Statistics in Environmental Research (BUC Workshop Series) IV
Problem sheet 2

Website: `http://www.stat.ubc.ca/~gavin/STEPIBookNewStyle/course_bath.html`

Design of monitoring network

This lab covers, the design of monitoring networks as well as the analysis of spatio-temporal data theoretical exercises as well as data analysis. `EnviroStat`, an R package downloadable from the R-CRAN side is featured since it enables in a spatio-temporal setting:

- spatial mapping of random fields;
- the characterisation of the distribution of complex metrics, e.g. the maximum of a random spatial field;
- projection of domains geo-referenced by (latitude, longitude) onto flat surfaces with distances measured by kilometers;
- the elimination of non-stationarity in a random field using the SG method so that the correlation between the random responses between site-pairs becomes a monotone function of their intersite distance;
- the design of monitoring networks.

This illustration below shows how an hourly Ozone field with the staircase pattern of missing data commonly encountered in spatio-temporal data from environmental monitoring networks may be handled in the hierarchical framework described in [2]. It particular, it also shows how new stations could be optimally added to such networks.

In Chapter 14 of [3], a more extended version of the analysis is given and those enrolled in the Lab are invited to carry out that analysis after working through the example. It is assumed that you have installed from the course webpage, the DEMO files in your root directory EnviRo.stat.1.0.1. In that case the more extended analysis begins in Windows with the R commands:

```
data = matrix(scan("C:/EnviRo.stat.1.0.1/DEMO/data/data.ch14.txt"),byrow=T,ncol=14)
location = matrix(scan("C:/EnviRo.stat.1.0.1/DEMO/data/location.ch14.txt"),byrow=T,
       ncol=2)
```

But first lets work through the example.

```
#########################################################################
#########################################################################
# Then start R to begin with our example
# Dynamically linking the dll files and installing the R functions
#########################################################################

dyn.load("C:/EnviRo.stat.1.0.1/SG-method/SG.dll")
dyn.load("C:/EnviRo.stat.1.0.1/Design/design.1.0.dll")

# The following commands are not needed if the R functions have already been installed. Next

source("C:/EnviRo.stat.1.0.1/SG-method/SG.1.0.1.r")
source("C:/EnviRo.stat.1.0.1/Para.est/LZ-EM.est.1.0.1.r")
source("C:/EnviRo.stat.1.0.1/Design/LZ-design.1.0.1.r")
source("C:/EnviRo.stat.1.0.1/Pred.dist/LZ-predict.1.0.1.r")

# Read the data

data = matrix(scan("C:/EnviRo.stat.1.0.1/DEMO/data/data.NY.txt"),byrow=T,ncol=38)

# format: Month, Weekday, sqrt03 (36 columns: 9 stations x 4 hours - 8am-12noon)

# Extract the data - month, weekday, and sqO3
month = data[,1]
weekday = data[,2]
sqO3 = data[,3:38]


# Read information on locations (lat, long)
site.loc = matrix(scan("C:/EnviRo.stat.1.0.1/DEMO/data/location.NY.txt"),byrow=T, ncol=2)

#########################################################################
#########################################################################
# Fit the hierarchical model in the LZ method with covariates Z
#########################################################################
#########################################################################

# Rearrange data to have a monotone pattern
# Calculate number of missing observations
apply(is.na(sqO3),2,sum)
# Output
# [1]   0   0   0   0 109 109 109 109   0   0   0   0   0   0   0   0   3   3   3
# [20]   3  84  84  84  84   0   0   0   0   0   0   0   0   0   0   0   0   0

# Thus, the order of stations should be (2,6,5,1,3,4,7,8,9)
# Note that within each stations, the elements must have the same missing number
# of observations
```

```
norder = c(2,6,5,1,3,4,7,8,9)
tt = NULL
for (i in 1:9) tt = c(tt,c(1:4)+4*(norder[i]-1))
ndata = sqO3[,tt]
nloc = site.loc[norder,]

# Plot the stations
plot(nloc[,2],nloc[,1],xlab="Long", ylab="Lat", type="n")
text(nloc[,2],nloc[,1], c(1:9))


# Preliminary analysis (not included here) suggests 'month' and 'weekday'
# as important covariates and the sqrt transformation of O3 levels to
# be approximately nornal

# Set the design matrix for the covariates using the 'model.matrix' function

month[1:5]
weekday[1:5]

ZZ =model.matrix(~as.factor(month)+ as.factor(weekday))

ZZ[1:2,]

# EM staircase fit using the 'staircase' function - only missing data at the beginning
# of the series allowed but all elements within a station must have the same
# number of missing observations

#############################################################################################
#staircase.EM =function(data,p=1,block=NULL,covariate=NULL,B0=NULL,init=NULL,a=2,r=.5,verbose=F,
#    maxit=20,tol=1e-6) {
#
#########################
# Model:
#      data  ~   MVN ( z x beta , kronecker(I, Sigma) )
#      beta  ~   MVN (Beta0 , kronecker(Finv , Sigma )
#      Sigma ~   GIW (Theta , delta's )
#
#      Theta is a collection of hyperparamaters including Xi0, Omega, Lambda, Hinv
#
# Input:
#  data: data matrix, organized to have a staircase pattern of missing observations;
#   stations having the same number of missing observations are put in the same step
#       and the steps are organized in order of decreasing number of missing
#       observations, ie. step 1 has more missing observations than step 2.
#  Default structure:
#       Each column represent data from a station; rows are for time
#
```

```
# Optional input:
#  p: number of pollutants measured at each stations.
#  (first p columns of y are for p pollutants from station 1, block 1).
#
#  block: a vector sequentially indicating the number of stations to be grouped together.
# Ex: block=rep(1,g) where g is number of stations will have one station in each block.
# Note that stations in each block must be contained in ONLY ONE step.
#       That is, stations within each step could be separated into multiple blocks as an option.
# Default structure: The steps are used as blocks


#  covariate: design matrix for covariates created with "model.matrix" with "as.factor"
#  B0: Provided if the hyperparameter beta_0 (B0) is known and not estimated
#  init: Initial values for the hyperparameters; output of this function can be used for that
#  a,r : When p=1, the type-II MLE's for delta's are not available. Delta's are assumed to follow
#        a gamma distribution with parameters (a,r)
#  verbose: flag for writing out the results at each iteration
#  maxit: the default maximum number of iteration
#  tol: The convergence level.
#
# Output:
#  Delta:  The estimated degrees freedom for each of the blocks (list)
#  Omega:  The estimated covariance matrix between pollutants
#  Lambda: The estimated conditional covariance matrix between stations in each block given
#           data at stations in higher blocks (less missing data) - (list)
#  Xi0:    The estimated slopes of regression between stations in each blocks and those in higher
#            blocks (list). Note that tau_0i = kronecker(Xi0, diag(p)) - same across stations
#            for each pollutants.
#  Beta0:  Coefficients - assumed to be the same across stations for each pollutant
#  Finv:   Scale associated with beta_0
#  Hinv :  The estimated hyperparameters (list) - inverse of H_j
#  Psi:    The estimated (marginal) covariance matrix between stations
#  block:  From input
#  n.miss: Vector showing the number of missing observations in each of the blocks
#  data:   From input
#  covariate: From input
#
#  Lambda.1K: The inverse Bartlett decomposition
#
##############################################################################################

# Here the staircase steps are used for the block structure (default)
#
em.fit = staircase.EM(ndata,p=4,covariate= ZZ,maxit=200,tol=.000001, verbose=T)
# This takes a couple of minutes!

em.fit$block
# 4 blocks with the first block having 1 station and the last one 6 stations
```

```
em.fit$Omega
# between hours covariance


em.fit$Lambda
#  residual covariances

em.fit$Beta0[,1:4]
# Current version only allows exchangeable structure for each elements
# ie. the same b0 is assumed for each hour across all stations but different
#      b0's for different hours

# Get the marginal correlation matrices ('corr.est': spatial and 'omega': between hours)
cov.est = em.fit$Psi[[1]]
dim1= dim(cov.est)[1]
dim2= dim(em.fit$Omega)[1]
corr.est = cov.est / sqrt( matrix(diag(cov.est),dim1,dim1)*t(matrix(diag(cov.est),dim1,dim1)) )
#Spatial correlation
round(corr.est,2)

# Correlation between hours

omega=  em.fit$Omega/sqrt( matrix(diag(em.fit$Omega),dim2,dim2)*t(matrix(diag(em.fit$Omega),
      dim2,dim2)) )
round(omega,2)

# Plot correlation vs inter-distances
# Projecting the coordinates into a rectangular ones using Lambert projection
#                          ('Flamb2': a function in SG package)
#
##############################################################################
# Flamb2 <- function(geoconfig, latrf1 = NA, latrf2 = NA, latref = NA, lngref = NA)
# Evaluate Lambert projection for geoconfig: (lat, -long)
# latrf1, latrf2, latref, lngref: point of reference - to be computed if not provided.
##############################################################################

coords = Flamb2(abs(nloc))
coords

# Calculate distance between the locations using the 'Fdist' function - a function in SG package

#########################################################################
#Fdist <- function(crds)
# # Function to compute interpoint distances for nxp coordinate matrix.
#########################################################################
```

```
dist = Fdist(coords$xy)

# Plot the spatial correlation vs inter-distances
par(mfrow=c(1,1))
plot(-.2,0,xlim=c(0,250),ylim=c(-.2,1),xlab="Dist",ylab="Spatial correlation",type="n")
for (i in 1:8) for (j in (i+1):9) points(dist[i,j],corr.est[i,j])

# Plot the dispersion vs inter-distances

disp = 2-2*corr.est
plot(-.2,0,xlim=c(0,250),ylim=c(0,2),xlab="Dist",ylab="Dispersion",type="n")
for (i in 1:8) for (j in (i+1):9) points(dist[i,j],disp[i,j])

# Add a fitted exponential variogram using the 'Fvariogfit3' function (in SG package)
# with model=1 for Exponential (default) and = 2 for Gaussian
# Exponential variogram: a[1]+ (2-a[1])*(1-exp(-t0 *h))
# Gaussian variogram: a[1]+ (2-a[1])*(1-exp(-t0 *h^2))

h.lt = dist[row(dist) < col(dist)]
disp.lt = disp[row(disp) < col(disp)]
variogfit <- Fvariogfit3(disp.lt, h.lt, a0=1.5, t0=.1)
 x = seq(min(h.lt),max(h.lt),1)
 a0 = variogfit$a[1]
 t0 = variogfit$t0
 lines(x, a0+(2-a0)*(1-exp(- (t0* x))))
# Save it for later comparison!

#########################################################################
#########################################################################
#
# Use SG method to extend the spatial covariance to new locations
#
#
#########################################################################
#########################################################################
#
# Step 1: Identifying a thin-plate sline mapping transformation with no smoothing (lambda =0)
#         using the Falternate3 function - a SG function
#         Starting with the spatial correlation
#
#############################################################################################
#Falternate3 <- function(disp, coords, model = 1., a0 = 0.1, t0 = 0.5,
#     max.iter = 50., max.fcal= 100., alter.lim = 50., tol = 1e-05, prt = 0.,
#    dims = 2., lambda = 0., dev.mon = "postscript", ncoords)
#{
# Do simultaneous estimation of coords and exponential or gaussian
# variogram by alternating weighted least squares.
# This version permits dimension > 2 for scaling.
```

```
# In the plotting we'll use a plot symbol proportional to the
# third coordinate.
#
# This version also passes a smoothing parameter to the optimization.
# This parameter probably is not scaled exactly the same as it is
# in sinterp and this has not been investigated yet.
# Warning: make sure that coords are scaled reasonably small
# before attempting to compute; otherwise matrix inversion is
# likely not to work in calculation of bending energy matrix.
#
# Other arguments:
#       model:   1 (exponential), 2 (gaussian)
#       a0,t0:   initial variogram parameter estimates
#       max.iter, max.fcal:  control parameters for calls to nlmin
#                (same values used in MDS step and in variogram step)
#       dev.mon: device number for plot monitoring convergence of objective
#       ncoords: optional initial coordinates to use (2 dim) if not G-plane
#
# Note: Exponential variogram: a[1]+ (2-a[1])*(1-exp(-t0 *h))
#       Gaussian variogram: a[1]+ (2-a[1])*(1-exp(-t0 *h^2))

##############################################################################################

#  Smaller distances seem to work better with the multidimensional scaling method
X11()
coords.lamb  = coords$xy/10

#         The exponential variogram (default) is used
#         If Gaussian one is preferred, set model=2
sg.est = Falternate3(disp,coords.lamb,max.iter=100,alter.lim=100, model=1)

sg.est


# Step 2: Selecting a smoothing parameter for the identified spline mapping
#         through the 'Ftransdraw' function.

##############################################################################
#Ftransdraw <- function(disp, Gcrds, MDScrds, gridstr, sta.names, lambda = 0.,
#                       lsq = F, eye, model = 1., a0 = 0.1, t0 = 0.5)
#{
# Purpose: for varying (user-supplied) values of the spline smoothing
#          parameter lambda,
# - compute image of coordinates, interpoint distances, and dist-disp plot.
# - compute and draw image of regular grid (2D or 3D perspective plot)
       # Computed prior to execution of this code:
       # - disp: spatial dispersion matrix
       # - Gcrds: geographic coordinates (nx2)
```

```
       # - MDScrds:  kyst mds solution (nx2 or nx3) - using Falternate3
       # - gridstr: regular grid structure on the G-plane (from Fmgrid)
##############################################################################

#    First create a coordinate grid for examining the mapping transformation

apply(coords.lamb, 2, range)
coords.grid = Fmgrid(range(coords.lamb[,1]), range(coords.lamb[,2]))
par(mfrow=c(1,2))
temp = setplot(coords.lamb, ax=T)
deform  = Ftransdraw(disp=disp, Gcrds=coords.lamb,MDScrds=sg.est$ncoords,
                      gridstr=coords.grid)


# Note: In window, click on the window to register the curse before proceeding
#        and this interactive function will provide instructions - Particularly
#        when a new lambda value is given in R Console window.
#
# Lambda = 50 may be ok
# The fitted variogram is quite similar to the one with out any transformation (above)
# The mapping is almost linear!

# Step 3 - combining to get an optimal thin-plate spline using the
#          the 'sinterp' function & results stored in Tspline
Tspline = sinterp(coords.lamb, sg.est$ncoords, lam = 50 )



# Plotting the biorthogonal grid characterizing the deformation of the G-space

par(mfrow=c(1,1))
Tgrid  = bgrid(start=c(0,0), xmat=coords.lamb , coef=Tspline$sol)

tempplot = setplot(coords.lamb, ax=T)
text(coords.lamb,labels = c(1:(dim(coords.lamb)[1])))
draw(Tgrid, fs=T)
# Solid lines indicate contraction and dashed lines indicate expansion
# See Sampson+Guttorp (1992) for detail on interpretation


# Step 4- estimating the dispersion between new locations and the stations
#                         using the SG fit from steps 1-3 above
#         Here new locations are created using a grid of 100 points between the stations

lat10 = seq(min(nloc[,1]),max(nloc[,1]),length=10)
lat10

long10 = seq(max(abs(nloc[,2])),min(abs(nloc[,2])),length=10)
```

```
long10

llgrid =  cbind(rep(lat10,10),c(outer(rep(1,10),long10)))
llgrid[1:10,]

# the locations are ordered as (lat1,long1),(lat2,long1), ...,
#                     (latn,long1),(lat1,long1), ...

# Project the new locations using the same Lambert project for stations above,
# ie. using the same reference point.
# Note the same scale factor of 10 is used as before
z = coords
newcrds.lamb = Flamb2(llgrid,latrf1=z$latrf1, latrf2=z$latrf2, latref=z$latref,
      lngref=z$lngref)$xy/10

#Combine the new locations and stations together begining with new locations
allcrds = rbind(newcrds.lamb,coords.lamb)


# Using the 'corrfit' function to obtain correlations between the stations

##############################################################################################
# corrfit <- function(crds, Tspline, sg.fit, model = 1)
#
# This function estimates correlations between all the locations(new+stations)
# using the results of the SG step
#
#
#Input
#   crds : coordinates of all locations beginning with new locations
#   Tspline: the thin-spline fit from the SG-steps
#   sg.fit: the mapping resulted from the SG method
#   Model: variogram model; 1: exponential  2: gaussian
#Output
#   cor: correlation matrix among the locations
##############################################################################################

corr.est = corrfit(allcrds, Tspline = Tspline, sg.fit  = sg.est, model = 1)
round(corr.est$cor[1:5,1:5],2)

# Step 5: Interpolating the variance field
#
diag(cov.est)
# Non-homogeneous and interpolating using the same thin-plate spline
Tspline.var = sinterp(allcrds[101:109,],matrix(diag(cov.est),ncol=1),lam=50)

# The 'seval' function is used to obtain variance estimates at the locations
# Using the thin-plate spline and then arranged in to a matrix
```

```
varfit = seval(allcrds,Tspline.var)$y
temp = matrix(varfit,length(varfit),length(varfit))

# Combine to get the covariance matrix for all stations
covfit = corr.est$cor * sqrt(temp * t(temp))

# That completes the SG-method for extending the covariance matrix to ungauged sites
# stored in covfit
#########################################################################


#########################################################################
#########################################################################

# Extend the results to estimate hyperparameters associated with the new
# locations through the 'staircase.hyper.est' function

#########################################################################
#############################################################################
#staircase.hyper.est <- function(emfit,covfit,u,p,g,d0=NULL)
#
# This function combines the results from the "staircase.EM" fit and the
# SG method to estimate the hyperparameters associated with the ungauged sites
#
#Input
# emfit : Output from the staircase.EM fit
# covfit : The covariance matrix between all locations (with new locations
#          at the beginning). This is an output from the SG fitting
# u: number of new locations
# p: dimension of the multivariate response
# g: number of stations
# d0 (Optional): The degrees of freedom for the new locations (ungauged block)
#Output
#      Delta.0: The degree of freedoms for the new locations
#               = d0 if given (must be > u*p+2)
#               else
# = mean(emfit$delta) if > u*p+2
#           = u*p+ min(emfit$delta) otherwise
#      Lambda.0 : Conditional variance between new locations given the gauged stations
#      Xi0.0 : the regression slope (Note: Tau_0i = Kronecker(Xi0 , diag(p))
#      H.0 : The variance matrix for the rows of Tau^[u]
#  Also all components of the output of the staircase.EM fit (for blocks 1-K).
#
#############################################################################################

u = 100 # number of new locations
p = 4  # dimension of the multivariate response
hyper.est = staircase.hyper.est(emfit= em.fit,covfit=covfit,u =u, p=p)
```

```
#############################################################################
# This completes the estimation of all hyperparameters
#############################################################################
#
#
# The predictive distribution can be used for spatial interpolation
#
# Eg. Get the predictive mean and covariance for Day 183
#
x = hyper.est
tpt = 183
Z = x$covariate[tpt,]
y = x$data[tpt,]
# The beta0 for u= 100 ungauged locations each with p=4 hours
b0 = matrix(rep(c(x$Beta0[,1:p]),u),nrow=length(Z))

# Predictive mean for hours 8-12 on Day 183
mu.u = Z %*% b0 + (y- Z %*% x$Beta0) %*% kronecker(x$Xi0.0,diag(p))

X11()


# Plot the observed levels - Day 183
# Select a set of colors - color = colors() if default is preferred
color = colors()[c(12,26,32,37,53,60,70,80,84,88,94,101,116,142,
                                    366,371,376,386,392,398,400:657)]

# Plot the observed values for Day 183 by hours
par(mfrow=c(1,1))
plot(c(7,13),range(y), type="n", xlab="Hours",ylab="Levels (log)")
for (i in 1:p) points(rep(i+7,9),y[i+p*c(0:8)], col=color)
dev.off()

X11()
par(mfrow=c(2,2))
# Plot the contour for hours
for (i in 1:p) {
tt = i+ p*c(0:(u-1))
mu = mu.u[tt]
hr = matrix(mu,byrow=T, ncol=length(lat10))
print(range(hr))
contour(-long10,lat10, hr, xlab="Long", ylab="Lat",
     main=paste("Mean: Day 183 - Hour ", 7+i))
}

# Note that the predictive covariance may have to be obtained recursively
```

```
# (ie. when steps are involved). Generally the derivation is not
# simple in this case. A simpler approach is to simulate realizations from
# the predictive distribution and estimate the mean and covariance
# from the simulated data as demonstrated next.


#########################################################################
#########################################################################
# Interpolation by simulation using the predictive distribution -
# via the 'pred.dist.simul' function with N= 1000
#########################################################################
#########################################################################
#pred.dist.simul = function(hyperest, tpt, include.obs = T, N =1)
#
#
# This function simulates N- replicates from the predictive distribution for
# a given time point (tpt) from 1 to n (length of the data).
#
# Input
#   hyperest: Output from the "staircase.hyper.est" functions, containing
#             estimates of all hyperparameters
#   tpt:      A specific time point - from 1 to n corresponding to the
#             number of time points from the data set
#   include.obs:  If True, the observed data, for time "tpt", are also returned
#   N:        Number of replicates
#
# Output:
#   A matrix with N rows; the number of columns depends on whether the observed data are returned
#   The columns are organized consistent with the observed data
#         (ie. uxp ungauged blocks, g1xp, g2xp , ...)

# Note: This function could be slow if there are missing data at gauged sites
#       correspondind to the selected time point. That is, it is fastest at time points
#       corresponding to Block 1 and slower with higher blocks.

#########################################################################################

simu = pred.dist.simul(hyper.est,tpt = 183, N=1000)

# extract the simulated data at the gauged stations and
# plot the contours of the mean in a new graphic window

x = apply(simu,2,mean)[1:(p*u)]
X11()
par(mfrow=c(2,2))
# Plot the contour for hours
for (i in 1:p) {
tt = i+ p*c(0:(u-1))
```

```
x1 = x[tt]
hr = matrix(x1 ,byrow=T, ncol=length(lat10))
print(range(x1 ))
contour(-long10,lat10, hr, xlab="Long", ylab="Lat",
     main=paste("Mean: Day 183 - Hour ", 7+i))
}

# Plot the corresponding variance field
x = simu[,1:(u*p)]
X11()
par(mfrow=c(2,2))
# Plot the contour for hours
for (i in 1:p) {
tt = i+ p*c(0:(u-1))
x1 = x[,tt]
x2 = diag(var(x1))
vv = matrix(x2 ,byrow=T, ncol=length(lat10))
contour(-long10,lat10, vv, xlab="Long", ylab="Lat",
     main=paste("Var: Day 183 - Hour ", 7+i))
points(nloc[,2],nloc[,1])
}



##############################################################################
##############################################################################
# Design Solution
##############################################################################
##############################################################################

# The desing solution can be obtained with the 'ldet.eval' function

##############################################################################
# ldet.eval = function(covmat,k,all = FALSE)
# {
# This R function calculates the log |determinant| off all sub-covariance
# matrices of size (k x k) from a covariance matrix.
#
# Input:
#   covmat: a covariance matrix (ie. non-negative definite,
#           square and symmetric)
#   k: dimension of sub-covariance matrices considered
#   Optional: if True, returns all combinations with corresponding log|det|
# Note - This option may need additionally a large amount of memory and
#            so may not work for a large number of combinations!!
# Output:
#   coord.sel: The k coordinates having the largest log|det|
#   log.det  : The log|det| of the submatrix corresponding the coord.sel
```

```
#   all.comb : Null if all = False
#              all combinations and their log|det| if all = True
###############################################################################


# To select additional 3 stations among the 100 new locations from the above grid.
# Their conditional covariance is in 'hyper.est$Lambda.0'
# The  Entropy criterion selects the combination with the largest log|det|
#
# The 'ldet.eval' above evaluates the log|det| for sub-covariance matrices of size 'nsel'
# and returns the combination with largest value using option 'all =F'.
# Option 'all=T' returns all combinations with corresponding values. This
# option needs a very large memory allocation if the number of combinations is big
# and so should be used only for small number of potential sites, say < 15.
#
# Note that when the number of combinations is large, this function
# could be very slow even with option 'all=F'. For example, it could take about 30'
# to select 5 out of 100.

nsel = 3
yy = ldet.eval((hyper.est$Lambda.0+ t(hyper.est$Lambda.0))/2,nsel,all =F)


# Option all = T for a smaller matrix

yy1 = ldet.eval(((hyper.est$Lambda.0+ t(hyper.est$Lambda.0))/2)[1:10,1:10],nsel,all =T)
```

References

[1] Shaddick, G., Zidek, J. V., Spatio-temporal methods in environmental epidemiology. *Chapman and Hall/CRC Press*, London, 2015.

[2] Nhu, L. D., Zidek, J. V., Statistical analysis of environmental space-time processes. *Springer*, New York, 2006.

[3] EnviroStat, Statistical analysis of environmental space-time processes. The Comprehensive R Archive Network, 2015.