Appendix S4: User Code Documentation

This appendix provided details of the R functions provided for practitioners to use on their own data. Five tutorial examples are provided to demonstrate how to implement the code.

Installation Instructions

Download and unzip the file BlockBootCode. Open R. At the command line, run: source("<<file pathname>>\ LoadFunctions.R"), inserting the path name to the BlockBootCode folder. The CRAN packages dplyr and geoR are required (for variogram fitting).

The R markdown files to run the examples are also provided in the zip file (e.g. Example1.Rmd). These can be run by opening with RStudio and clicking Knit to Html.

Functions provided

BlockBootApply and **BlockBootID** are two functions to implement the block bootstrap and perform block size selection according the emprirical MSE method (Lahiri and Zhu, 2006).

select_b_length_off_variogram returns the practical range of a variogram.

CombineBatchesofBootstraps combines **BlockBootApply** objects that may have been submitted in parallel to a cluster computer.

Usage

BlockBootID (x, y, block_Ls, NBoot = 500, Grid_space = NA, shape = "disc", sampling_type = "area")

BlockBootApply (x, y, block_Ls, Stat.function, dat, NBoot = 500, Grid_space = NA, type = "SE" method.block.length.select = NA, lookuptables.folderpath = NA, subregion_n_x = 3, subregion_n_y = 3, shape = ”disc”, sampling_type = “area”, long_format_required = FALSE, ...)

select_b_length_off_variogram_envelope (x,y, resids, breaks, ini, max.dist, plot = TRUE,...)

CombineBatchesofBootstraps (list_of_boot_batches)

Arguments to BlockBootID and BlockBootApply

x vector of the x co-ordinates of sites (in Cartesian co-ords like UTM)

y vector of the y co-ordinates of sites (in Cartesian co-ords like UTM)

block_Ls vector of block size parameters to run block bootstrap with. For BlockBootID, only one block size may be entered.

dat A dataframe, argument to Stat.function. Each row should be a site.

NBoot The number of bootstrap replicates to run. Defaults to 500. It is recommended to use a small NBoot first to test run code.

Grid_space The space between sampling points. The smaller the space, the more sampling points there will be available to sample (and the more overlapping blocks may be), but the longer the resampling algorithm will take. If left blank this is assigned to be 1/3 f the smallest block length.

Stat.function Function for a statistic T. This should take a dataframe dat as an argument with sites as rows, and may take unlimited additional arguments. However all site level data should be contained in dat (as this will be resampled).

type Either “SE” or “Pval”. Whether to calculate a Standard error or p-value (one sided, upper tail) from the bootstrap distribution of the statistic. Raw bootstrap distribution values of the statistic T are also returned in case of other uses.

method.block.length.select What method to use to select a block length. May take values “Lahiri” or NA. IF NA and block_Ls is of length 1 the result is given for that block length.

lookuptables.folderpath A folder pathname for saving “lookup tables”. Saving lookup tables (tables identifying sites with sampling points, based on the block length) can greatly speed up resampling time, if the same (x,y) data is going to be resampled multiple times, fro example if submitting batches of resamples in parallel . If this argument is present, the function looks in the folder to see if a lookup table (for those sites, with that block length) exists and if it does, uses that table. We recommend supplying this argument.

m_x, m_y Numbers giving how many subregions to divide the region into for the Lahiri block length selection method. The region will be divided in m_x horizontally and m_y vertically. Defaults to 3×3.

tuning_block_length Index giving which block length to use as the tuning block length in the Lahiri block length selection method. E.g. if tuning_block_length = 2, the second element of block_Ls is used as tuning block length.

shape Either “disc”(default) or ”square”. The shape of the block.

sampling_type Either “area”(default) or “sites”. This gives the stopping criteria to the bootstrap resampling. i.e. Keep resampling blocks of sites, until the cumulative area of the blocks sampled is equal to the *area* of the observation region. Alternatively, keep resampling blocks of sites until the number of sites in the resample is equal to the number of sites in the observation region. We did check sensitivity of results to this criteria and did not find this to be an important distinction for our results, but used “area” throughout the results in this manuscript. However for some applications- e.g. to integrate the block bootstrap with other software using BlockBootID, the other software may require input as a matrix, in which case it is desirable to have all resamples contain the same number of sites.

long_format_required If dat (and hence resampled versions of dat) needs to be transformed to long format for a multispecies example, e.g. as in the 4th corner example in this paper, then set this to TRUE.

(...) Additional arguments to Stat.function

Arguments to select_b_length_off_variogram

resids vector of residuals for variogram

breaks Argument to geoR::variog

ini, max.dist Argument to geoR::variofit

(...) Additional arguments to variog and variofit

Arguments to CombineBatchesofBootstraps

list_of_boot_batches A list of **BlockBootApply** objects with the same arguments.

Description

BlockBootID calculates an ID matrix. The number of rows will be the number of bootstrap resamples. The number of columns will be the number of sites.

BlockBootApply does lots of stuff...

select_b_length_off_variogram_envelope selects a block length based on a variogram of the residuals supplied. It first fits an empirical variogram (using geoR::variog), then fits the practical range of that variogram (using geoR::variofit). It returns a block length equal to the practical range. This block length can then be used as an argument (to block_Ls or tuning block length) in BlockBootID or BlockBootApply. It is recommended to check variogram plots and check sensitivity to breaks, max.dist and ini. See geoR documentation.

If TestStatFunction takes a while to compute, then it is recommended to do some of the bootstrap resample computing in parallel. Instead of NBoot = 1000 for example, run 50 lots of NBoot = 20, submitted in parallel to a cluster computer. One way to estimate running time is to first running BlockBootApply with a small NBoot (e.g. 2), the time increases approximately proportional to NBoot. Things that decrease running time: changing (increasing) Grid_space (a very fine grid space means the lookup tables (which say which sites are within the block associated with those grid co-ordinates) will take a very long time to calculate); searching over fewer block lengths for the optimal block length; decreasing NBoot; including a pathname to LookupTables so that you save the table associating sites and grid co-ordinates.

Values of BlockBootApply

boot.reps.of.Stat.function A list of length(block_Ls). Each list element contains the bootstrap distribution of Stat (so a list of length NBoot).

Stat The Statistic which is being bootstrapped

Lahiri_block_size If method.block.length.select = ”Lahiri”, the Lahiri chosen block length

Empirical.MSE If method.block.length.select = ”Lahiri”, a table of the Empirical MSE for each block length (rows). Columns indicate the tuning block length. So for a given tuning block length the row which contains the minimum value for that column is the block length which minimises the empirical MSE.

SE.estimate The Standard Error Estimate of the statistic

sigma_stat_subregion Vector of estimates of SE(T)- standard error of statistic T, for each block length in block_Ls, on each subregion

n_in_subregions The number of sites in each subregion. Singularities will occur if subregions have zero sites.

sigma_stat_hat Vector of estimates of SE(T)– standard error of statistic T, for each block length in block_Ls

block_Ls Argument supplied to function

tuning_block_length Argument supplied to function

EXAMPLES

These examples are to help users understand the mechanics of how to operate the functions on their own data.

Example 1 Gets the standard error of a regression coefficient via block bootstrapping (BlockBootApply), selecting the block size using the Lahiri method.

Example 3 As in Example 2 except with block size selection via BlockBootApply, (Lahiri method).

Example 4 Gets the standard error of the AUC of a model via block bootstrapping (BlockBootApply), selecting the block size using the Lahiri method.

Example 5 Demonstrates the use of CombineBatchesofBootstraps. Runs Example 4, and assembles results as if batches of bootstrap resamples have been submitted in parallel to a cluster computer.

Example 1: Standard Error of a Coefficient

Setting Up

First load the data and R functions you need into the R workspace.

```
load("PlayData_for_Examples.RData") ### load the data for the examples
source("LoadFunctions.R") #### Source the functions required to run the block bootstrap
```

Motivation

Imagine we want to know the standard error of our estimate for β- the coefficient of temperature. First fit the model.

```
set.seed(42)


fit1 = lm (response ~ temperature, data = dat)
summary(fit1)
```

```
##
## Call:
## lm(formula = response ~ temperature, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5766 -0.9687  0.0120  0.9065  5.1925
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.03474    0.06584   0.528    0.598
## temperature  1.14190    0.06975  16.372   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.455 on 498 degrees of freedom
## Multiple R-squared:  0.3499, Adjusted R-squared:  0.3486
## F-statistic:   268 on 1 and 498 DF,  p-value: < 2.2e-16
```

Getting Variogram Practical Range

Fit a variogram on the residuals- notice we can see spatial autocorrelation. Use the variogram practical range as a tuning parameter for the Lahiri method of block selection. Always check the variogram visually. Sometimes you may need to change max.dist, breaks or initial conditions to get a sensible practical range. It is hard to automate this process.

```
ini.vals <- expand.grid(seq(0,10,l=100), seq(0,1,l=100)) #inital values for variofit
variogram_block_length = select_b_length_off_variogram_envelope(x,y,resids = fit1$resid,
max.dist = 0.4, breaks=c(0,0.025,0.05,0.075,0.1,0.15,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1.5),ini=ini.vals)
```

```
## variog: computing omnidirectional variogram
## variofit: covariance model used is matern
## variofit: weights used: cressie
## variofit: minimisation function used: optim
## variofit: searching for best initial value ... selected values:
##               sigmasq phi    tausq kappa
## initial.value "2.02"  "0.02" "0"   "0.5"
## status        "est"   "est"  "est" "fix"
## loss value: 95.7219934941921
```

```
variogram_block_length
```

```
## [1] 0.20739591 0.06922791
```

Coding StatFunction argument

Define BetaCoeff a function which will be the *StatFunction* argument to *BlockBootApply*. BetaCoeff gets a coefficient for temperature for inference

```
BetaCoeff = function(dat){
  fit1 = lm ("response~temperature", data=dat)
  Beta = fit1$coefficients["temperature"]
  Beta
}
```

Run Block Bootstrap

First the user should create a folder called e.g. LookupTables in their R working directory. As all the examples below use the same site coordinates they all share the same lookup tables. However if a new analysis is being done on different data (with different site co-ordinates), then a new lookup table needs to be created.

```
lookuptables.folderpathname = "LookupTables/"
```

Run Block Bootstrap, using Lahiri method of block size selection, with the variogram to select the tuning parameter. We search over 4 blocks sizes (0- i.e. IID, 0.05, 0.1, 0.2). BlockBootApply uses the size which minimises the Empirical MSE of SE(β). Use 0.2 as tuning block length as per variogram practical range (tuning block length =4).

```
Results = BlockBootApply (x = x ,y = y ,block_Ls = c(0,0.05,0.1, 0.2), Grid_space = 0.01 ,
                          dat = dat ,
                          Stat.function = BetaCoeff,  tuning_block_length = 0.2,
                          NBoot = 500, method.block.length.select = "EmpiricalMSE",
                          type="SE", lookuptables.folderpath=lookuptables.folderpathname)
```

```
## [1] "has foldername, checking for lookuptable"
## [1] "file exists"
## [1] "using lookup"
## [1] "has foldername, checking for lookuptable"
## [1] "file exists"
## [1] "using lookup"
## [1] "has foldername, checking for lookuptable"
## [1] "file exists"
## [1] "using lookup"
## [1] "mutually.exclusive"
## [1] "subregion1"
## [1] "mxy33"
## [1] "subregion2"
## [1] "mxy33"
## [1] "subregion3"
## [1] "mxy33"
## [1] "subregion4"
## [1] "mxy33"
## [1] "subregion5"
## [1] "mxy33"
## [1] "subregion6"
## [1] "mxy33"
## [1] "subregion7"
## [1] "mxy33"
## [1] "subregion8"
## [1] "mxy33"
## [1] "subregion9"
## [1] "mxy33"
## [1] "subregion1"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion2"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion3"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion4"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion5"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion6"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion7"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion8"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion9"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion1"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion2"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion3"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion4"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion5"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion6"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion7"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion8"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion9"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion1"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion2"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion3"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion4"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion5"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion6"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion7"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion8"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
## [1] "subregion9"
## [1] "mxy33"
## [1] "file exists"
## [1] "using existing coords..."
```

Check sensitivity to tuning parameter. See that some columns are minimised by block length 0.1, and some columns by 0.05 so tuning block length was imporant here

```
Results$Empirical.MSE
```

```
##                  tuninglength0 tuninglength0.05 tuninglength0.1
## blocklength0      0.0003547597     0.0006171809     0.002031718
## blocklength0.05   0.0008426232     0.0004558952     0.001049705
## blocklength0.1    0.0027512821     0.0015788562     0.001179298
## blocklength0.2    0.0016315219     0.0013584784     0.002096021
##                  tuninglength0.2
## blocklength0         0.004278006
## blocklength0.05      0.002573408
## blocklength0.1       0.001828421
## blocklength0.2       0.003746269
```

The block size parameter chosen is:

```
Results$EmpiricalMSE_block_size
```

```
## tuning = 0.2
##          0.1
```

This is the standard error (estimated via block bootstrap to account for spatial autocorrelation) of β

```
Results$SE.estimate.EmpiricalMSE_block_size
```

```
## [[1]]
## [1] 0.1167922
```

Example 2: Community-level Hypothesis Test using BlockBootID with mvabund

Setting Up

First load the data and R packages you need into the R workspace.

```
load("PlayData_for_Examples.RData") ### load the data for the examples
source("LoadFunctions.R") #### Source the functions required to run the block bootstrap


#Install packages if required
if(!require(mvabund)) { install.packages("mvabund", repos = "http://cran.us.r-project.org");
require(mvabund) }
library(mvabund)
```

Motivation

We want to test an assemblage/ community of species for the significance of a bunch of variables, using "mvabund::summary".

This example assumes a block size is selected by the user already, from either pilot study or as in Example 3. They can then use **BlockBootID** to generate an ID matrix for input into mvabund.

```
set.seed(42)

lookuptables.folderpathname = "LookupTables/"

######## Get a bootID matrix
BootID.example2 = BlockBootID(x = x ,
                              y = y,
                              block_Ls = 0.1,
                              NBoot = 500,
                              Grid_space = 0.01,
                              lookuptables.folderpath =  lookuptables.folderpathname)
```

Run hypothesis test/ summary in mvabund

```
responseMultiSpecies=mvabund(multispecies_dat[,1:20]) #20 species multivariate reponse
mod.1  = manyglm(responseMultiSpecies~temperature, data = multispecies_dat,family="binomial")
mod.2  = manyglm(responseMultiSpecies~temperature*treatment, data = multispecies_dat,family="binomial")
anova.results = anova(mod.1, mod.2, bootID=BootID.example2, resamp="case")
```

```
## Warning in anova.manyglm(mod.1, mod.2, bootID = BootID.example2, resamp =
## "case"): 'montecarlo' or 'pit.trap' should be used for binomial regression.
## Warning in anova.manyglm(mod.1, mod.2, bootID = BootID.example2, resamp =
## "case"): case resampling with score and LR tests is under development. try
## case resampling with wald test.

## Using <int> bootID matrix from input.
## Time elapsed: 0 hr 0 min 29 sec
```

```
anova.results
```

```
## Analysis of Deviance Table
##
## mod.1: responseMultiSpecies ~ temperature
## mod.2: responseMultiSpecies ~ temperature * treatment
##
## Multivariate test:
##       Res.Df Df.diff   Dev Pr(>Dev)
## mod.1    498
## mod.2    496       2 37.42        1
## Arguments:
##  Test statistics calculated assuming uncorrelated response (for faster computation)
##  P-value calculated using 500 resampling iterations via case resampling (to account for correlation
```

The treatment is not significant.

Example 3: Community-level Hypothesis Test using BlockBootID with mvabund, block length selection

Setting Up

First load the data and R packages you need into the R workspace.

```
load("PlayData_for_Examples.RData") ### load the data for the examples
source("LoadFunctions.R") #### Source the functions required to run the block bootstrap


#Install packages if required
if(!require(mvabund)) { install.packages("mvabund", repos =
"http://cran.us.r-project.org"); require(mvabund) }

library(mvabund)
```

Motivation

```
set.seed(42)
treatment =rbinom(500,1,0.5)
multispecies_dat = data.frame(multispecies_dat,treatment)
```

Coding StatFunction argument

Define MultSpeciesTestStat a function which will be the *StatFunction* argument to *BlockBootApply*.

```
MultSpeciesTestStat = function(dat){
responseMultiSpecies = mvabund(dat[,1:20])

null.model = manyglm (responseMultiSpecies~1, data=dat, family="binomial")
alt.model = manyglm (responseMultiSpecies~temperature*treatment, data=dat, family="binomial")
Test.stat = sum(alt.model$two.loglike) - sum(null.model$two.loglike)
;
Test.stat
}

MultSpeciesTestStat (multispecies_dat)
```

```
## [1] 10261.29
```

Run Block Bootstrap

Run Block Bootstrap, using Lahiri method of block size selection. We search over 4 blocks sizes (0- i.e. IID, 0.05, 0.1, 0.2). BlockBootApply uses the size which minimises the Empirical MSE of SE(T), where T is the multispecies likelihood ratio test statistic. Use 0.05 as tuning block length, (tuning_block_length =2).

```
lookuptables.folderpathname = "LookupTables/"
```

```
Results.Example3 = BlockBootApply (x = x ,y = y ,block_Ls = c(0,0.05,0.1,0.2), Grid_space = 0.01 ,
dat = multispecies_dat , Stat.function = MultSpeciesTestStat,  tuning_block_length = 2, NBoot = 2,
method.block.length.select = "Lahiri",  type="SE", lookuptables.folderpath=lookuptables.folderpathname)


L.example3  = Results.Example3$Lahiri_block_size
#The chosen block size parameter
L.example3
```

```
## [1] 0.1
```

Now make a bootID matrix and feed into mvabund::summary.

```
BootID.example3 = BlockBootID(x = x ,
y = y,
block_Ls = L.example3,
NBoot = 500,
Grid_space = 0.01,
lookuptables.folderpath =  lookuptables.folderpathname, shape="disc")

#######

responseMultiSpecies = mvabund(multispecies_dat[,1:20])

mod.full  = manyglm(responseMultiSpecies~temperature*treatment, data = multispecies_dat,
family="binomial")

anova(mod.full, bootID = BootID.example3, resamp="case")
```

```
## Warning in anova.manyglm(mod.full, bootID = BootID.example3, resamp =
## "case"): 'montecarlo' or 'pit.trap' should be used for binomial regression.

## Warning in anova.manyglm(mod.full, bootID = BootID.example3, resamp =
## "case"): case resampling with score and LR tests is under development. try
## case resampling with wald test.

## Using <int> bootID matrix from input.
## Time elapsed: 0 hr 1 min 22 sec

## Analysis of Deviance Table
##
## Model: manyglm(formula = responseMultiSpecies ~ temperature * treatment,
## Model:     family = "binomial", data = multispecies_dat)
##
## Multivariate test:
##                       Res.Df Df.diff   Dev Pr(>Dev)
## (Intercept)              499
## temperature              498       1 10224    0.421
## treatment                497       1    19    0.990
## temperature:treatment    496       1    19    0.982
## Arguments:
##  Test statistics calculated assuming uncorrelated response (for faster computation)
##  P-value calculated using 500 resampling iterations via case resampling (to account for correlation
```

```
##########
```

Example 4: Standard Error of AUC

Setting Up

First load the data and R packages you need into the R workspace.

```
load("PlayData_for_Examples.RData") ### load the data for the examples
source("LoadFunctions.R") #### Source the functions required to run the block bootstrap


#Install packages if required
if(!require(pROC)) { install.packages("pROC", repos = "http://cran.us.r-project.org"); require(pROC) }

library(pROC)
```

Motivation

Imagine we want to know the standard error of the AUC (area under the ROC curve) of our model. Perhaps we have two models, and we want to see if variation in AUC is explained by sampling error or if it is due to the predictors in the model.

```
set.seed(42)

fit1 = glm (PresAbsresponse ~ temperature, data = dat, family="binomial")

auc(response=dat$PresAbsresponse, predictor= predict(fit1))
```

```
## Area under the curve: 0.7735
```

Coding StatFunction argument

Define getAUC a function which will be the *StatFunction* argument to *BlockBootApply*.

```
GetAUC = function(dat){
fit1 = glm ("PresAbsresponse~temperature", data=dat, family="binomial")
AUC = auc(response=dat$PresAbsresponse, predictor= predict(fit1))
;
AUC
}
```

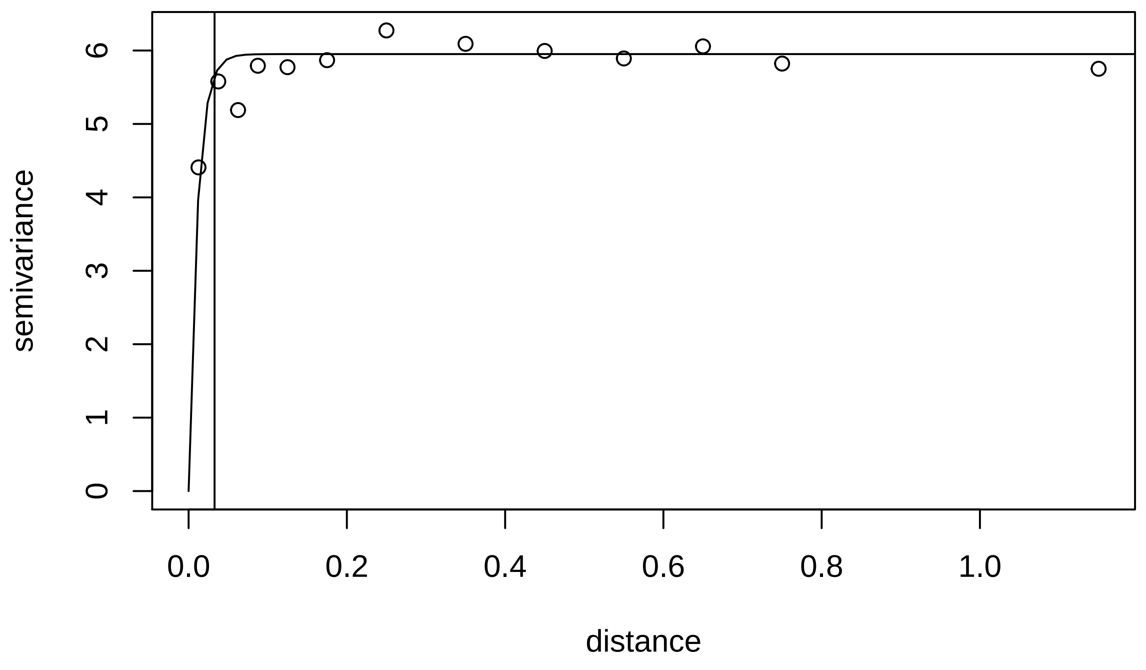
Getting Variogram Practical Range

Fit a variogram on the residuals of fit1- notice we can see spatial autocorrelation. Use the variogram practical range as a tuning parameter for the Lahiri method of block selection. Always check the variogram visually. Sometimes you may need to change max.dist, breaks or initial conditions to get a sensible practical range. It is hard to automate this process.

```
ini.vals <- expand.grid(seq(0,10,l=100), seq(0,1,l=100))


variogram_block_length = select_b_length_off_variogram_envelope(x,y,resids = fit1$resid,
max.dist = 1.2, breaks=c(0,0.025,0.05,0.075,0.1,0.15,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1.5),
ini=ini.vals)
```

```
## variog: computing omnidirectional variogram
## variofit: covariance model used is matern
## variofit: weights used: npairs
## variofit: minimisation function used: optim
## variofit: searching for best initial value ... selected values:
##               sigmasq phi    tausq kappa
## initial.value "5.96"  "0.01" "0"   "0.5"
## status        "est"   "est"  "est" "fix"
## loss value: 3853.7181489579
```



```
variogram_block_length
```

```
## [1] 0.03280132
```

Run Block Bootstrap

Run Block Bootstrap, using Lahiri method of block size selection, with the variogram to select the tuning parameter. We search over 4 blocks sizes (0- i.e. IID, 0.05, 0.1, 0.2). BlockBootApply uses the size which minimises the Empirical MSE of SE(β). Use 0.05 as tuning block length as per variogram practical range ($tuning_block_length = 2$).

```
lookuptables.folderpathname = "LookupTables/"
```

```
Results.example4 = BlockBootApply (x = x ,y = y ,block_Ls = c(0,0.05,0.1,0.2), Grid_space = 0.01 ,
dat = dat , Stat.function = GetAUC,  tuning_block_length = 2, NBoot = 500,
method.block.length.select = "Lahiri", type="SE", lookuptables.folderpath=lookuptables.folderpathname)
```

Check sensitivity to tuning parameter. See that all columns are minimised by block length 0.1, so tuning block length wasn't terribly imporant here

```
Results.example4$Empirical.MSE
```

```
##                  tuninglength0 tuninglength0.05 tuninglength0.1
## blocklength0       0.001626589       0.02305761      0.07962379
## blocklength0.05    0.007378852       0.02219981      0.07119708
## blocklength0.1     0.029063621       0.04260590      0.09013901
## blocklength0.2     0.040642723       0.08243159      0.16230864
##                  tuninglength0.2
## blocklength0           0.1593789
## blocklength0.05        0.1441683
## blocklength0.1         0.1617980
## blocklength0.2         0.2629567
```

The block size parameter chosen is:

```
Results.example4$Lahiri_block_size
```

```
## [1] 0.05
```

This is the standard error (estimated via block bootstrap to account for spatial autocorrelation) of the AUC.

```
Results.example4$SE.estimate
```

```
## [1] 0.02576194
```

Example 5: Parallel Computing

Setting Up

First load the data and R packages you need into the R workspace.

```
load("PlayData_for_Examples.RData") ### load the data for the examples
source("LoadFunctions.R") #### Source the functions required to run the block bootstrap


#Install packages if required
if(!require(pROC)) { install.packages("pROC", repos = "http://cran.us.r-project.org"); require(pROC) }

library(pROC)
```

Motivation

As in Example 4, imagine we want to know the standard error of the AUC (area under the ROC curve) of our model. However we wish to run the bootstrap in parallel on a cluster computer. For NBoot = 500, we could run 10 batches, each with 50 bootstrap resamples. We then assemble the results using the function **CombineBatchesofBootstraps**.

To use **CombineBatchesofBootstraps**, each batch should contain an equal number of bootstrap resamples.

```
set.seed(42)

fit1 = glm (PresAbsresponse ~ temperature, data = dat, family="binomial")
```

Coding StatFunction argument

As in Example 4, define getAUC a function which will be the *StatFunction* argument to *BlockBootApply*.

```
GetAUC = function(dat){
fit1 = glm ("PresAbsresponse~temperature", data=dat, family="binomial")
AUC = auc(response=dat$PresAbsresponse, predictor= predict(fit1))
;
AUC
}
```

Run Block Bootstrap

First the user should create a folder called e.g. LookupTables in their R working directory. As all the batches of bootstraps use the same site coordinates they all share the same lookup tables.

```
lookuptables.folderpathname = "LookupTables/"
```

```
batchID = 1
assign (paste0("Results.example4.batch", batchID ),
        BlockBootApply (x = x ,y = y ,block_Ls = c(0,0.05,0.1,0.2), Grid_space = 0.01 ,
                        dat = dat, Stat.function = GetAUC, tuning_block_length = 2,
                        NBoot = 250, method.block.length.select = "Lahiri",type="SE",
                        lookuptables.folderpath=lookuptables.folderpathname))
```

```
batchID = 2
assign (paste0("Results.example4.batch", batchID ),
        BlockBootApply (x = x ,y = y ,block_Ls = c(0,0.05,0.1,0.2), Grid_space = 0.01 ,
                        dat = dat, Stat.function = GetAUC, tuning_block_length = 2,
                        NBoot = 250, method.block.length.select = "Lahiri",type="SE",
                        lookuptables.folderpath=lookuptables.folderpathname))
```

```
batchIDs=1:2

list_of_boot_batches = lapply(paste0("Results.example4.batch", batchIDs), get)

Results.example4.all = CombineBatchesofBootstraps(list_of_boot_batches=
list_of_boot_batches )
```

Check sensitivity to tuning parameter. See that all columns are minimised by block length 0.1, so tuning block length wasn't terribly imporant here

```
Results.example4.all$Empirical.MSE
```

```
##                   tuninglength0 tuninglength0.05 tuninglength0.1
## blocklength0       0.0003802906       0.02173633      0.08486327
## blocklength0.05    0.0093301256       0.02342151      0.07875828
## blocklength0.1     0.0266339887       0.03785125      0.09010597
## blocklength0.2     0.0382523887       0.08122859      0.16753968
##                   tuninglength0.2
## blocklength0            0.1325652
## blocklength0.05         0.1225259
## blocklength0.1          0.1323171
## blocklength0.2          0.2269502
```

The block size parameter chosen is:

```
Results.example4.all$Lahiri_block_size
```

```
## [1] 0
```

This is the standard error (estimated via block bootstrap to account for spatial autocorrelation) of AUC

```
Results.example4.all$SE.estimate
```

```
## [1] 0.0194651
```

Literature Cited

Lahiri, S. N. and Zhu, J. (2006). Resampling methods for spatial regression models under a class of stochastic designs. *Ann. Statist.*, **34** 1774–1813.