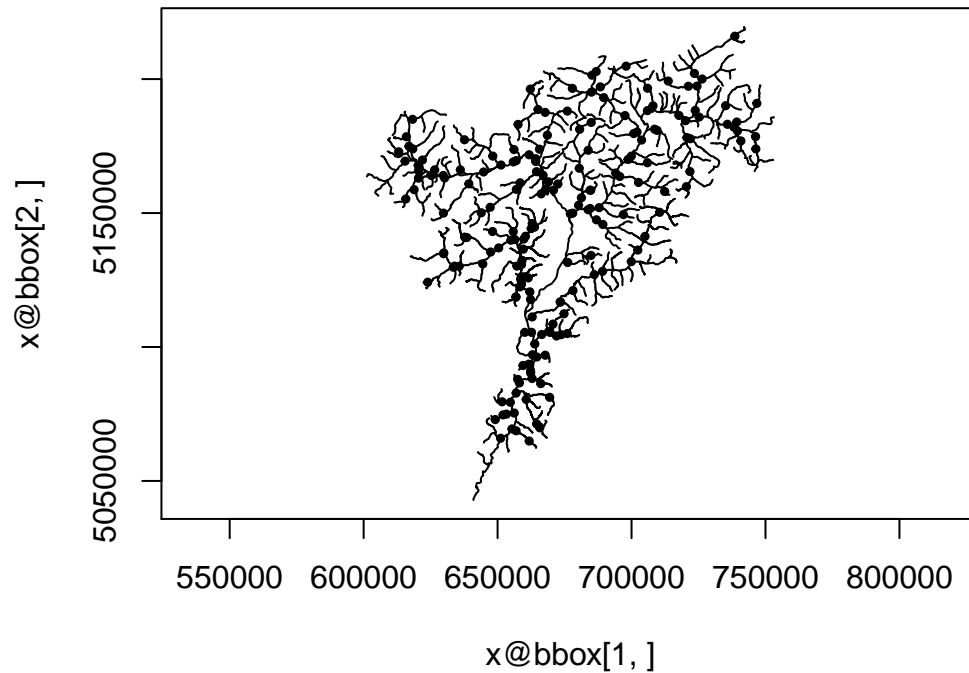
```
names(spat.course.ssn)
```

```
## $Obs
##   [1] "code"      "X"          "Y"          "richness"   "shannon"
##   [6] "simpson"   "density"    "Star_icmi"  "T_wat"      "pH"
##  [11] "Ox_sat"    "Ox_dis"     "N_tot"      "NO3"        "NH4"
##  [16] "P"         "T_air"      "upDist"     "h2oArea"    "h2oAgric"
##  [21] "h2oWood"   "Z"          "buf_agric_" "buf_wood_a" "mean_slope"
##  [26] "afvArea"   "NEAR_FID"   "NEAR_DIST"  "NEAR_X"     "NEAR_Y"
##  [31] "NEAR_ANGLE" "rid"        "ratio"      "locID"      "netID"
##  [36] "pid"       "PCA1.long"  "LIMeco"     "buf.agr.p"  "buf.wood.p"
##  [41] "h2oAgric_p" "h2oWood_p" "feed_gra"   "feed_xyl"   "feed_shr"
##  [46] "feed_gat"  "feed_pff"   "feed_pre"   "feed_oth"
```

## Explore plotting functions

Simple *plot* function can also work. You see the network and the sampling points.

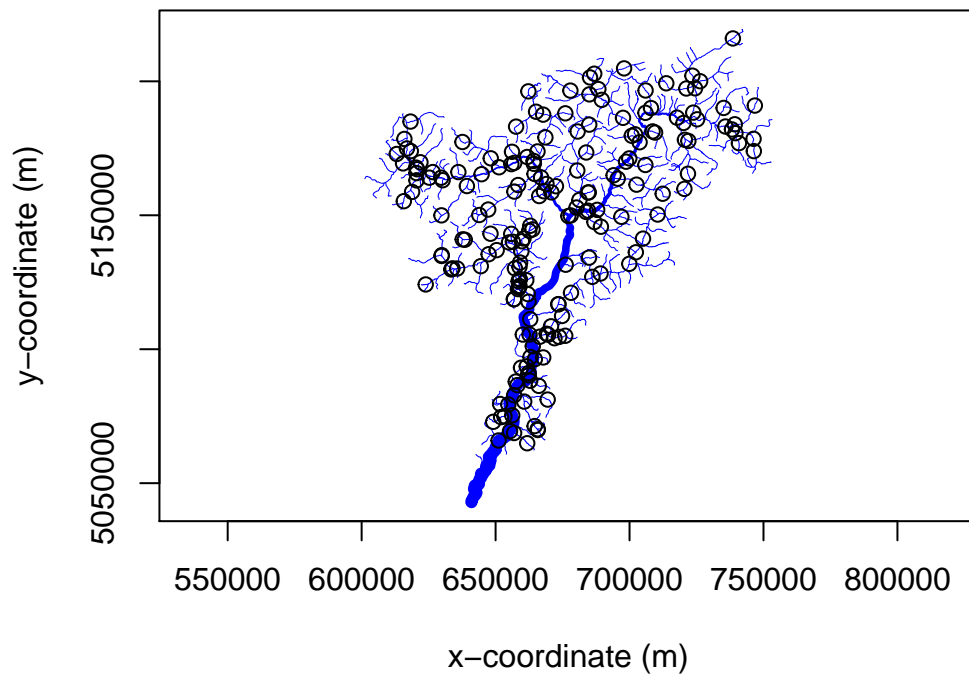
```
plot(spat.course.ssn, asp=1, cex=0.5)
```



Nicer option for plotting; can weight the line width by catchment area.

The option here is: 'lwdLineCol=afvArea'.

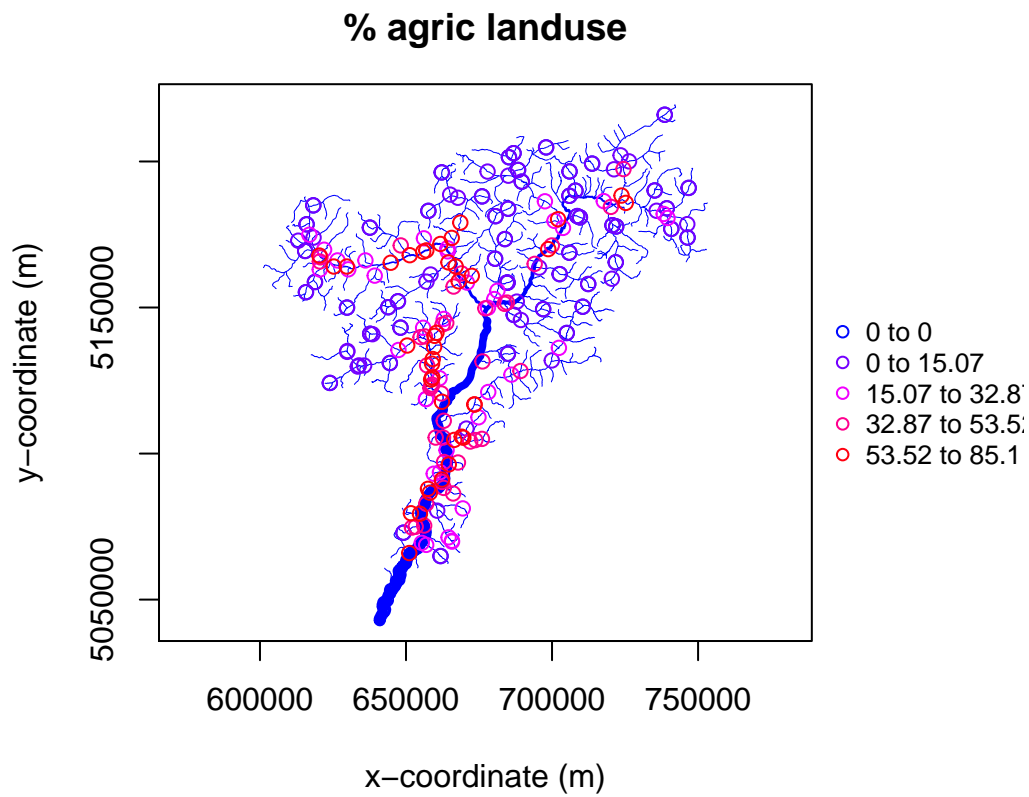
```
plot(spat.course.ssn, lwdLineCol = "afvArea", lwdLineEx = 6, lineCol = "blue",
     pch = 1, xlab = "x-coordinate (m)", ylab = "y-coordinate (m)",
     asp = 1)
```



## Exploring patterns in specific variables

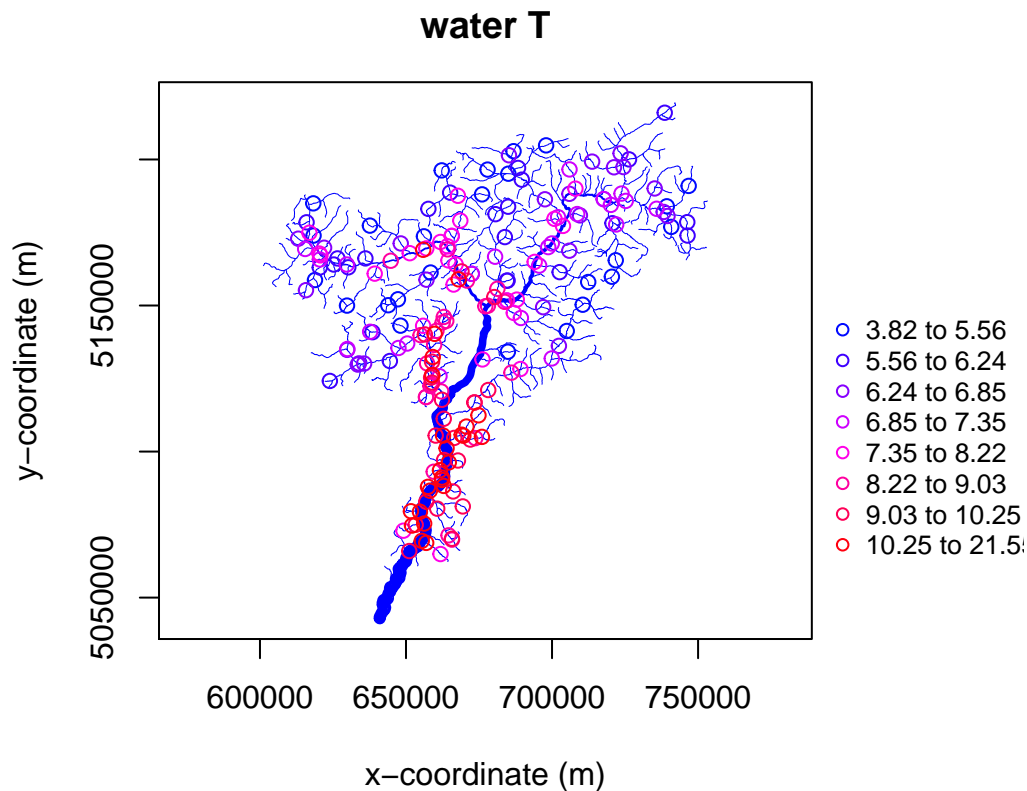
You can also plot specific variable values, with color classes (defined in 'nclasses'). Here e.g. the % of agricultural land-use around each site (buf.agr.p).

```
plot(spat.course.ssn, 'buf.agr.p',, lwdLineCol = "afvArea", lwdLineEx = 6, lineCol = "blue",
     pch = 1, xlab = "x-coordinate (m)", ylab = "y-coordinate (m)",
     asp = 1, nclasses=5, main='% agric landuse')
```



Or plot the water temperature ( $T_{\text{wat}}$ ).

```
plot(spat.course.ssn, 'T_wat',, lwdLineCol = "afvArea", lwdLineEx = 6, lineCol = "blue",
     pch = 1, xlab = "x-coordinate (m)", ylab = "y-coordinate (m)",
     asp = 1, nclasses=8, main='water T')
```



## Manipulate the data frames inside a SSN object

In this example you need to: e.g. Fill NAs in the water quality index: LIMeco Re-transform “logit”-transformed traits to proportions for some traits values.

### Extract dataframe from ssn object.

This command extract the data matrix from the ssn object. We can work on it, and then put it back into a ssn object. This is useful if we need to modify or add data to our dataframe.

```
datassn=getSSNdata.frame(spat.course.ssn)
```

### Work on the dataframe. Explore structure.

This is now like a standard dataframe in R.

```
str(datassn)
```

```
## 'data.frame':   195 obs. of  49 variables:
## $ code       : chr  "11104" "11105" "11106" "11107" ...
## $ X         : num  616913 620774 629715 644783 656953 ...
## $ Y         : num  5174929 5167246 5164037 5165336 5169574 ...
```

```

## $ richness : num 26.6 21 18.8 15.3 17.6 ...
## $ shannon : num 10.56 6.35 7.94 4.15 4.93 ...
## $ simpson : num 7.31 4.02 5.9 3.1 3.41 ...
## $ density : num 1399 1156 523 1528 1661 ...
## $ Star_icmi : num 1.09 0.993 0.888 0.703 0.869 ...
## $ T_wat : num 7.4 8.18 6.86 8.84 7.52 ...
## $ pH : num 7.85 8.2 7.89 7.96 7.86 ...
## $ Ox_sat : num 112 104 108 107 108 ...
## $ Ox_dis : num 11.6 11.2 12 11.6 12.3 ...
## $ N_tot : chr "NA" "NA" "NA" "NA" ...
## $ NO3 : num 0.93 1.3 NA NA NA ...
## $ NH4 : num 0.0418 0.105 0.0561 0.0564 0.0425 ...
## $ P : num 0.0264 0.0325 0.0359 0.0321 0.0273 ...
## $ T_air : chr "9.546428571" "16" "12.52727273" "13.52142857" ...
## $ upDist : num 216962 206978 197260 180683 167066 ...
## $ h2oArea : num 232 479 892 1294 1625 ...
## $ h2oAgric : num 1.24 8.15 31.96 66.28 95.92 ...
## $ h2oWood : num 44.3 87.8 201.1 317.4 394.8 ...
## $ Z : num 1302 888 864 602 510 ...
## $ buf_agric_ : num 661581 1430320 1997098 2187835 2640478 ...
## $ buf_wood_a : num 809991 1098029 195539 0 339981 ...
## $ mean_slope : num 7.313 0.742 0.1 1.138 0.682 ...
## $ afvArea : num 0.0277 0.0539 0.099 0.1339 0.1581 ...
## $ NEAR_FID : int 523 517 142 488 219 520 472 259 256 443 ...
## $ NEAR_DIST : num 6.96e-05 9.99e-05 7.37e-06 1.25e-05 4.98e-05 ...
## $ NEAR_X : num 616913 620699 629715 644784 656954 ...
## $ NEAR_Y : num 5174929 5167181 5164033 5165332 5169559 ...
## $ NEAR_ANGLE : num 169.2 -141.7 85.2 103.9 93.4 ...
## $ rid : int 522 516 141 487 218 519 471 258 255 442 ...
## $ ratio : num 0.5493 0.3346 0.0723 0.7451 0.1208 ...
## $ locID : Factor w/ 195 levels "1","100","101",...: 1 101 123 134 144 152 163 174 185 29 ...
## $ netID : Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
## $ pid : int 1 2 3 4 5 6 7 8 9 13 ...
## $ PCA1.long : num -1.034 -0.429 0.201 0.778 1.015 ...
## $ LIMeco : num 0.87 0.85 0.82 0.68 0.865 0.88 0.76 NA NA 0.91 ...
## $ buf.agr.p : num 21.1 45.5 63.6 69.6 84 ...
## $ buf.wood.p : num 25.78 34.95 6.22 0 10.82 ...
## $ h2oAgric.p : num 0.535 1.704 3.584 5.124 5.903 ...
## $ h2oWood.p : num 19.1 18.3 22.6 24.5 24.3 ...
## $ feed_gra : num 0.371 0.517 0.4 0.337 0.346 ...
## $ feed_xyl : num -5.13 -5.65 -5.48 -6.1 -6.05 ...
## $ feed_shr : num 0.0886 0.0652 0.1189 0.0398 0.1357 ...
## $ feed_gat : num 0.234 0.268 0.28 0.271 0.313 ...
## $ feed_pff : num 0.1445 0.0451 0.0895 0.2723 0.0983 ...
## $ feed_pre : num 0.1555 0.1013 0.1074 0.0778 0.1044 ...
## $ feed_oth : num -3.66 -3.66 -3.66 -3.66 -3.66 ...

```

*#some feeding traits proportions are expressed as logit (e.g feed\_xyl)*  
*#we can convert it back to proportions if we want.*

Look for missing values

```
sum(is.na(datassn$LIMeco))
```

```
## [1] 27
```

```
# there are missing values in LIMeco
```

We can fill missing values in LIMeco with overall mean.

```
datassn$LIMeco[is.na(datassn$LIMeco)]<- mean(datassn$LIMeco, na.rm=T) #
```

Just a function to transform logit back to proportion of some feeding traits.

*delogit function:* convert logit transformed proportions back to proportions

```
delogit=function(x){  
  x1=exp(x)  
  prop=x1/(1+x1)  
  return(prop)  
}
```

Convert these traits back to proportions.

```
datassn$feed_oth=delogit(datassn$feed_oth)  
datassn$feed_xyl=delogit(datassn$feed_xyl)
```

**Then we create a second SSN object with the new modified dataframe.**

We can call it the same way for convenience (overwrite the existing one).

```
spat.course.ssn=putSSNdata.frame(datassn, spat.course.ssn)  
#save(spat.course.ssn, file='SpatCourse.ssn.RData')
```

To specifically explore the data within a SSN object:access the slots using `*@*`

```
head(spat.course.ssn@obspoints@SSNPoints[[1]]@point.data)
```

```
##      code      X      Y richness  shannon  simpson  density Star_icmi  
## 1 11104 616912.8 5174929 26.55556 10.558511 7.311361 1398.7778 1.0895556  
## 2 11105 620773.8 5167246 21.00000 6.349482 4.022947 1156.1667 0.9925090  
## 3 11106 629715.1 5164037 18.77778 7.944358 5.899238 523.2222 0.8879836  
## 4 11107 644783.1 5165336 15.33333 4.151542 3.096338 1528.2222 0.7025518  
## 5 11109 656953.2 5169574 17.55556 4.926763 3.410201 1660.8889 0.8694444  
## 6 11110 661774.8 5171761 15.25000 5.554083 4.289583 904.0000 0.8027500  
##      T_wat      pH  Ox_sat  Ox_dis N_tot      NO3      NH4      P  
## 1 7.403571 7.853571 111.8593 11.57857 NA 0.930000 0.04178571 0.02642857  
## 2 8.175000 8.200000 104.0900 11.17500 NA 1.300000 0.10500000 0.03250000  
## 3 6.861364 7.890909 108.4798 11.98864 NA      NA 0.05613636 0.03590909  
## 4 8.842857 7.964286 107.3411 11.60000 NA      NA 0.05642857 0.03214286  
## 5 7.524742 7.861856 107.9668 12.27010 NA      NA 0.04247423 0.02731959  
## 6 8.004167 7.991667 107.3012 12.26250 NA 1.804167 0.04166667 0.02875000
```



```

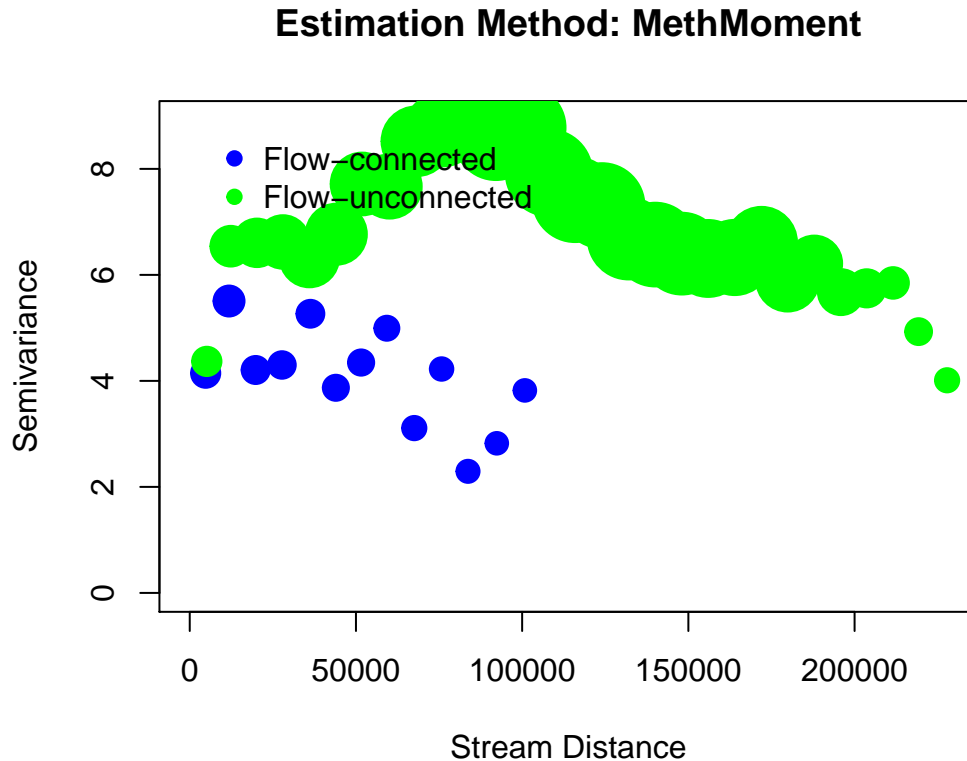
##      T_air   upDist   h2oArea   h2oAgric   h2oWood       Z   buf_agric_
## 1 9.546428571 216962.2 232.0061 1.241550 44.34142 1301.8166 661581
## 2      16 206977.5 478.6735 8.154412 87.76638 888.1400 1430320
## 3 12.52727273 197260.4 891.5869 31.958520 201.14650 864.1875 1997098
## 4 13.52142857 180682.9 1293.6352 66.280986 317.37853 601.6891 2187835
## 5 10.95463918 167065.8 1625.0196 95.918329 394.75059 510.3274 2640478
## 6      13.25 161210.9 1702.2652 108.820232 420.01351 352.2998 1694594
##   buf_wood_a mean_slope   afvArea NEAR_FID   NEAR_DIST   NEAR_X   NEAR_Y
## 1 809990.7 7.3127922 0.02774349      523 6.958429e-05 616913.2 5174929
## 2 1098029.0 0.7420453 0.05394963      517 9.985782e-05 620699.3 5167181
## 3 195539.0 0.1000978 0.09902194      142 7.374326e-06 629714.7 5164033
## 4      0.0 1.1382939 0.13388565      488 1.249576e-05 644784.2 5165332
## 5 339981.4 0.6819947 0.15811525      219 4.979581e-05 656954.0 5169559
## 6 558753.6 4.3350407 0.16251227      520 1.658291e-05 661774.9 5171761
##   NEAR_ANGLE rid      ratio locID netID pid  PCA1.long LIMeco buf.agr.p
## 1 169.24863 522 0.54934191      1      1 1 -1.0340673 0.870 21.05877
## 2 -141.68478 516 0.33461351      2      1 2 -0.4285926 0.850 45.52848
## 3 85.16966 141 0.07233777      3      1 3 0.2011001 0.820 63.56959
## 4 103.92335 487 0.74509422      4      1 4 0.7775352 0.680 69.64093
## 5 93.35882 218 0.12080495      5      1 5 1.0154177 0.865 84.04901
## 6 -100.84065 519 0.59876641      6      1 6 1.3097225 0.880 53.94058
##   buf.wood.p h2oAgric_p h2oWood_p feed_gra   feed_xyl   feed_shr   feed_gat
## 1 25.782801 0.535137 19.11218 0.3712922 0.005875961 0.08863374 0.2342132
## 2 34.951344 1.703544 18.33533 0.5172471 0.003505701 0.06517135 0.2676625
## 3 6.224199 3.584454 22.56050 0.3996249 0.004156538 0.11885428 0.2804388
## 4 0.000000 5.123623 24.53385 0.3365782 0.002229260 0.03977609 0.2711229
## 5 10.821943 5.902595 24.29205 0.3464179 0.002363468 0.13569226 0.3128262
## 6 17.785677 6.392672 24.67380 0.3350684 0.005149049 0.05198261 0.2312810
##   feed_pff feed_pre feed_oth
## 1 0.14450043 0.1554770 0.02500710
## 2 0.04509914 0.1012767 0.02503563
## 3 0.08946262 0.1074478 0.02501434
## 4 0.27234322 0.0778204 0.02512339
## 5 0.09827923 0.1044210 0.02500000
## 6 0.26211074 0.1144082 0.02500000

```

## Exploring variograms for river networks. the *Torgegram*

Explore default *torgegrams*. SSN package offers this default plots for exploring variograms. It shows the variance for flow-connected and flow-unconnected observation pairs. The size of the spots reflect the #observation pairs for each distance lag. Here an example of *torgegram* for 'shannon diversity'. *Clearly different spatial patterns for flow-connected and unconnected locations.*

```
tor.shn=Torgegram(spat.course.ssn, 'shannon', nlag = 50, maxlag = 400000)
plot(tor.shn)
```



Let's see how the `torgegram` object looks like.

It shows the distance lags, the variance ('gam') and the n. of observation pairs within the lag ('np'). And it does so for flow-connected and unconnected locations.

```
tor.shn

## $distance.connect
## [1] 4788.988 11824.163 19830.945 27750.220 36303.389 43957.028
## [7] 51547.647 59295.204 67544.649 75759.661 83708.374 92369.460
## [13] 100813.985
##
## $gam.connect
## [1] 4.144887 5.506717 4.205762 4.300243 5.266434 3.870225 4.346274 4.993600
## [9] 3.109768 4.224341 2.294015 2.822745 3.820626
##
## $np.connect
## [1] 114 141 97 89 94 67 69 51 40 28 22 15 15
##
## $distance.unconnect
## [1] 5132.361 12348.225 20231.462 28030.915 36001.415 44093.875
```

```
## [7] 51847.018 60120.128 68193.509 76080.959 84185.824 92101.997
## [13] 100062.904 108060.507 115917.945 124009.023 132043.921 140025.045
## [19] 148057.577 155961.784 163934.849 172010.235 179866.338 187863.722
## [25] 195893.623 203638.621 211588.487 219258.639 227831.461
##
## $gam.unconnect
## [1] 4.367907 6.541150 6.604840 6.621026 6.323810 6.767396 7.714934 7.668727
## [9] 8.520837 8.749968 8.922053 8.576814 8.779165 7.925553 7.420135 7.306521
## [17] 6.685355 6.571591 6.401043 6.309791 6.328198 6.614536 5.900969 6.215039
## [25] 5.674788 5.746042 5.848463 4.929490 4.011468
##
## $np.unconnect
## [1] 116 287 405 480 565 594 620 642 725 796 880 931 974 976 953 953 906 936 906
## [20] 833 807 742 620 515 367 249 150 79 39
##
## $call
## $call$object
## spat.course.ssn
##
## $call$ResponseName
## [1] "shannon"
##
## $call$maxlag
## [1] 4e+05
##
## $call$nlag
## [1] 50
##
## attr("class")
## [1] "Torgegram"
```

Variograms can be better examined including the three distance types.

Exploring the shapes of variograms over the Euclidean, flow-unconnected and flow-connected dimensions provide more insight into the main spatial patterns of the data.

Let's examine the variogram for water T, along the three distances.

```
# first the classic variogram over Euclidean dist
emp.var.wt=EmpiricalSemivariogram(spat.course.ssn, 'T_wat', nlag = 30, maxlag = 200000)
#then the togegram for flow-connected and unconnected dist
tor.wt=Torgegram(spat.course.ssn, 'T_wat', nlag = 50, maxlag = 400000)
```

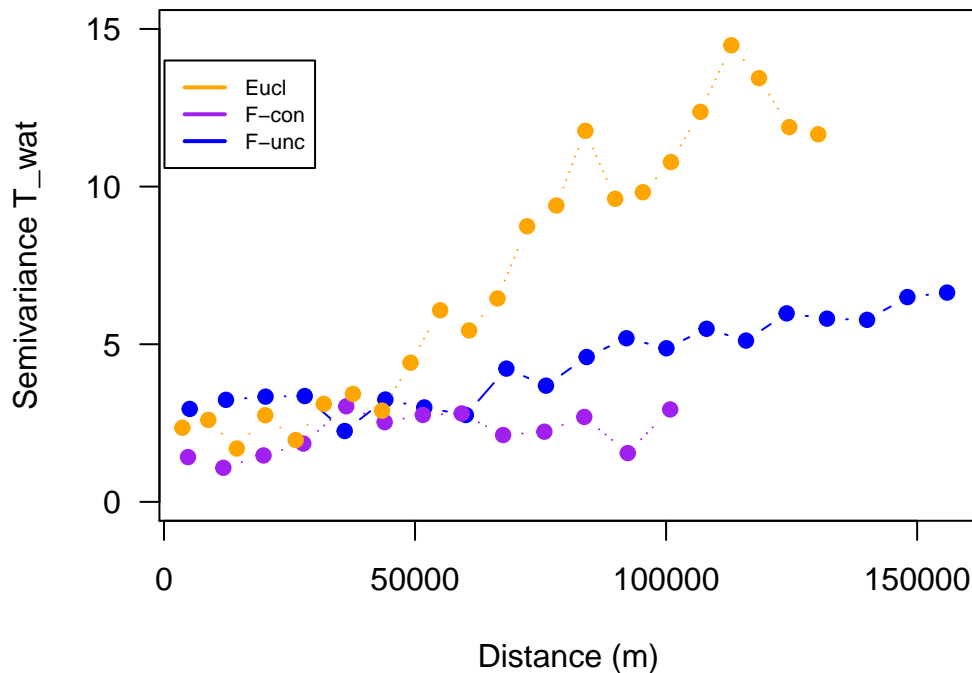
Plot all distances together

```
plot(tor.wt$gam.unconnect[1:20]~tor.wt$distance.unconnect[1:20], options(scipen = 5),
     type='b', lty=1, pch=19, ylim=c(0,15), col='blue', ylab='Semivariance T_wat',
     xlab='Distance (m)', cex.lab=1, las=1)
```

```

lines(tor.wt$gam.connect[1:13]~tor.wt$distance.connect[1:13], type='b', col='purple',
      lty=3, pch=19)
lines(emp.var.wt$gamma[1:23]~emp.var.wt$distance[1:23],col='orange', type='b',
      pch=19, lty=3 )
legend(100, 14, legend=c("Eucl", "F-con", 'F-unc'),
      col=c("orange", "purple", 'blue'), lty=c(1,1,1), lwd=2, cex=0.7)

```



The above plot shows that, for a given distance, water T varies little over the watercourse dimension, and more over the Euclidean dimension, but only beyond 50 km distance.

Now let's examine the variogram for local land-use

```

# first variogram along the Euclidean dimension
emp.var.agr=EmpiricalSemivariogram(spat.course.ssn, 'buf.agr.p', nlag = 30, maxlag = 200000)
# then the torgregram over flow-connected and unconnected distances
tor.agr=Torgregram(spat.course.ssn, 'buf.agr.p', nlag = 50, maxlag = 400000)

```

Plot it together

```

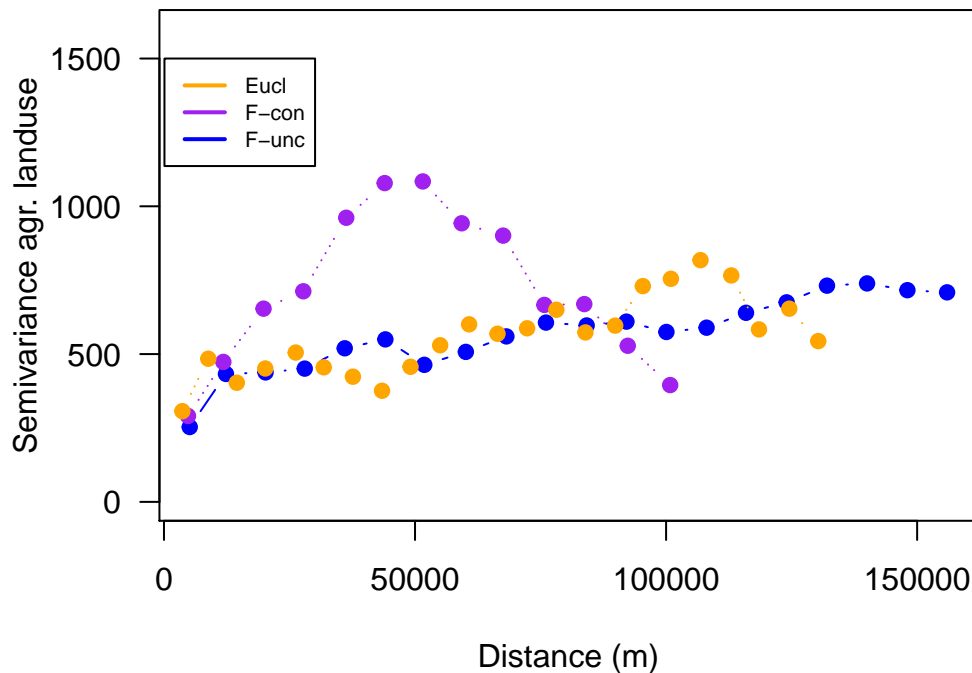
plot(tor.agr$gam.unconnect[1:20]~tor.agr$distance.unconnect[1:20], options(scipen = 5),
     type='b', lty=1, pch=19, ylim=c(0,1600 ), col='blue', ylab='Semivariance agr. landuse',
     xlab='Distance (m)', cex.lab=1, las=1)

```

```

lines(tor.agr$gam.connect[1:13]~tor.agr$distance.connect[1:13], type='b',
      col='purple', lty=3, pch=19)
lines(emp.var.agr$gamma[1:23]~emp.var.agr$distance[1:23],col='orange',
      type='b', pch=19, lty=3 )
legend(100, 1500, legend=c("Eucl", "F-con", "F-unc"),
      col=c("orange", "purple", 'blue'), lty=c(1,1,1), lwd=2, cex=0.7)

```



*# Here we see that local landuse varies more rapidly along flow-connected locations  
# reflecting the changes from headwaters to mainstems*

The above variogram show that agricultural landuse changes rapidly along flow-connected locations, reflecting changes from up to downstream within ~50km.

As extreme case, we can explore the variogram for the variable *PCA1.long*. This is a synthetic variable refelecting the logitudinal continuum. As expected, the variance increases along flow-connected locations, but remains low over the other dimensions.

```

# first the variogram for Euclidean and then the Torgegram
emp.var.PCA1=EmpiricalSemivariogram(spat.course.ssn, 'PCA1.long', nlag = 30, maxlag = 200000)
tor.PCA1=Torgegram(spat.course.ssn, 'PCA1.long', nlag = 50, maxlag = 400000)

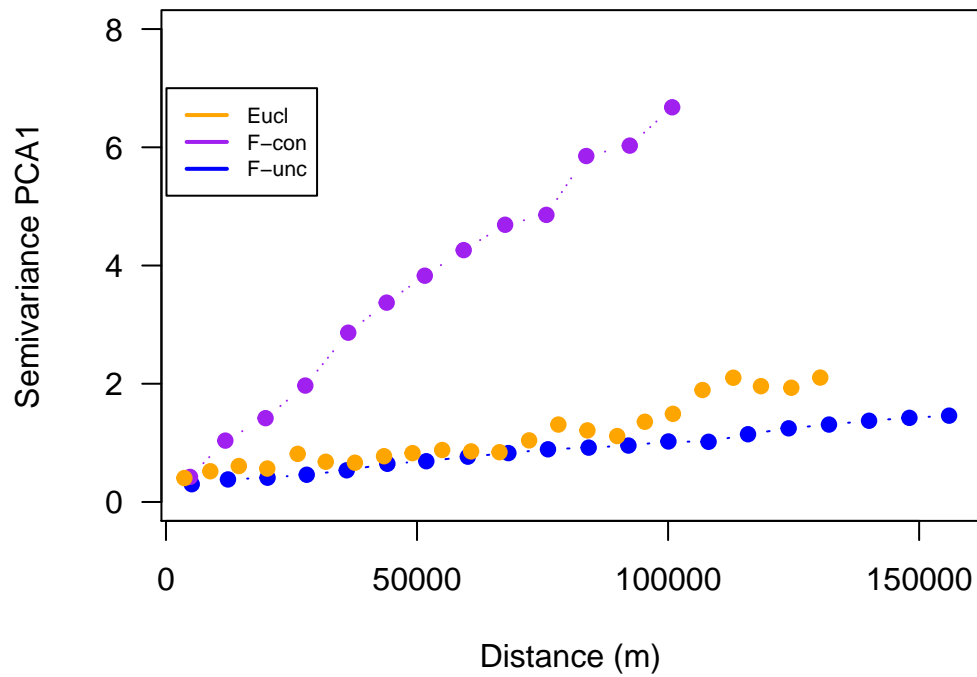
```

Plot it together as before

```
plot(tor.PCA1$gam.unconnect[1:20]~tor.PCA1$distance.unconnect[1:20], options(scipen = 5),
     type='b', lty=1, pch=19, ylim=c(0,8), col='blue', ylab='Semivariance PCA1',
     xlab='Distance (m)', cex.lab=1, las=1)

lines(tor.PCA1$gam.connect[1:13]~tor.PCA1$distance.connect[1:13], type='b', col='purple',
      lty=3, pch=19)
lines(emp.var.PCA1$gamma[1:23]~emp.var.PCA1$distance[1:23], col='orange', type='b',
      pch=19, lty=3 )

legend(100, 7, legend=c("Eucl", "F-con", 'F-unc'),
      col=c("orange", "purple", 'blue'), lty=c(1,1,1), lwd=2, cex=0.7)
```



Now let's run some models. Explore the influence of key environmental drivers on macroinvertebrate metrics.

First, have a look again at the variables in the data;

```
names(spat.course.ssn)
```

```
## $Obs
## [1] "code"      "X"          "Y"          "richness"   "shannon"
## [6] "simpson"   "density"    "Star_icmi"  "T_wat"      "pH"
## [11] "Ox_sat"    "Ox_dis"     "N_tot"      "NO3"        "NH4"
## [16] "P"         "T_air"      "upDist"     "h2oArea"    "h2oAgric"
## [21] "h2oWood"   "Z"          "buf_agric_" "buf_wood_a" "mean_slope"
## [26] "afvArea"   "NEAR_FID"   "NEAR_DIST"  "NEAR_X"     "NEAR_Y"
## [31] "NEAR_ANGLE" "rid"        "ratio"      "locID"      "netID"
## [36] "pid"       "PCA1.long"  "LIMeco"     "buf.agr.p"  "buf.wood.p"
## [41] "h2oAgric_p" "h2oWood_p" "feed_gra"   "feed_xyl"   "feed_shr"
## [46] "feed_gat"  "feed_pff"   "feed_pre"   "feed_oth"
```

As example, we fit models for the porportion of shredders (*feed\_shr*) in the community.

We will include the longitudinal gradient (*PCA1.long*), the % of agricultural landuse (*buf.agr.p*) and water quality (*LIMeco*), as predictor (covariates).

```
shr.0=glmssn(feed_shr~PCA1.long+buf.agr.p+LIMeco,spat.course.ssn,
             CorModels = NULL, use.nugget = TRUE,EstMeth = "ML")

summary(shr.0)
```

Start with a non-spatial model (CorModels=NULL). This is equivalent to a classic lm.

```
##
## Call:
## glmssn(formula = feed_shr ~ PCA1.long + buf.agr.p + LIMeco, ssn.object = spat.course.ssn,
##       CorModels = NULL, use.nugget = TRUE, EstMeth = "ML")
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.074099 -0.026835 -0.006961  0.017427  0.161439
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.0014740  0.0277824   0.053   0.958
## PCA1.long    -0.0043976  0.0035455  -1.240   0.216
## buf.agr.p     -0.0007341  0.0001541  -4.763 < 2e-16 ***
## LIMeco        0.1322798  0.0322791   4.098  0.00006 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Covariance Parameters:
## Covariance.Model Parameter Estimate
##           Nugget   parsill  0.00162
##
## Residual standard error: 0.04026982
## Generalized R-squared: 0.3545288
```

We see that both landuse and water quality are important factors.

Now we can fit some spatial models. We start with the full autocovariance functions (Euclidean, Tail.up, Tail.down).

```
shr.1=glmssn(feed_shr~PCA1.long+buf.agr.p+LIMeco,spat.course.ssn,
             CorModels = c("Exponential.tailup", "Exponential.taildown",
                           "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")
summary(shr.1)
```

```
##
## Call:
## glmssn(formula = feed_shr ~ PCA1.long + buf.agr.p + LIMeco, ssn.object = spat.course.ssn,
##       CorModels = c("Exponential.tailup", "Exponential.taildown",
##       "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.073192 -0.023236 -0.003967  0.021409  0.166026
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.0141687  0.0279533   0.507  0.61283
## PCA1.long    -0.0063257  0.0037939  -1.667  0.09709 .
## buf.agr.p    -0.0007112  0.0001586  -4.484  0.00001 ***
## LIMeco       0.1110628  0.0327433   3.392  0.00084 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##      Covariance.Model Parameter      Estimate
##      Exponential.tailup   parsill    0.0003413
##      Exponential.tailup   range 1015608.7873397
##      Exponential.taildown  parsill    0.0000703
##      Exponential.taildown  range  8916.6028923
##      Exponential.Euclid   parsill    0.0002278
##      Exponential.Euclid   range  5548.2069642
##      Nugget               parsill    0.0009745
##
## Residual standard error: 0.04017313
## Generalized R-squared: 0.3256791
```



Here we see a different output. The variogram parameters for each autocovariance functions are also given. We also see that the model parameters (*Estimate*, *Std.Error*) are different from the non-spatial model. We also see that the Sill of the tail.down component is very small. We could eventually omit it from the model.

**Model selection is a stepwise process. Once we are happy with the covariates (the fixed effect), we can work on the autocorrelation structures.**

Here we fit a couple of additional models with different variogram functions.

```
shr.2=glmssn(feed_shr~PCA1.long+buf.agr.p+LIMeco,spat.course.ssn,
             CorModels = c( "LinearSill.tailup",
                           "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")

#summary(shr.2)
```

```
shr.3=glmssn(feed_shr~PCA1.long+buf.agr.p+LIMeco,spat.course.ssn,
             CorModels = c( "Exponential.tailup"), addfunccol = "afvArea", EstMeth = "ML")

#summary(shr.3)
```

```
shr.4=glmssn(feed_shr~PCA1.long+buf.agr.p+LIMeco,spat.course.ssn,
             CorModels = c( "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")

#summary(shr.4)
```

We can then use the command *InfoCritCompare* to compare the different models for e.g. Root Mean Square Prediction Error (RMSPE) and AIC.

```
InfoCritCompare(list(shr.0, shr.1, shr.2, shr.3, shr.4))
```

```
##                                formula EstMethod
## 1 feed_shr ~ PCA1.long + buf.agr.p + LIMeco      ML
## 2 feed_shr ~ PCA1.long + buf.agr.p + LIMeco      ML
## 3 feed_shr ~ PCA1.long + buf.agr.p + LIMeco      ML
## 4 feed_shr ~ PCA1.long + buf.agr.p + LIMeco      ML
## 5 feed_shr ~ PCA1.long + buf.agr.p + LIMeco      ML
##                                Variance_Components
## 1                                Nugget
## 2 Exponential.tailup + Exponential.taildown + Exponential.Euclid + Nugget
## 3                        LinearSill.tailup + Exponential.Euclid + Nugget
## 4                                Exponential.tailup + Nugget
## 5                        Exponential.Euclid + Nugget
##      neg2LogL      AIC      bias      std.bias      RMSPE      RAV
## 1 -699.3530 -689.3530 0.000009075508 0.00003360497 0.04101413 0.04069292
## 2 -708.8503 -686.8503 0.000309403257 0.00117571626 0.03891968 0.03920649
## 3 -709.6860 -691.6860 0.000321798906 0.00123339173 0.03893167 0.03884631
## 4 -707.2990 -693.2990 0.000293362168 0.00111114130 0.03926642 0.03932925
## 5 -703.7241 -689.7241 0.000031537393 0.00011793856 0.03979320 0.03956483
##      std.MSPE      cov.80      cov.90      cov.95
```

```
## 1 1.0184230 0.8205128 0.9076923 0.9487179
## 2 0.9955477 0.8153846 0.8923077 0.9435897
## 3 1.0098952 0.8102564 0.8974359 0.9435897
## 4 1.0078079 0.8153846 0.8871795 0.9435897
## 5 1.0138924 0.8307692 0.9128205 0.9384615
```

We can print specific values for clarity (e.g. Spatial components, AIC, RMSPE)

```
InfoCritCompare(list(shr.0, shr.1, shr.2, shr.3, shr.4))[,c(3,5,8,12)]
```

```
##                                     Variance_Components
## 1                                     Nugget
## 2 Exponential.tailup + Exponential.taildown + Exponential.Euclid + Nugget
## 3                               LinearSill.tailup + Exponential.Euclid + Nugget
## 4                               Exponential.tailup + Nugget
## 5                               Exponential.Euclid + Nugget
##      AIC      RMSPE    cov.90
## 1 -689.3530 0.04101413 0.9076923
## 2 -686.8503 0.03891968 0.8923077
## 3 -691.6860 0.03893167 0.8974359
## 4 -693.2990 0.03926642 0.8871795
## 5 -689.7241 0.03979320 0.9128205
```

We see that spatial models including tail-up and/or tail-down (accounting for river topology) provide better predictions (lower RMSPE) than simple non-spatial model (shr.0), or models that only include Euclidean component (shr.4).

The *varcomp* function provide information on the variance components of the model (covariates + spatial components)

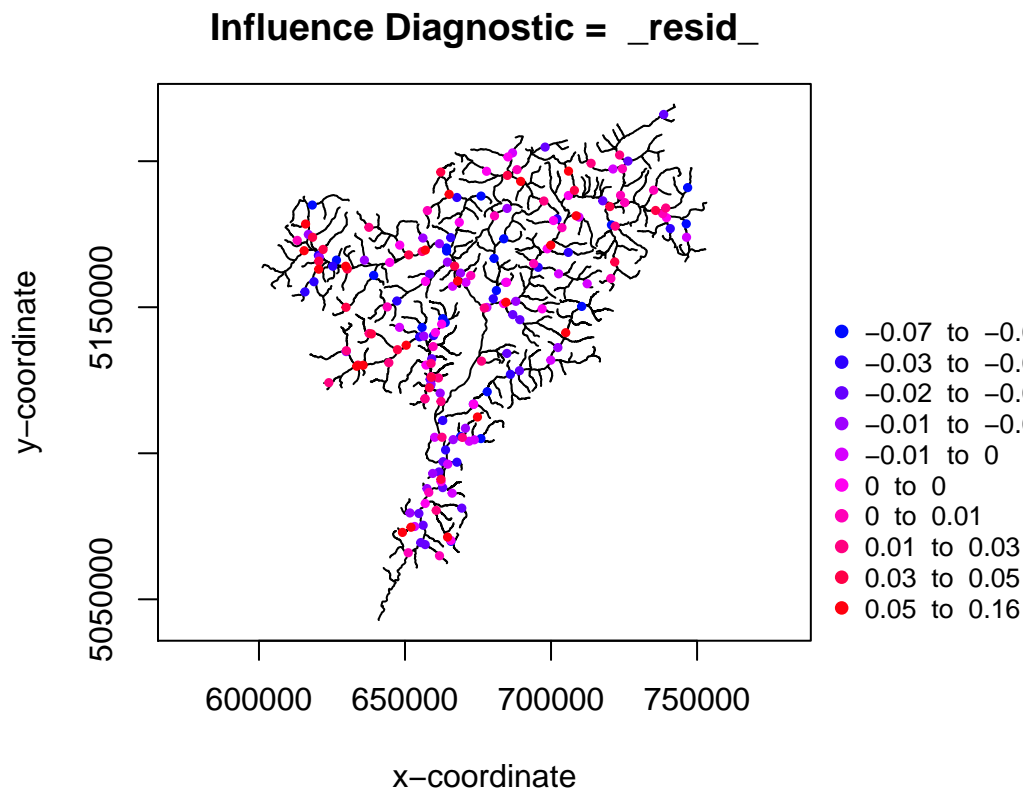
```
varcomp(shr.2)
```

```
##      VarComp Proportion
## 1 Covariates (R-sq) 0.33073666
## 2 LinearSill.tailup 0.15081926
## 3 Exponential.Euclid 0.43798818
## 4      Nugget 0.08045591
```

In the model *shr.2*, the included covariates (PCA1.long, Limeco, Buf.agr.p) explained 34% of variation. The tail-up autocovariance (reflecting correlation along flow-connected locations) accounted for 13% of variation, while the Euclidean covariance for 27%. The Nugget reflects the unexplained variance (with no spatial structure).

We can also plot the residuals of the model easily; this helps visualising spatial patterns in the residuals.

```
plot(resid(shr.2), asp=1, cex=0.5)
```



## Exercise that you can try yourself:

- Plot additional variograms for other variables (e.g. Limeco, Shannon)
- Fit models for the proportion of predators (*feed\_pre*) using the same covariates used before.
- Try to fit both non-spatial and spatial models (with different covariance functions).
- Compare the models using *InfoCritCompare*.