Modelling Line Transect survey data as a filtered point process in INLA

1 Introduction

In this paper, we would like to show how to use INLA package to analyse line transect survey data, in particular the density surfance depends on at least one environmental covrariate and the detection function is assumed to be half normal and only depends on the distance perpendicular to the transect line. We use a suitbale dataset to illustrate the method and provides details about how the model is implemented in INLA.

In general, we need three datasets to construct a filtered point process model for the line trasnect survey data: 1) the obervation data which include all the information for each sighting; 2) the effort data which specify the location for each transect line and 3) the covariate data which provides the covariates for the whole survey area. Unlike other methods (add ref here), we need the covraties not just on the sample area, but for the whole survey area. The filtered point process is defined on the whole survey area.

Add ref here to introduce the idea of using point process model to analyse line transect data, what has been done here and what is new in this paper.

2 blue whales: data exploring and preliminary analysis

```
sightorig = read.csv(file = paste(datadir, "Bmus86_06.csv", sep = ""),
    head = TRUE, sep = ",")
head(sightorig)
```

```
  Sp1 Sp2 Sp3 Sp4 Cruz Sght E Obs Year Month Day Time    Lat
1  75  NA  NA  NA 1267    1 1   7 1989    11  11  639 10.619
2  75  NA  NA  NA 1081    1 1  51 1987     8  10  816 26.513
3  75  NA  NA  NA 1268    1 1  55 1989     9  24  826 -1.256
4  75  NA  NA  NA 1267    1 0   4 1989    10  22 1320 10.221
5  75  NA  NA  NA 1614    1 1 198 1999     7  28 1622 31.731
6  75  NA  NA  NA 1370    1 1  55 1990    12   2  658 21.641
     Long   GS TotGS   PD Bf SH  SD RF HS VS WSp WDi Cue Me Ph Bi
1  -98.95 1.16  1.16 1.37  2 -1  -1  1  1  3  -1  -1   6  4 -1  0
2 -116.18 2.32  2.32 0.78  2 -1  -1  1 -1 -1  -1  -1   6  4 -1  0
3  -91.31 2.32  2.32 1.59  4 -1  -1  2 -1 -1  -1  -1   6  5 -1  0
4  -98.36 1.16  1.16 0.56  5 -1  -1  2  7  1  -1  -1   6  6 -1  0
5 -116.94 2.32  2.32 0.93  3  4 302  1  3  2  13 323   6  4  0  0
6 -115.47 2.76  2.76 0.16  5 -1  -1  1  4  3  -1  -1   6  5 -1  0
  Mx Crs  SST  Vis Angl Retcl RadD InitID MSp
1  0  65 27.2 -1.9   21   1.0 3.84     -1  -1
2  0 154 21.7 -1.9   20   2.4 2.27     -1  -1
3  0 120 19.4 -1.9   35  -1.0 2.78     -1  -1
4  0 323 26.1 -1.9  330  -1.0 1.11     -1  -1
5  0 156 -1.0 11.1  346   1.0 3.83     -1  -1
6  0 343 -1.0 -1.9  350  -1.0 0.93     -1  -1
```

```
nrow(sightorig)
```

```
[1] 266
```

```
## get rid of the non-LT-mode sightings
sightdat = sightorig[sightorig$E == 1, ]
nrow(sightdat)  ## no. of sightings at the LT mode
```

```
[1] 195
```

```
nrow(sightdat.allmode[sightdat.allmode$E == 0, ])
```

```
Error: object 'sightdat.allmode' not found
```

```
## Tim commented that it is not LT mode if Bft>5
sightdat = sightdat[sightdat$Bf ≤ 5, ]
unique(sightdat$Bf)
```

```
[1] 2 4 3 5 1 0
```

```
## after excluding all sightings that are not LT mode
nrow(sightdat)
```

```
[1] 194
```

```
## summary of data
unique(sightdat$E)
```

```
[1] 1
```

```
unique(sightdat$Year)
```

```
 [1] 1989 1987 1999 1990 1988 1986 2006 1992 2000 1998 2003 1993
```

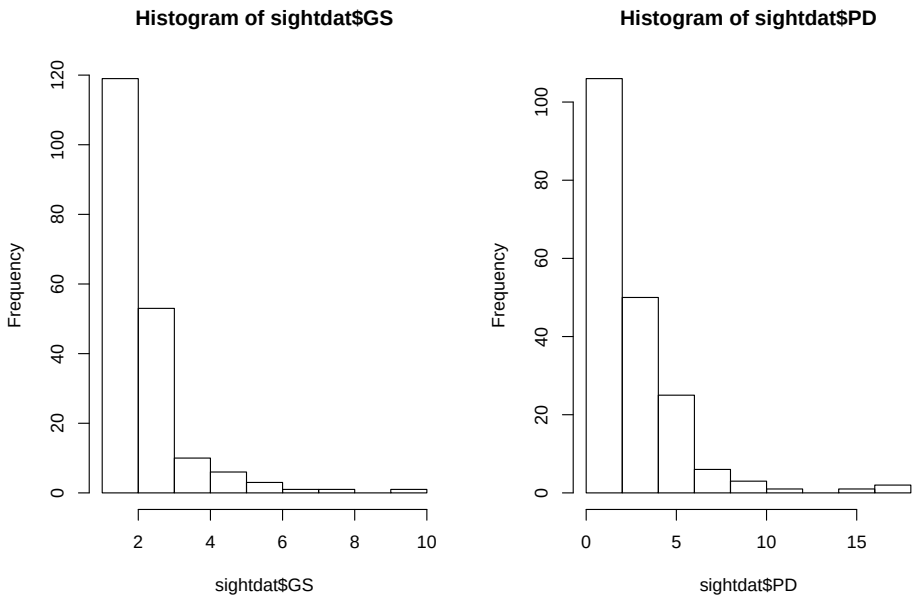
```
range(sightdat$PD)
```

```
[1]  0.00 16.67
```

```
range(sightdat$GS)
```

```
[1] 1.15 9.60
```

```
par(mfrow = c(1, 2))
hist(sightdat$GS)  #, main ='histograme of group size')
hist(sightdat$PD)  #, main ='histograme of PD')
```



2.1 Problem with the effort data

```
# ################ ### effort data for line transect locations
# efforig = read.table(file=paste(datadir, 'Eff86_06.csv', sep=''),
# header=T,sep=',' ,colClasses='numeric') head(efforig) ## get rid
# of the faulty entries in the effort datasets efforig =
# efforig[-which(efforig$lon> -75),] # ## understand how
# duplicate() works # tt = matrix(data=c(1:12), nrow =6, ncol =2) #
# tt[2,]=tt[1,] # tt[6,]=tt[1,] # tt # anyDuplicated(tt, MARGIN=1)
# # duplicated(tt, MARGIN=1) ## it should be duplicate the entire
# row not just lon or lat
# ######################################## ## duplicate function
# won't work for my case due to the complcated NAs in the data
# allID.NAeff = which(is.na(efforig$year), arr.ind=T)
# allID.NAeff=allID.NAeff[-1] head(allID.NAeff)
# duplicatedID.withNAs = NULL for(idx.LTsegment in
# c(2:length(allID.NAeff))) { IDstart =
# allID.NAeff[idx.LTsegment-1]+1 IDend =
```

```
# allID.NAeff[idx.LTsegment]-1 p0lonlat = c(efforig[IDstart,
# c('lon', 'lat')]) p1lonlat = c(efforig[IDend, c('lon', 'lat')])
# L=rdist.earth(x1=t(as.numeric(p0lonlat)),
# x2=t(as.numeric(p1lonlat)), miles = FALSE)
# if(L<1e-3){duplicatedID.withNAs =
# c(duplicatedID.withNAs,c((IDstart-1):IDend))} }
# length(duplicatedID.withNAs)## 975 duplicated rows effNoDuplicate
# = efforig[-duplicatedID.withNAs,] dim(effNoDuplicate)
# effNoDuplicate[1:50,] save(effNoDuplicate, file=paste(datadir,
# 'effNoDuplicate.Rdata',sep=''))
```

The above code gets rid of the duplicated rows in the effort data, and there is some complilcation in this process due to the NAs which are used in the original dataset to note the start and end of transect segments. Because of these NAs, I cannot use the function dumplicate(), and a loop has to be used.

There are further errors in the effort data, and the related datails are given in an email to Tim. I cannot deal with all the issues as there are so many. So I will just get rid of the transect segments that are longer than 200km.

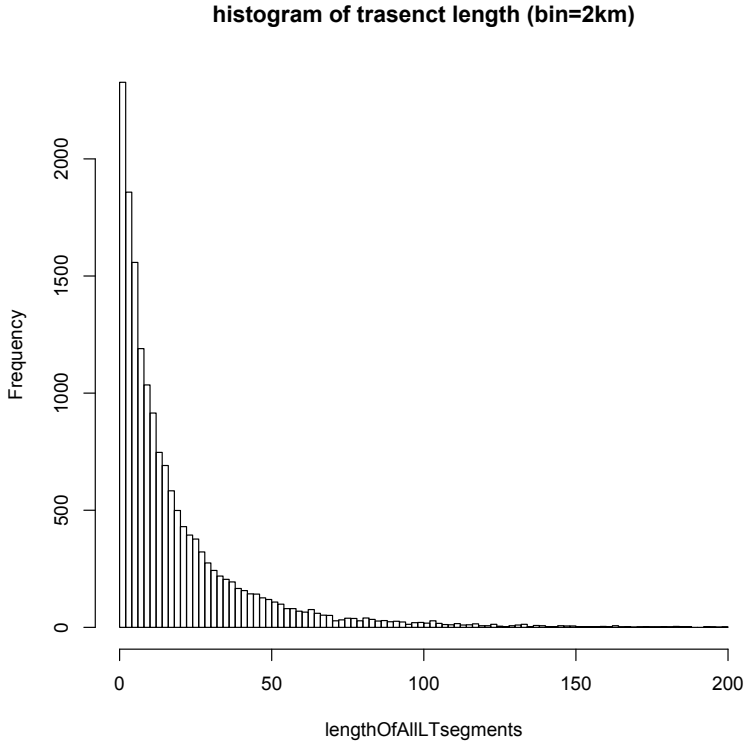


Figure 1: The histogram of the length of transect segments that are less than 200 km.

```
# efforig = read.table(file=paste(datadir, 'Eff86_06.csv', sep=''),
# header=T,sep=',' ,colClasses='numeric')
load(file = paste(datadir, "effNoDuplicate.Rdata", sep = ""))
eff = effNoDuplicate
############################################ load Bnd data The STAR boundary (black lines) apply to data from
############################################ 1998-2006.
boundSTART = read.table(paste(datadir, "BoundSTAR.dat", sep = ""),
    header = F, skip = 1)
colnames(boundSTART) = c("lat", "lon")
head(boundSTART)
```

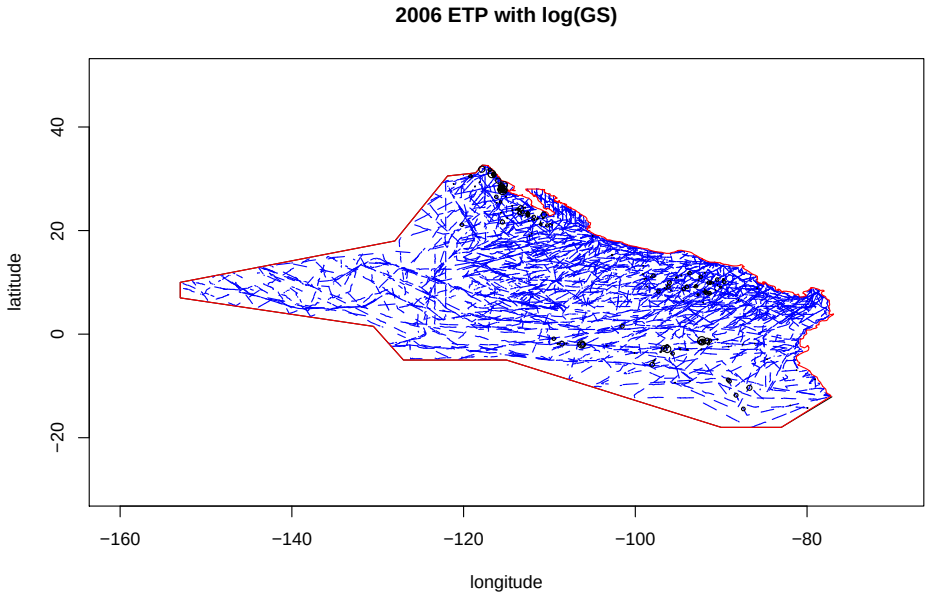
```
    lat    lon
1 32.59 -117.5
2 32.63 -117.8
3 31.13 -118.6
4 30.54 -121.9
5 18.00 -128.0
6 10.00 -153.0
```

```
range(boundSTART[, "lon"])
```

```
[1] -153.0  -77.1
```

```
## part of the boundary is the coastline
coast = read.table(paste(datadir, "AreaSTAR2.dat", sep = ""), sep = ",",
    colClasses = "numeric", comment.char = "*", col.names = c("lat",
        "lon"))
## original dataset is not numberic
coast$lat = as.numeric(as.character(coast$lat))
coast$lon = as.numeric(as.character(coast$lon))
## plot the sighting with effort&Bnd quartz()
plot(boundSTART[, "lon"], boundSTART[, "lat"], ylim = c(-30, 50), xlim = c(-160,
    -70), type = "l", xlab = "longitude", ylab = "latitude", main = "2006 ETP with log(GS)")
lines(eff$lon, eff$lat, type = "l", col = "blue")

lines(coast[, "lon"], coast[, "lat"], type = "l", col = "red")
points(sightdat$Long, sightdat$Lat, col = "black", cex = log(sightdat$GS)/2)
## pch=15 for filled solid circle
```




```
install.packages("fields")
```

```
Error: trying to use CRAN without setting a mirror
```

```
library(fields)
load(file = paste(resultdir, "effNo200plus_nodup.Rdata", sep = ""))
## have checked that effNo200plus has no LT longer than 200km
eff = effNo200plus
# effNo200plus has no duplicated entries and no 200plus LT
# ============================== ## length of the 1st LT segment L1
# =rdist.earth(x1=t(as.numeric(eff[2,c('lon', 'lat')])),
# x2=t(as.numeric(eff[3,c('lon', 'lat')])), miles = FALSE) IDna =
# which(is.na(eff$year), arr.ind=T) IDna = IDna[-1]## loop starts
# from the 2nd LT segment head(IDna) lengthOfAllLTsegments = L1
# for(idx.LTsegment in c(2:length(IDna))) { IDstart =
# IDna[idx.LTsegment-1]+1 IDend = IDna[idx.LTsegment]-1 p0lonlat =
# c(eff[IDstart, c('lon', 'lat')]) p1lonlat = c(eff[IDend, c('lon',
# 'lat')]) L=rdist.earth(x1=t(as.numeric(p0lonlat)),
# x2=t(as.numeric(p1lonlat)), miles = FALSE) lengthOfAllLTsegments
# = c(lengthOfAllLTsegments, L) } save(lengthOfAllLTsegments,
# file=paste(resultdir, 'lengthOfAllLTsegments.Rdata',sep='’))
# which(is.na(lengthOfAllLTsegments), arr.ind=T)
# range(na.omit(lengthOfAllLTsegments)) ID200plus =
# which(lengthOfAllLTsegments>200, arr.ind=T)

##### obtain the original rowID for LT longer than 200km IDna =
```

```
##### which(is.na(eff$year), arr.ind=T) IDna = IDna[-1]## loop starts
##### from the 2nd LT segment head(IDna) IDlength200plus = NULL
##### for(idx.LTsegment in c(2:length(IDna))) { IDstart =
##### IDna[idx.LTsegment-1]+1 IDend = IDna[idx.LTsegment]-1 p0lonlat =
##### c(eff[IDstart, c('lon', 'lat')]) p1lonlat = c(eff[IDend, c('lon',
##### 'lat')]) L=rdist.earth(x1=t(as.numeric(p0lonlat)),
##### x2=t(as.numeric(p1lonlat)), miles = FALSE) if(L>200){
##### IDlength200plus= c(IDlength200plus,c((IDstart-1):IDend))} }
##### save(IDlength200plus, file=paste(resultdir,
##### 'IDlength200plus.Rdata',sep='')) length(IDlength200plus)
##### head(IDlength200plus) # IDend=1866 # IDstart=1865 # p0lonlat =
##### c(eff[IDstart, c('lon', 'lat')]) # p1lonlat = c(eff[IDend,
##### c('lon', 'lat')]) # L=rdist.earth(x1=t(as.numeric(p0lonlat)),
##### x2=t(as.numeric(p1lonlat)), miles = FALSE) effNo200plus =
##### eff[-IDlength200plus,] save(effNo200plus, file=paste(resultdir,
##### 'effNo200plus_nodup.Rdata',sep=''))
```

2.2 Preliminary analysis for the detection function

We start with truncation distance $w = 6km$. We fit a half-normal detection function to the PD distances.

```
sightdat = sightdat[sightdat$Bf ≤ 5, ]
library(Distance)
```

```
Loading required package: mrds
This is mrds 2.1.6
Built: R 3.0.2; ; 2014-06-11 06:27:08 UTC; unix
```

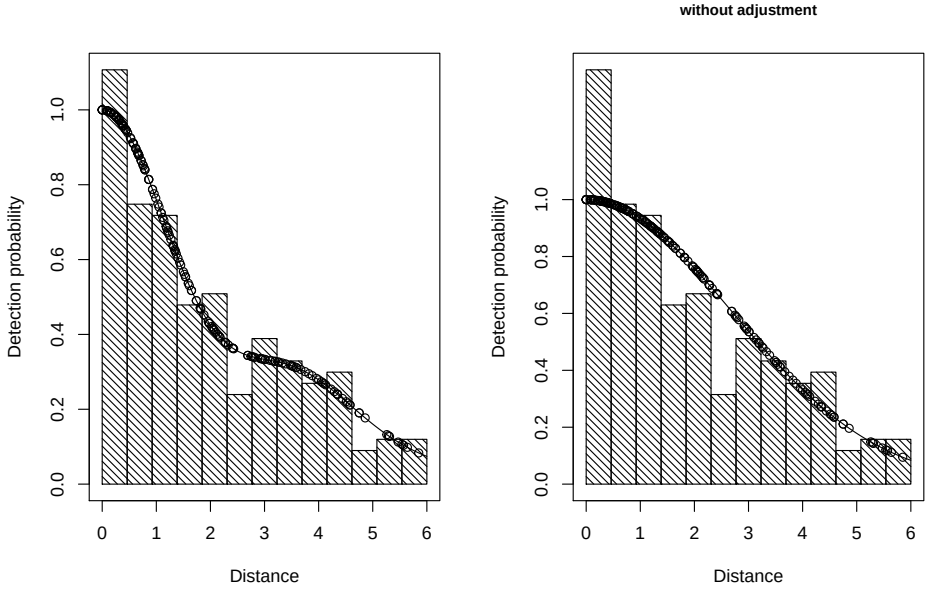
```
xPD = data.frame(distance = sightdat$PD)
mod1 ← ds(xPD, truncation = 6)
```

```
Starting AIC adjustment term selection.
Fitting half-normal key function
AIC= 586.248
Fitting half-normal key function with cosine(2) adjustments
AIC= 582.603
Fitting half-normal key function with cosine(2,3) adjustments
AIC= 582.483
No survey area information supplied, only estimating detection function.
```

```
mod2 ← ds(xPD, truncation = 6, adjustment = NULL)
```

```
Fitting half-normal key function
AIC= 586.248
No survey area information supplied, only estimating detection function.
```

```
par(mfrow = c(1, 2))
plot(mod1)
plot(mod2, main = "without adjustment")
```



3 The log-likelihood of line tranect data as a filtered point process(FPP)

Following the log-likelihood given by (3) in **?**, we give the log-likelihood of line transect survey data with half-normal detection function. We use $\mathcal{C}_k$ to denote the area sampled by the kth transect line, where $k = 1, 2, \ldots, K$. n_k denotes the number of animals sighted on the kth transect line. $\boldsymbol{x}(\boldsymbol{s})$ denotes the covariate vector for site $\boldsymbol{s}$ and $\boldsymbol{\beta}$ is the corresponding parameter vector. $\omega(\boldsymbol{s})$ is the realization of the GMRF at site $\boldsymbol{s}$. $z_k(\boldsymbol{s}_i)$ denotes the perpendicular distance from $\boldsymbol{s}_i$ to the kth transect line. Then the log-likelihood of line transect survey data is

$$l_{LT} = \sum_{k=1}^{K}\sum_{i=1}^{n_k}\left[\boldsymbol{x}(\boldsymbol{s}_i)^T\boldsymbol{\beta} + \omega(\boldsymbol{s}_i) - \frac{1}{2}\left(\frac{z_k(\boldsymbol{s}_i)}{\sigma_g}\right)^2\right] - \sum_{k=1}^{K}\int_{\mathcal{C}_k}\exp\left[\boldsymbol{x}(\boldsymbol{u})^T\boldsymbol{\beta} + \omega(\boldsymbol{u}) - \frac{1}{2}\left(\frac{z_k(\boldsymbol{u})}{\sigma_g}\right)^2\right]d\boldsymbol{u} \quad (1)$$

which consists of two parts: the first is for evaluating the density surface of the filtered point process at the observed locations, and the second is the integral of the density function over the sample area.

We approximated the integration part numerically by

evaluate the field at the integration points

scheme designed on tangent planes to the sphere.

4 The integration steps for each transect segment

Here I do not use the term transect line, as in practice, each whole transect line consists of several transect segments, as the ship has to stop for some reasons.

The effort dataset uses NAs to separate different transect segments. Let p_0 denote the starting point of a transect segment, and p_1 denote the finishing point of a transect segment. The earth is considered as a sphere in a Euclidean space $\mathcal{R}^3$ with the centre of the earth being origin, denoted by $\mathcal{O}$. Let $\overrightarrow{p_0p_1}$ denote the vector starting from p_0 and pointing towards p_1.

It is very unlikely that transect lines are perfectly on the east-to-west or north-to-south drection, for which the boundary of transect strips is easy to specify. In the longitude and latitude system, it is difficult to apply geometric operations to find vectors that are perpendicular to $\overrightarrow{p_0p_1}$ on each side of the transect line, and loction the points that are 10 km away from p_0 along the perpendicular direction on each side.

So first we transform the longitude and latitude system into $\mathcal{R}^3$ so that geometric operations become applicable. Then we construct local coordinate system for each $\overrightarrow{p_0p_1}$ with p_0 as the new origin, and design the integration on the tangent plane defined by $\overrightarrow{p_0p_1}$ and a unit vector that is perpendicular to $\overrightarrow{p_0p_1}$. For simplicity, we use $\boldsymbol{v}_1$ to denote $\overrightarrow{p_0p_1}$, and $\boldsymbol{v}_3$ to denoted the direction that is perpendicular to $\overrightarrow{\boldsymbol{w}_0}$. Following the righthand rule, v_1 is the 3rd coordinate which is perpendicular to the tangent plane, denoted by $\boldsymbol{v}_1$.

The advange of the $(\boldsymbol{v}_1\boldsymbol{v}_2, \boldsymbol{v}_3)$ system with p_0 as origin is that $\boldsymbol{v}_3$ and $-\boldsymbol{v}_3$ give 90 degree counterclockwise and clockwise to transect lines, respectively. Furthermore, if the truncation distance is 10km and transect segment is

39km, then the four corners of the transect rectangle can be simplied specified by $(0,0,10)$, $(0,0,-10)$, $(0,39,10)$ and $(0,39,-10)$, on top of which the integration grid is then designed to approximate the point process likelihood. (These transects regions are very small, so the Euclidean distance in the tangent plane is almost he same as the spherical distance. Note that both of those distances are not precisely the same as the true distance metric anyway. For now, we use the mean radius of the earch and ignore the fact that the earth radius varies by latitude.)

The integatrion points are the centers of each cell in an equally spaced grid on the transect tectangle, and the size of each grid cell is viewed as the weight for each integration point. We transform these integration points from (v_1v_2, v_3) back to longtitude and latitude, but keep the weights in the tangent plane system as the effect of curvation on the change of the area of transect strip is ignorable given the small size of transect strips in comparison of the earth.(Finn pointed out that at some point a more exact formula should be written down, if only to prove that it's not needed).

- Step1: tranform to $\mathcal{R}^3$
  For any point $\boldsymbol{p}$ with longitude and latitude, $long_p$ and lat_p, we use the following tranformation to find its coordinates (px, py, pz) in $\mathcal{R}^3$

$$\begin{aligned} px &= Rcos(long_p)cos(lat_p) && (2)\\ py &= Rcos(lat_p)sin(long_p) && (3)\\ pz &= Rsin(lat_p) && (4) \end{aligned}$$

- Step2: for any p_0 and p_1, calculat the tangent plane system
  The following calculation should be down in the shown sequence

$$\begin{aligned} \boldsymbol{v}_1 &= \boldsymbol{p}_0 \Big/ \|\boldsymbol{p}_0\| && (5)\\ \boldsymbol{v}_3' &= \boldsymbol{v}_1 \times (\boldsymbol{p}_1 - \boldsymbol{p}_0); \boldsymbol{v}_3 = \boldsymbol{v}_3'/\|\boldsymbol{v}_3'\| && (6)\\ \boldsymbol{v}_2 &= \boldsymbol{v}_3 \times \boldsymbol{v}_1 && (7) \end{aligned}$$

- Step 3 : Define integration grid on the tangent plane on the (v_1v_2, v_3) system

  Let R denote the rotation matrix which gives the coordiates with respect to (v_1v_2, v_3)

$$\boldsymbol{R} = \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{bmatrix} \quad (8)$$

  where $\boldsymbol{v}_1 = (v_{11}, v_{12}, v_{13})^T$.

  For any vector $\boldsymbol{v}$ (a column vector) in $\mathcal{R}^3$, we tranform $\boldsymbol{v}$ in the tangent plane system by using the totation matrix $\boldsymbol{R}$, and the new vector is

$$\boldsymbol{v}' = \boldsymbol{R}\boldsymbol{v} \quad (9)$$

  Note that $\boldsymbol{R}$ is an orthogonal matrix and $\boldsymbol{R}^{-1} = \boldsymbol{R}^T$, for any $\boldsymbol{v}'$ in (v_1v_2, v_3) system, we simply tranform it back to $\mathcal{R}^3$ by

$$\boldsymbol{v} = \boldsymbol{R}^T\boldsymbol{v}' \quad (10)$$

  where $\boldsymbol{p} = (px, py, pz)^T$.

  Vectors are simpler to transform than points as vectors represent directions in space whereas points represent locations in space. The direction is not affected by change of origin of coordinate system, however the points are. When transforming a point into a new coordinate sysmte, we need to incorporate the information about the change of origin.

  For any point $\boldsymbol{p} = (px, py, pz)^T$ in $\mathcal{R}^3$, if we use $\boldsymbol{p}_0$ as the new origin in the tangent plane system, then the coordinates of $\boldsymbol{p}$ after tranformation is

$$\boldsymbol{p}' = \boldsymbol{R}(\boldsymbol{p} - \boldsymbol{p}_0) \quad (11)$$

- Step 4: design a grid on plane given by $(v2, v3)$

- Step 5: transform back the points on grid to (lon,lat) and evaluate the PP.

the mean earth radius = 6371 km

5 Design the integration on the tangent plan

```
## detail step functions are in file 'RfunsIntegrationDesign.r'
range(na.omit(eff$lon))
```

```
[1] -152.99  -77.26
```

```
range(na.omit(eff$lat))
```

```
[1] -18.00  32.58
```

```
## step1: tranform from long&lat to XYZ using mean radius
p0lonlat = c(eff[5, c("lon", "lat")])
p0 = tranformLongLatToXYZ(long = p0lonlat$lon, lat = p0lonlat$lat,
    R = 6371)

p1lonlat = c(eff[6, c("lon", "lat")])
p1 = tranformLongLatToXYZ(long = p1lonlat$lon, lat = p1lonlat$lat,
    R = 6371)

# Radiants = degree*pi/180 degree = radiant*180/pi
cos(0)
```

```
[1] 1
```

```
cos(90/180 * pi)
```

```
[1] 6.123e-17
```

```
## note that cosine in R works on degree instand of radiant
cos(29.7667/180 * pi)
```

```
[1] 0.8681
```

All the code to produce integration matrix can be found in "rfunIntegrationDesign.R".

Which new orgin to use, $v1 = p0$ or $v1 = -p0$, does not matter for the new coordinates in the new system. Here we use p0 as the new origin. I have checked my code by tranforming back and forth and it gives consistent results. As long as p0 is used as the origin in the new coordinate system, it does matter v1 = p0 or v1 = -p0.

Note the simplicity resulting from using (v1, v2, v3) system. Summarise these steps in one function and also another one to transform back

The configiration of the integration grid:

- Along the direction perpendicular to the transect line, the cell size is chosen to be be divisors of the truncation distance w. For the most sparse grid we use the cell size equal to w.
- It is slightly more complicated design for the integration cell size along the transect direction. As the length of transect segments vaies from 0.01km to 200 km (for now I exlucde all the segments that are longer than 200 km), we deal with the segments differently depending on their length
  - if the length of a segment is no longer than dLT (the cell size along line transect direction), then we simply treat this segment as one grid cell and the integration points along the middle of the transect line.
  - if the length of a segment is longer than dLT, then we divide the transect segment by dLT. Let nLT denote the number of cells along the transect direction. If the reminder of length(segment) divided by dLT is larger than 0.5, then $nLT = round(length(segment)/dLT)$, otherwise, $nLT = round(length(segment)/dLT) - 1$
- Note that not all integration cells have the same weight, though most of them have the size $dLT \times dPD$

```
## to check the code by using the 1st point in intgrid
## lonlatTopIntGrid.onsphere=convert.fromXYZtoLongLat(pxyz=topIntGrid.onsphere,R=6371)
## lonlatTopIntGrid.onsphere ???? equally spaced grid, the weight is
## just dLT*dPD

## transfrom from xyz to long&lat???  lat=atan2(z,sqrt(x*x+y*y))
```

```
## long=atan2(y,x) Given a point location p and normalised transect
## direction v, the perpendicular CCW (counterclockwise) vector is
## given by the cross-product $w = p \times v$.  (p,v,w) then forms
## the basis of a local coordinate system, with (1,a,b) being the
## coordinates of a point in the tangent plane.  This can then be
## transformed back into longitude latitude by first normalising the
## point to have unit length (so it's on the sphere, and then
## computing the long-lat mapping.  The mappings can be done with ##
## the following funs simply provide mapping between R3 and sphere
## long&lat system
onthesphere = inla.mesh.map(loc = intgrid.lonlat.try1 , projection = "longlat")
```

```
Error: object 'intgrid.lonlat.try1' not found
```

```
lonlat = inla.mesh.map(onthesphere , projection = "longlat", inverse = FALSE)
```

```
Error: object 'onthesphere' not found
```

```
str(onthesphere)
```

```
Error: object 'onthesphere' not found
```

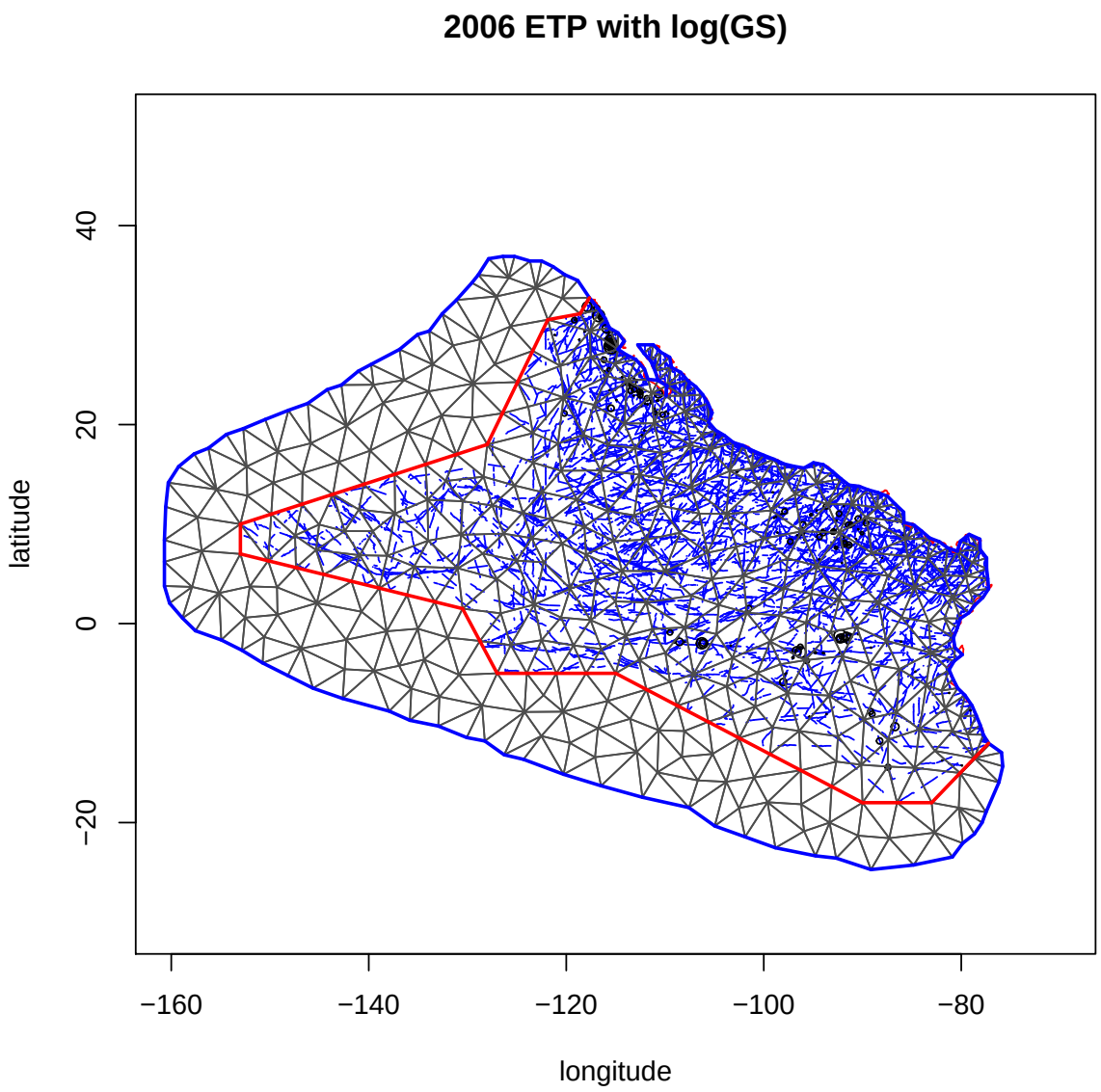
```
str(lonlat)
```

```
Error: object 'lonlat' not found
```

6 Stack together: integration points

In future coding, A.pp, e.pp and y.pp should be kept together, as any change in one of them, the rest should be changed accordingly.

```
load(file = paste(datadir, "mesh_stitchInterior.Rdata", sep = ""))
plot(boundSTART[, "lon"], boundSTART[, "lat"], ylim = c(-30, 50), xlim = c(-160,
    -70), type = "l", xlab = "longitude", ylab = "latitude", main = "2006 ETP with log(GS)")
lines(eff$lon, eff$lat, type = "l", col = "blue")
lines(coast[, "lon"], coast[, "lat"], type = "l", col = "red")
points(sightdat$Long, sightdat$Lat, col = "black", cex = log(sightdat$GS)/2)
plot(mesh, add = T)
```




```
# nv = mesh$n
spde = inla.spde2.matern(mesh = mesh, alpha = 2)
spde$n.spde
```

```
[1] 443
```

```
head(sightdat)
```

```
  Sp1 Sp2 Sp3 Sp4 Cruz Sght E Obs Year Month Day Time    Lat
1  75  NA  NA  NA 1267    1 1   7 1989    11  11  639 10.619
2  75  NA  NA  NA 1081    1 1  51 1987     8  10  816 26.513
3  75  NA  NA  NA 1268    1 1  55 1989     9  24  826 -1.256
5  75  NA  NA  NA 1614    1 1 198 1999     7  28 1622 31.731
6  75  NA  NA  NA 1370    1 1  55 1990    12   2  658 21.641
7  75  NA  NA  NA 1081    2 1  64 1987    10  24 1030  9.918
     Long   GS TotGS   PD Bf SH  SD RF HS VS WSp WDi Cue Me Ph Bi
1  -98.95 1.16  1.16 1.37  2 -1  -1  1  1  3  -1  -1   6  4 -1  0
2 -116.18 2.32  2.32 0.78  2 -1  -1  1 -1 -1  -1  -1   6  4 -1  0
3  -91.31 2.32  2.32 1.59  4 -1  -1  2 -1 -1  -1  -1   6  5 -1  0
5 -116.94 2.32  2.32 0.93  3  4 302  1  3  2  13 323   6  4  0  0
6 -115.47 2.76  2.76 0.16  5 -1  -1  1  4  3  -1  -1   6  5 -1  0
7  -96.03 2.65  2.65 1.65  3 -1  -1  1 -1 -1  -1  -1   6  4 -1  0
  Mx Crs  SST  Vis Angl Retcl RadD InitID MSp
1  0  65 27.2 -1.9   21   1.0 3.84     -1  -1
```

```
2  0 154 21.7 -1.9   20   2.4 2.27     -1  -1
3  0 120 19.4 -1.9   35  -1.0 2.78     -1  -1
5  0 156 -1.0 11.1  346   1.0 3.83     -1  -1
6  0 343 -1.0 -1.9  350  -1.0 0.93     -1  -1
7  0 203 27.8 -1.9   20   0.6 4.84     -1  -1
```

```
nrow(sightdat)
```

```
[1] 194
```

```
sightdatTrunc = sightdat[sightdat$PD ≤ 6, ]
nrow(sightdatTrunc)
```

```
[1] 181
```

```
sightloc = cbind(lon = sightdatTrunc$Long, lat = sightdatTrunc$Lat)
## ndata is the no. of sightings (truncated)
ndata = nrow(sightloc)
## 194 sighting (LT mode) for BW over 12 years
length(unique(na.omit(sightdatTrunc$Year)))
```

```
[1] 12
```

```
## matrix for evaluating likelihood at the sighting locations
locmat = inla.spde.make.A(mesh, loc = sightloc)
dim(locmat)
```

```
[1] 181 443
```

```
############ load the integration points matrix
version = "_v9"
load(paste(resultdir, "IntPointMatrixLonLat", version, ".Rdata", sep = ""))
nrow(IntPointMatrixLonLat)
```

```
[1] 140328
```

```
head(IntPointMatrixLonLat)
```

```
        lon   lat weight
[1,] -115.8 29.72     36
[2,] -115.8 29.63     36
[3,] -115.8 29.72     36
[4,] -115.8 29.64     36
[5,] -115.9 29.71     36
[6,] -115.9 29.63     36
```

```
## load the v1v2v3 integration design matrix on v1v2v3
load(file = paste(resultdir, "IntMatrixv1v2v3List", version, ".Rdata",
    sep = ""))
dPD = unique(IntMatrixv1v2v3List[[1]]$dPD)
## integration matrix
intmat = inla.spde.make.A(mesh, IntPointMatrixLonLat[, -3])
nInt = nrow(intmat)
## PD for the integtration points
xPDIntPoints = rep(dPD, nrow(IntPointMatrixLonLat))

############ combine locmat and intmat together
A.pp = rBind(intmat, locmat)
dim(A.pp)
```

```
[1] 140509    443
```

```
y.pp = rep(0:1, c(nInt, ndata))
length(y.pp)
```

```
[1] 140509
```

```
# , nInt is no. of integration points and n is number of
# observations
e.pp = c(IntPointMatrixLonLat[, "weight"], rep(0, ndata))
length(e.pp)
```

```
[1] 140509
```

```
######### the stack: without any covariates; stk.pp =
######### inla.stack(data=list(y=y.pp, e=e.pp), A=list(A.pp), tag='pp',
######### effects=list(i=1:spde$n.spde)) Your effects info is wrong. The
######### effect should index the latent model, not the integration points
######### (which from the point of view of inla are 'data'). the code
######### should be effects=list(i=1:spde$n.spde)
```

Note that I don't have the SST data for all the 12 years, therefore, I cannot incorporate the SST now.

```
stk = inla.stack(data=list(y=y.pp, e = e.pp),
                 A=list(A.pp,1), tag='fPP1',
                  effects=list(list(i=1:spde$n.spde),
                               data.frame(Intercept=1,
                                          Lon = c(IntPointMatrixLonLat[,"lon"], sightloc[,"lon"]),
                                          Lat = c(IntPointMatrixLonLat[,"lat"], sightloc[,"lat"]),
                                                  # SST = c(SSTonMesh, SSTonSightLoc),
                                                  xPDmod =c(-xPDIntPoints^2/2,
                                                            -as.numeric(sightdatTrunc$PD)^2/2)
                                          #need to transform the xPD as a covariate into INLA.call
                                                  )))
# formula for Flitered PP
formula.fpp = y~ -1+ Intercept + Lon +Lat + f(xPDmod, model="clinear", range = c(0, Inf))
                   + f(i, model=spde)
```

```
Error: invalid argument to unary operator
```

```
## have to run the code on INLA server ##
# result.fpp = inla(formula.fpp, family = "poisson",   data=inla.stack.data(stk),
#               control.predictor=list(A=inla.stack.A(stk)),
#               # the following line is necessary, the filtering component is treated as an iid random effect w
#               control.family = list(hyper = list(prec = list(initial =log(1/0.01^2),
#                                                              fixed=TRUE))),
#                E=inla.stack.data(stk)$e, inla.call="remote")
#
# summary(result.fpp)
#
# plot(result.fpp)
```

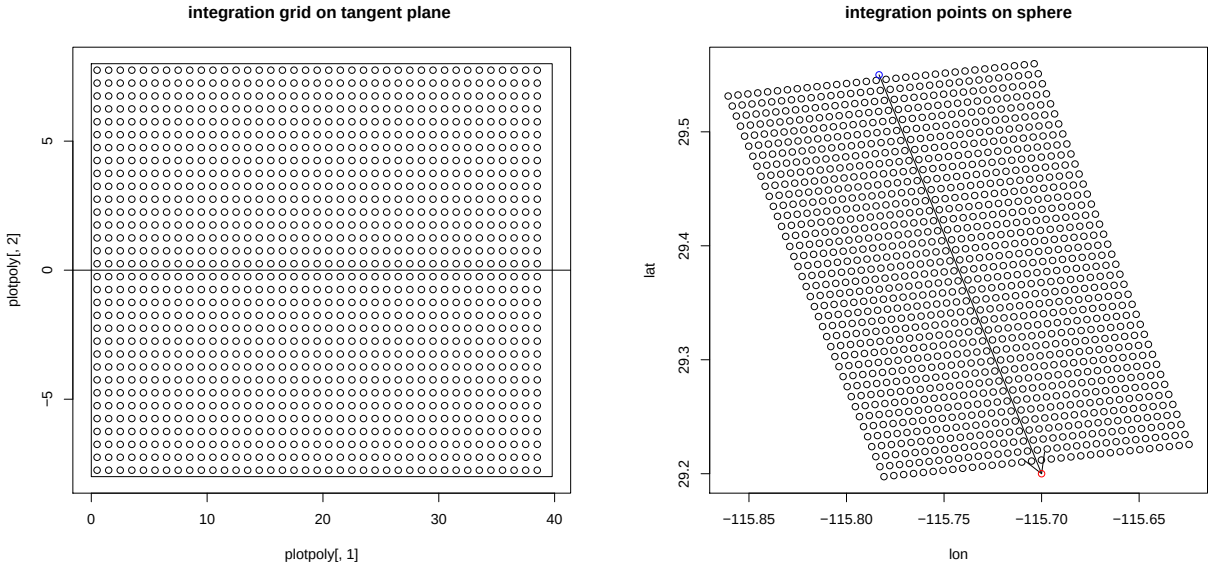


Figure 2: Design integration grid on tangent plane and then tranform to sphere: the integration points and weights are all specified on the tangent plane, and the integration points are then transformed to the longitute and latitute on the sphere, while the integration weights stay the same as the curvation effect is ignored here.