Ville Väänänen / 63527M
ville.vaananen@aalto.fi

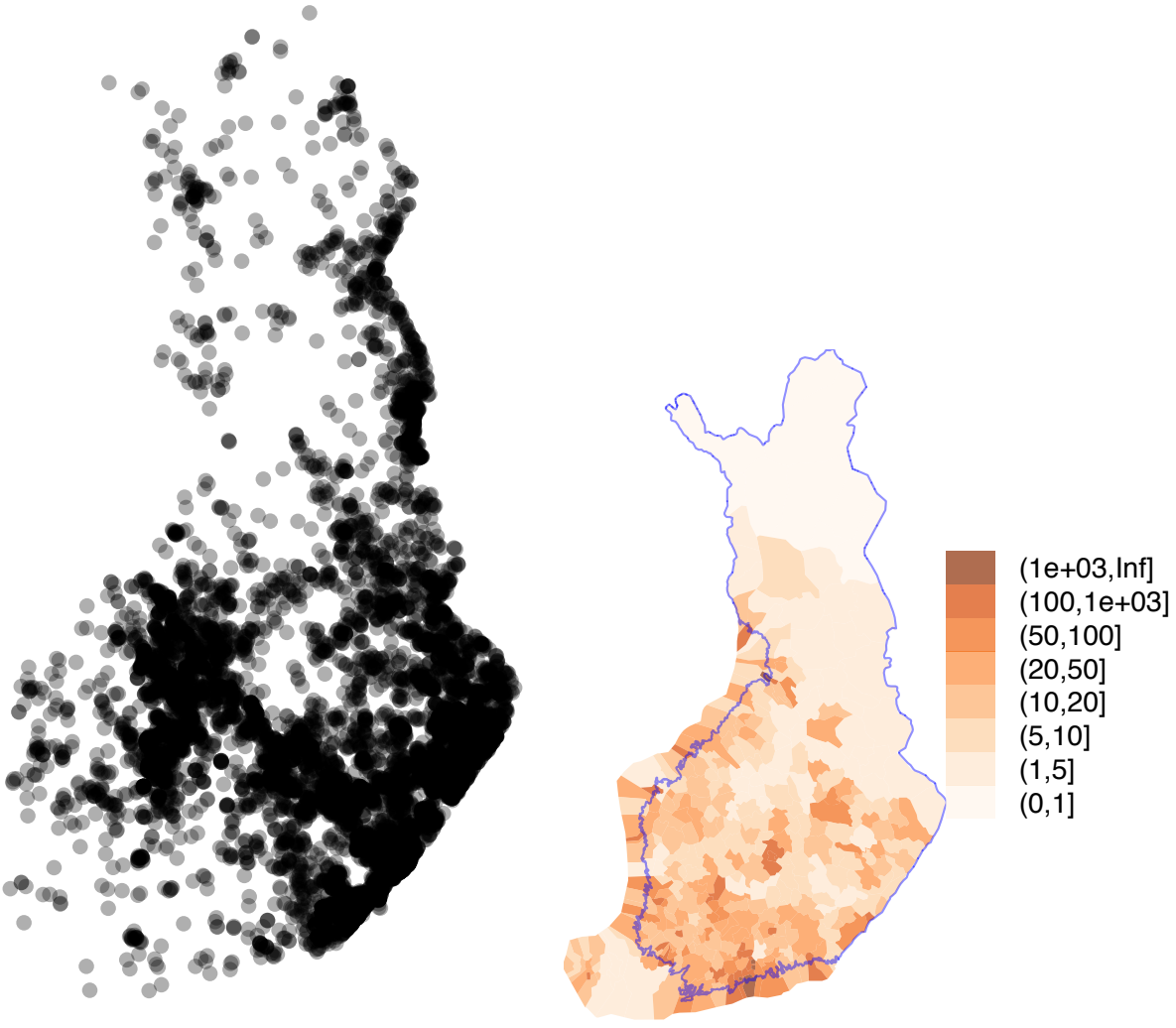
Bear observations in Finland in 2010

S-114.4202 Special Course in Computational Engineering II

March 26, 2012

1 Data description

The bear observations in the year 2010 are plotted in figure 1a and the population density of the municipalities (the number of people in the municipality per its land area) is plotted in figure 1b. As can be seen, there is some clustering which has to be associated with increased probability of reporting due population density. Thus an interesting problem is, how should we take into account the probability for an animal to get observed in a given area? Is it directly proportional to population density? Is there some other factors associated such as the amount of hunters in the area?



(a) Bear observations in 2010

(b) Population density

Figure 1: Bear observations in 2010 and the population density of Finland

2 Methods

Cox processes

Cox processes are a broad class of point processes that are nonhomogoneous Poisson processes given a *latent random intensity field* $\lambda(\mathbf{s})$. Throughout the report, we will denote vectors and vector valued functions with bold lower-case symbols and matrices with bold upper case symbols. We will denote a *spatial point process* defined in some bounded region $\Omega \subset \mathbb{R}^2$, also called the *observation window*, with $\mathcal{Y}(\mathbf{s})$ where $\mathbf{s} \in \Omega$. A realisation of a point process is called a *point pattern*

$$\mathbf{y} = \big[\mathcal{Y}(\mathbf{s}_1), \ldots, \mathcal{Y}(\mathbf{s}_n)\big]^T, \tag{1}$$

that can be thought of as an n dimensional random vector governed by a well defined multivariate distribution [1]. It is important to realise that the elements of $\mathbf{y}$ belong to some abstract sample space, i.e. bears or wolves in case of this report [1]. Also, (1) implicitly defines a matrix $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_n]$ of point locations.

Going back to Cox processes, given the intensity field, the number of points $N(D)$ inside a region $D \subset \Omega$ is a Poisson distributed random variable with mean $\Lambda(D) = \int_D \lambda(\mathbf{s}) \ \mathrm{d}\mathbf{s}$, i.e.

$$p\big(N(D) = k \mid \Lambda(D)\big) = \frac{\Lambda(D)^k}{k!} e^{-\Lambda(D)^k}. \tag{2}$$

In a log-Gaussian Cox process, the latent intensity field has the form

$$\lambda(\mathbf{s}) = \exp\big(Z(\mathbf{s})\big) \tag{3}$$

where Z is a *Gaussian random field* (also called a Gaussian process, GP) defined by a mean function $\mu(\mathbf{s})$ and a covariance function $Q(\mathbf{s}, \mathbf{s}')$ [2]. In this report a linear model will be used for the mean function, i.e

$$\big\langle \log \lambda(\mathbf{s}) \big\rangle = \boldsymbol{\beta}^T \mathbf{z}, \tag{4}$$

where typically $z_1(\mathbf{s}) = 1$, so that β_1 is the intercept. With this definition, possible covariate data can be included directly in $\mathbf{z}$.

Latent Gaussian models

Our aim is to be able to use a computationally efficient Bayesian procedure for inference in latent Gaussian models called INLA (or integrated nested Laplace approximations) [3]. Latent Gaussian models are a large set of flexible models, where given a realisation $\mathbf{x}$ of a latent (unobserved) Gaussian random field the observations $\mathbf{y}$ are independent and identically distributed. More precisely we

have

$$Z(\mathbf{s})|\boldsymbol{\theta} \sim \mathcal{GP}(\mu(\mathbf{s}, \boldsymbol{\theta}), Q(\mathbf{s}, \mathbf{s}', \boldsymbol{\theta}) \tag{5}$$

$$\mathbf{y}|Z(\mathbf{s}), \boldsymbol{\theta} \sim \prod_{k=1}^{n} p\big(y_k \mid Z(\mathbf{s}), \boldsymbol{\theta}\big) \tag{6}$$

Here we have introduced the hyperparameter $\boldsymbol{\theta}$. It is worth noting that in this vector notation $\mathbf{x}$ has to be thought of as belonging to an infinite dimensional Hilbert space.

Likelihood approximation

LGCP's can be thought of as latent Gaussian models and in their case the logarithm of the conditional likelihood function (6) can be written as [2, 1, 4] (going further, dependence on the hyperparameter will be suppressed but is assumed)

$$\log p(\mathbf{y} \mid Z) = |\Omega| - \int_{\Omega} e^{Z(\mathbf{s})} \, \mathrm{d}\mathbf{s} + \sum_{k=1}^{n} Z(\mathbf{s}_k) \tag{7}$$

Unfortunately (7) cannot be evaluated without first forming some kind of approximation to the latent random field $Z(\mathbf{s}) = \log \lambda(\mathbf{s})$. With the help of such an approximation, the last term requires just the evaluation of the approximated field in the observed locations. Since the integral of the intensity function over the observation window (i.e. the mean number of points in the pattern) will remain analytically intractable it will be computed numerically [4].

A common approach for approximating the conditional likelihood is to partition the observation window to a regular grid and assume the field constant inside a grid cell. The value chosen for a cell can c_k be for example the value of the field in the cell's center point $\mathbf{c}_k$. Clearly in this case the number of points in a grid cell is Poisson distributed with mean $|c_k| \, e^{Z(\mathbf{c}_k)}$. As argued in [4] it is far from optimal to use the grid both for approximating the intensity field *and* the locations of the points y_k. A nicer alternative would be to form a finite dimensional *continous* approximation to the field, so that it could be evaluated at the exact point locations.

This can be achieved by forming a piecewise linear basis function approximation to $Z(\mathbf{s})$

$$\hat{Z}(\mathbf{s}) = \sum_{j=1}^{m} z_j \phi_j(\mathbf{s}) \tag{8}$$

$$= \mathbf{z}^T \boldsymbol{\phi}(\mathbf{s}) \tag{9}$$

The details of this approximation are laid out in [5] and only the results will be stated here. First of all, computational efficiency leads us to only consider *Markovian* Gaussian random fields, and it turns out that a subset of the Gaussian random fields with the *Matérn* covariance functions has the spatial Markov

property. The Matérn covariance functions are functions of only the distance $r = \|\mathbf{s} - \mathbf{s}'\|$ between two points

$$c_M(r) = \frac{\sigma^2}{2^{\nu-1}\Gamma\nu}(\kappa\, r)^\nu K_\nu(\kappa\, r), \tag{10}$$

where $K_\nu(\cdot)$ is the modified Bessel function of the second kind, $\nu > 0$ is the smoothing parameter, $\kappa > 0$ is the range parameter and σ^2 is the variance. The Matérn fields have the Markov property when $\alpha = \nu + \dfrac{d}{2}$ is an integer. In our case we have $d = 2$ and we will choose $\nu = 3$, so that $\alpha = 2$. The piecewise linear basis functions will be defined on a triangulated mesh of the observation window, so that the value of $\phi_i(\mathbf{s})$ is 1 on vertex i and goes linearly to zero on the neighboring vertices.

The integral in (7) is approximated with a sum using a simple deterministic numerical integration scheme with weights w_i and sigma points $\hat{\mathbf{s}}_i$, so that the approximation to the integral of some function f over the domain Ω is given by

$$\int_\Omega f(\mathbf{s})\ \mathrm{d}\mathbf{s} \approx \sum_{i=1}^{p} w_i\ f(\hat{\mathbf{s}}_i) \tag{11}$$

Examples of such numerical integration rules are the well known midpoint rule, trapezoid rule and and the Simpson's rule (f is interpolated with a polynomial of order 0, 1 or 2 respectfully).

Putting all this together, the approximation to the conditional log-likelihood is given as

$$\log p(\mathbf{y} \mid Z) \approx C - \sum_{i=1}^{p} w_i \exp\left(\mathbf{z}^T \boldsymbol{\phi}(\hat{\mathbf{s}}_i)\right) + \sum_{k=1}^{n} \mathbf{z}^T \boldsymbol{\phi}(\mathbf{s}_k) \tag{12}$$

$$= C - \mathbf{w}^T \exp\left(\mathbf{A}_1\mathbf{z}\right) + \mathbf{1}^T\mathbf{A}_2\mathbf{z}, \tag{13}$$

where $[\mathbf{A}_1]_{ij} = \phi_j(\hat{\mathbf{s}}_i)$, $[\mathbf{A}_2]_{kj} = \phi_j(\mathbf{s}_k)$ and C is an unimportant constant.

3 Results

4 Conclusion

References

[1] A Gelfand, PJ Diggle, and P Guttorp. *Handbook of Spatial Statistics*. CRC Press, 2010. ISBN: 9781420072877. URL: http://eprints.ncrm.ac.uk/2066/.

[2] Jesper Møller and Rasmus P. Waagepetersen. "Modern Statistics for Spatial Point Processes". In: *Scandinavian Journal of Statistics* 34.June 2006 (Sept. 2007), 070927154002001–??? ISSN: 0303-6898. DOI: 10.1111/j.1467-9469.2007.00569.x. URL: http://doi.wiley.com/10.1111/j.1467-9469.2007.00569.x.

[3] Håvard Rue, Sara Martino, and Nicolas Chopin. "Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71.2 (Apr. 2009), pp. 319–392. ISSN: 13697412. DOI: 10.1111/j.1467-9868.2008.00700.x. URL: http://doi.wiley.com/10.1111/j.1467-9868.2008.00700.x.

[4] Daniel Simpson, Janine Illian, and Finn Lindgren. "Going off grid: Computationally efficient inference for log-Gaussian Cox processes". In: *Science And Technology* (2011). URL: http://adsabs.harvard.edu/abs/2011arXiv1111.0641S.

[5] Daniel Simpson and Finn Lindgren. "Think continuous: Markovian Gaussian models in spatial statistics". In: *Arxiv preprint arXiv:1110.6796* (2011). URL: http://arxiv.org/abs/1110.6796.

1 R-code

```
exportFigs <- 1
displayFigs <- 0

plots.obs <- TRUE
plots.mesh <- TRUE
plots.results <- FALSE

run.mesh <- TRUE
run.INLA <- FALSE

source("util.R",local=TRUE)
if(1 | !exists("kunnat")) {
  source("loadData.R",local=TRUE)
  source("dist.R",local=TRUE)
  ocoords <- coords
}
bb <- map@bbox
#corners <- matrix(c(bb[1,1],bb[2,1], bb[1,2],bb[2,1], bb[1,2],bb[2,2], bb
    [1,1],bb[2,2] ),4,2,byrow=TRUE)
#ocoords <- corners[dim(corners)[1]:1,]
#coords <- ocoords
lx <- max(ocoords[,1])-min(ocoords[,1])
ly <- max(ocoords[,2])-min(ocoords[,2])

offX <- mean(ocoords[,1])
offY <- mean(ocoords[,2])

# coords[,1] <- (ocoords[,1]-offX)/ly
# coords[,2] <- (ocoords[,2]-offY)/ly

if(run.mesh) {

  N <- nrow(coords)
  loc.bnd <- coords[N:1,]
  segm.bnd <- inla.mesh.segment(as.matrix(loc.bnd))

  maxedge <- ly/50
  minedge <- maxedge
  mesh <- inla.mesh.create(
                    ## Data locations:
                   bound=segm.bnd,
                    ## Where to put the mesh/graph files:
                    ## Set to >=0 for visual (not on Windows):
                    plot.delay=NULL,
                    keep=FALSE,
                    cutoff=minedge,
                    refine=(list(min.angle=26,
                                 max.edge.data=maxedge,
                                 max.edge.extra=2*maxedge))
          )
```

```r
  spde <- inla.spde.create(mesh, model="matern")
}

############################
#### OBSERVATIONS ########
############################

loc <- as.matrix(obs.y2010)
loc <- loc[!duplicated(loc[,c("x","y")]),]
loc <- loc[sample.int(min(2000,nrow(loc))),]

##############################
#### COVARIATES ##############
##############################
distmat <- crossdist(mesh$loc[,1],mesh$loc[,1],cov.pdens[,"x"],cov.pdens[,"y"])
cov.pdens <- cov.pdens[apply(distmat,1,which.min),"pdens"]

#loc <- loc[loc[,1] < 0.17,]

# kappa =0.5
# tau =0.5

############################
#### RESCALE ###############
############################

# loc[,1] <- (loc[,1]-offX)/ly
# loc[,2] <- (loc[,2]-offY)/ly

# ##image.plot(inla.mesh.project(proj, variances^0.5), col=cp(256))

# sample = inla.spde.query(spde, sample=list(tau=tau, kappa2=kappa^2))$sample

# max_points = 10000
# intercept =log(max_points) - max(sample)
# source("Daniel/simulate_lgcp.R",local=TRUE)
# loc <- simulate_lgcp(mesh,intercept+sample)

# print(dim(loc)[1])
# # discard points that fall outside the window
loc <- pip(loc,coords,bound=FALSE)

plot(mesh)
points(loc,col="#ff000080",pch=16)
nV <- mesh$n
nData <- dim(loc)[1]
print(nData)
print(nV)
# print(dim(loc)[1])

if(run.INLA) {
  ## Create the SPDE/GMRF model, (kappa^2-Delta)(tau x) = W:
  #Convenient definitions - number of vertices and number of observations

  #See paper - we need two A matrices, one for the location of the points and
```

```
    one
  #for the integration points (here taken to be the identity matrix)
  LocationMatrix = inla.mesh.project(mesh, loc)$A
  IntegrationMatrix = sparseMatrix(i=1:nV,j=1:nV,x=rep(1,nV)) #simple
      integration scheeme
  ObservationMatrix=rBind(IntegrationMatrix,LocationMatrix) #Glue matrices
      together

  #The integration weights (alpha in the paper)
  IntegrationWeights = diag(spde$internal$c0)
  E_point_process =c(IntegrationWeights,rep(0,nData)) #Glue everything together

  fake_data <- c(rep(0,nV),rep(1,nData)) #Put a zero where there aren't
      observations and a 1 where there is a point

  formula <- y ~ 1 + pdens + f(idx, model=spde) #Basic latent model - feel free
       to add covariates etc

  data <- list(y=fake_data,idx = c(1:nV),pdens=cov.pdens) #put hte data in

  #The INLA call.  Likelihood is Poisson with Observation Matrix and
      appropriate value fo E.
  #result = inla(formula, data=data, family="poisson",
  # control.predictor=list(A=ObservationMatrix),E=E_point_process,verbose=TRUE)

  sigma2.approx = 0.5^2
  range.approx = 0.4
  kappa = sqrt(8)/range.approx
  tau = 1/(4*pi*kappa^2*sigma2.approx)^0.5

  #init.mode = c(-5.5, 3) # works
  #init.mode = c(-8, 5.5) # works
  #init.mode = c(-4.905784,6.317846)
  #init.mode = c(log(tau), log(kappa^2))
  #init.mode = c(-1.304829,2.717723)

  #control.mode = list(theta = init.mode, restart=TRUE)

#The INLA call.  Likelihood is Poisson with Observation Matrix and appropriate
    value fo E.
result =
    inla(formula, data=data, family="poisson",
         control.predictor=list(A=ObservationMatrix,compute=TRUE),
         E=E_point_process,
         #control.mode = list(theta = init.mode, restart=TRUE),
         verbose=TRUE,
          ## We don't need the marginals:
          control.compute = list(return.marginals=FALSE),
          #control.mode = control.mode,
          ## We don't need to overoptimise:
          control.inla=list(tolerance=1e-4)
         )

}
```

```
############################
### PLOTS ##################
############################

if(displayFigs | exportFigs) {

  ms <-summary(mesh)

  wh <- (ms$xlim[2]-ms$xlim[1])/(ms$ylim[2]-ms$ylim[1])

if(plots.results) {
  ########################
  #Some fancy plotting code stolen from Finn
  ############################
  ## Prepare for plotting:

  ## Calculate mapping between triangulation vertices and grid points:

  proj = inla.mesh.projector(mesh, dims=c(200,200))

  ## Construct greyscale palette function:
  my.grey.palette = function (n,...) { return (grey.colors(n,0.05,0.95,...))}
  ## Use it:
  my.palette = my.grey.palette

    cp = colorRampPalette(c("darkblue", "blue", "cyan", "yellow", "red", "
        darkred"))
    my.palette = cp

  ## Construct map data appropriate for easy plotting:
  #mm = calc.map(map)

  levelplotmap = function(..., mm) {
      panel.levelplot(...)
      panel.lines(mm$x, mm$y, col="red")
  }

  ####################
  ## Plot results:

  ## Map resulting posterior mean field to a grid:
  plotdata = inla.mesh.project(proj, result$summary.random$idx[,"mean"])
  ## Plot PM contours:
  if(exportFigs) {
    pdf(file="report/postint_b2010.pdf",width=5,height=5/wh)
    par(mar=mar_tight)
  } else {

    dev.new()
  }
  bbb = (levelplot(row.values=proj$x, column.values=proj$y, x=plotdata,
                   mm=NULL, panel=levelplotmap,
                   col.regions=my.palette,
                   xlim=range(proj$x), ylim=range(proj$y), aspect="iso",
                   contour=TRUE, cuts=11, labels=FALSE, pretty=TRUE,
```

```
                  xlab="Easting",ylab="Northing"))
  print(bbb)
  if(exportFigs) {
    dev.off()
  }

  # pp<-myplot(plotdata,
  #   file="report/postint_b2010.pdf",
  #   plotfn=levelplot,
  #   width=5,
  #   height=5/wh,
  #   afterfn=function(p,k) {
  #     print(p)
  #   },
  #   mar=mar_tight,
  #   main="",
  #   col="#00000020",
  #   row.values=proj$x,
  #   column.values=proj$y,
  #     mm=NULL,
  #     panel=levelplotmap,
  #     col.regions=my.palette,
  #     xlim=range(proj$x), ylim=range(proj$y), aspect="iso",
  #     contour=TRUE, cuts=11, labels=FALSE, pretty=TRUE,
  #     lab="Easting",ylab="Northing"
  # )

  #Plot the interecept and the hyperparameter posteriors
  #plot(result, plot.random.effects=FALSE)

  #Plot the posterior for the RANGE of the random field
  range=inla.tmarginal(function(x) sqrt(8)/exp(x/2),(result$marginals.hyperpar$
      "K.0 for idx-basisK" ))
  attr(range,"inla.tag")="Range"
  #plot(range)

}

if(plots.mesh) {

  ms <-summary(mesh)

  wh <- (ms$xlim[2]-ms$xlim[1])/(ms$ylim[2]-ms$ylim[1])
  myplot(mesh,
    file="report/mesh.pdf",
    width=5,
    height=5/wh,
    afterfn=function(p,k) {
      lines(map,col="#0000ffff")
      print(p)
    },
    mar=mar_tight,
    main="",
    col="#00000020"
  )
```

```
}

if(plots.obs) {

  cut.val <- 0 ### Just to force it.
  theme.novpadding <-
      list(layout.heights =
           list(top.padding = cut.val,
                main.key.padding = cut.val,
                key.axis.padding = cut.val,
                axis.xlab.padding = cut.val,
                xlab.key.padding = cut.val,
                key.sub.padding = cut.val,
                bottom.padding = cut.val),
           layout.widths =
           list(left.padding = cut.val,
                key.ylab.padding = cut.val,
                ylab.axis.padding = cut.val,
                axis.key.padding = cut.val,
                right.padding = cut.val))

  oranges <- sapply(brewer.pal(8,"Oranges"),function(c) {addAlpha(c,0.7)})

  rajj <- list("sp.lines", map, col = "#0000ff70")
  pt <- list(
      "b2010"=list(
        "sp.points",
        obs.y2010,
        pch = 16,
        col = "#00000088",
        cex=0.5),
      "b2009"=list("sp.points",
        obs.y2009[obs.y2009$laji=="karhu",c("x","y")],
        pch = 16,
        col = "#00000088",
        cex=0.5),
      "l2009"=list("sp.points",
        obs.y2009[obs.y2009$laji=="ilves",c("x","y")],
        pch = 16,
        col = "#00000088",
        cex=0.5),
      "w2009"=list("sp.points",
          obs.y2009[obs.y2009$laji=="susi",c("x","y")],
          pch = 16,
          col = "#00000088",
          cex=0.5)
      )

  # plot the points

pnel <- function() {
    #panel.levelplot(...)
    panel.lines(map,col="#0000ffff")
}
```

```
  myplot(y~x,
    obs.y2010,
    file="report/b2010.pdf",
    plotfn=xyplot,
    #panel=panel.lines(map,col="#0000ffff"),
    width=5,
    #height=5/wh,
    afterfn=function(p,k) {
          trellis.par.set(axis.line=list(col=NA))
          print(p)
    },
    mar=mar_tight,
    main="",
    col="#00000050",
    pch=16,
    aspect="iso",
    settings=theme.novpadding
  )

  # plot the population density
  pl_b10 <- myplot(kunnat,
      sp.layout=list(rajj),
        plotfn=spplot,
        file="report/pdens.pdf",
        width=3,
        height=5.5,
        afterfn=function(p,k) {
          trellis.par.set(axis.line=list(col=NA))
          print(p)
        },
        par.settings=theme.novpadding,
        mar=mar_tight,
        main="",
        zcol="pdens_binned",
        col.regions=oranges,
        col="transparent",
        colorkey=TRUE,
        pretty=TRUE
      )
  }
}
```

```
kunnat <- readShapeSpatial("map45/HallintoAlue45Milj_ETRS.shp")
#kraj <- readShapeSpatial("map45/HallintoalueRaja45Milj_ETRS.shp",verbose=TRUE)
#kraj <- kraj[kraj@data$Kohdeluokk==82100|kraj@data$Kohdeluokk==84111,]
city <- readShapeSpatial("map/cityp.shp")
# turn the factor to numeric vector
kunnat@data$Kunta <- as.numeric(as.vector(kunnat@data$Kunta))
hcont <- readShapeSpatial("map45/hcont_l.shp")
map <- hcont[row.names(hcont@data)=="761"|hcont@data$Z==9999,]
#forest <- readShapeSpatial("map/forest.shp")
population <- read.csv(
  "asukastiheys.csv",
```

```
  sep=",",
  header=FALSE,
  skip=0,
  col.names=c("Kunta",
        "land_area",
        "water_area",
        "water_area2",
        "total_area",
        "population",
        "pdens_area",
        "pdens_land_area"))

pdens <- sapply(kunnat@data$Kunta, function(k) {
  if(sum(population$Kunta==k) > 0) {
    tmp <- population[population$Kunta==k,c("pdens_land_area")]
    return(tmp[1])
  }
  print(k)
  return(NA)
})

kunnat_cnt <- coordinates(kunnat)

cov.pdens <- as.data.frame(cbind(
        kunta=kunnat@data$Kunta,
        x=kunnat_cnt[,1],
        y=kunnat_cnt[,2],
        pdens=pdens))

pdensbins <- c(0,1,5,10,20,50,100,1000,Inf)
kunnat@data <- cbind(kunnat@data,pdens=pdens,pdens_binned=cut(pdens,pdensbins))

obs.y2010 <-read.csv("BearObs2010XY_ETRS.csv",
  header=FALSE,
  sep=" ",
  skip=0,
  col.names=c("x","y"))
obs.y2009 <- read.csv(
  "Obs2009XYS_ETRS.csv",
  sep=",",
  header=FALSE,
  skip=0,
  col.names=c("x","y","laji"))
```

```
sp1 <- sapply(coordinates(map), function(x) do.call("rbind", x))
N <- length(sp1)
#sp1 <- sp1[c(1:14,17:N)]
N <- length(sp1)
lps <- matrix(nrow=N,ncol=2)
fps <- matrix(nrow=N,ncol=2)
NNN <- 0
for(i in 1:N) {
  line <- sp1[[i]]
  NN <- nrow(line)
```

```
  NNN <- NNN+NN
  lps[i,] <- line[NN,]
  fps[i,] <- line[1,]

  # for(j in 1:N) {
  #   mi <- colMeans(sp1[[i]])
  #   mj <- colMeans(sp1[[j]])
  #   dist[i,j] = sqrt((mi[1]-mj[1])^2+(mi[1]-mj[1])^2)
  # }
}
#d <- as.matrix(dist(mns))
#d[d==0]=NA
c <- vector("integer",N)
c[1] <- 1
coords <- sp1[[c[1]]][nrow(sp1[[c[1]]]):1,]

for(i in 1:(N-1)) {
  dmin <- 99999999
  jmin <- 0
  fp <- fps[c[i],]
  for(j in 1:N) {
    lp <- lps[j,]
    d <- dist(rbind(fp,lp))[1]

    if(!is.element(j,c) & d < dmin) {
      jmin <- j
      dmin <- d
    }
  }
  if(jmin == 16)
    break
  # print(c[i])
  # print(jmin)
  # print(dmin)
  c[i+1]<-jmin
  nline <- sp1[[jmin]]
  coords <- rbind(coords,nline[nrow(nline):1,])
  #coords <- rbind(coords,nline)
}
coords <- rbind(coords,getAllCoords(map,c(16,3,4,5)))

# coords <- matrix(nrow=0,ncol=2)
# #c <- c(1,unique(c))
# for(i in 1:3) {
#   line <- sp1[c(i)]
#   NN <- nrow(line)
#   coords <- rbind(coords,line[NN:1,])
# }

# coords2 <- matrix(nrow=0,ncol=2)
# for(i in 1:N) {
#   coords2 <- rbind(coords2,fps[i,])
#   coords2 <- rbind(coords2,lps[c(i),])
# }
```