

SOP 10: Optimizing Survey Design

Version 1.0

Contents

1	Summary	2
2	Setting up the R workspace	2
3	Load model fitting results from SOP 9	2
4	Identify optimal set of random transects	3
5	Plot optimal random design	4
6	Exchange algorithm	6
7	Obtaining coordinates of the optimal design	8
7.1	West Arm of Glacier Bay	8
7.1.1	Coordinate conversion	9
7.2	East Arm of Glacier Bay	13
7.2.1	Coordinate conversion	15
7.3	Lower Glacier Bay	17
7.3.1	Coordinate conversion	19

List of Figures

1	Optimal random transects	7
2	West Arm transects	10
3	East Arm transects	14
4	Lower Glacier Bay transects	18

List of Tables

1 Summary

This protocol describes how to use available software to calculate optimal survey designs for sea otters in Glacier National Park. The methods used to estimate spatio-temporal occupancy and abundance are described in detail in Williams et al. (2017a), Williams et al. (2017b), and the methods used to calculate the optimal survey design is described in Williams et al. (2017c). The analysis of the data can be conducted using the **stoa** package (spatio-temporal occupancy abundance) in R statistical software. Steps describing the analysis are provided in the following sections.

2 Setting up the R workspace

Run the following script to install and load required packages needed for the analysis.

```
required.packages=c("coda", "devtools", "fBasics", "fields",  
                    "gridExtra", "inline", "parallel", "raster", "rasterVis",  
                    "RColorBrewer", "RcppArmadillo", "roxygen2" )  
  
## install.packages(required.packages)  
lapply(required.packages,library,character.only=TRUE)  
  
## Local stoa package  
setwd("~/Dropbox/Post-Doc/RFiles/EcologicalDiffusion/package")  
install("stoa")  
library(stoa)
```

3 Load model fitting results from SOP 9

```
load("~/Dropbox/Post-Doc/TechnicalReport/MCMCOutput.RData")  
(status=sum(out[[3]][,1]!="NA",na.rm=TRUE))  
  
## [1] 57500  
  
## Assign model output to objects  
C.ppd.tmp=out[[14]]  
C=out[[15]]  
CISU=out[[16]]  
N=out[[17]]  
Y=out[[18]]  
X=out[[19]]  
W=out[[20]]  
cell=out[[21]]  
ISU=out[[22]]
```

```

BoundaryNA=out[[23]]
new.data=out[[25]]

## Starting values
starting.values.l=list()
starting.values.l[[1]]=out[[3]][status,]
starting.values.l[[2]]=out[[4]][status,]
starting.values.l[[3]]=out[[5]][status,]
starting.values.l[[4]]=out[[6]][status,]

## Model settings
model.settings=out[[24]]

## Tuning parameters
starting.values.l[[5]]=out[[11]][status,]

## Number of MCMC iterations and save point
n.iter=20
check.point=10

```

4 Identify optimal set of random transects

The first step in choosing the optimal transects is to decide how many transects can be flown in a given year, n . Then randomly select n transects from the available transects that could be flown, and calculate the predictive variance (i.e., the design criterion) for these set of n transects. One set of n transects is one *design*. Then repeat this for a large number of designs (e.g., 64 designs, each containing $n = 20$ random transects were examined for 2017). Of the differing designs that are examined, choose the design that minimizes the design criterion. Note that this process can be done in parallel using a cloud cluster for computation. Generally, desktop PCs only contain 8–16 cores, whereas 64 can be obtained easily using cloud computing. The following algorithm may take several days to complete.

```

###
### Fit model to data collected from multiple designs in parallel
###

no.rand.transects=20
no.designs=4 ## 64 were used for the 2017 surveys using a cloud cluster
seq.of.random.seeds=sample(1:10000,no.designs,replace=FALSE)

## Fit models in parallel
MCMCOutput=run.chain.2pl.list(n.iter,check.point,seq.of.random.seeds,
                              C.ppd.tmp,C,CISU,N,Y,X,W,cell,ISU,

```

```
BoundaryNA,new.data,starting.values.l,  
model.settings,no.rand.transects)
```

After the algorithm has completed, the design criterion from the different designs can be assessed, and the design that is optimal can be identified.

```
## Function to calculate the design criterion  
calc.q=function(x){  
  y=sum((x-mean(x))^2)/length(x)  
  return(y)  
}  
  
## Calculate the design criterion  
scores=numeric(no.designs)  
for(i in 1:no.designs){  
  scores[i]=calc.q(MCMCOutput[[i]][,1])  
}  
  
## Identify which design criterion was best  
(best.score=which(scores==min(scores)))  
  
## [1] 2  
  
## The value of the design criterion to compare in the Exchange Algorithm (below)  
(OptimalScore=scores[scores==min(scores)])  
  
## [1] 658347  
  
## Save the seed of the optimal random design so it can be recovered  
(seed=seq.of.random.seeds[best.score])  
  
## [1] 5346
```

5 Plot optimal random design

The following script allows us to view the optimal random design, with respect to the prediction variance of abundance.

```
## Spatial and temporal settings  
xmin=model.settings[[1]][1]  
xmax=model.settings[[1]][2]  
ymin=model.settings[[1]][3]  
ymax=model.settings[[1]][4]
```

```

x=model.settings[[1]][5]
y=model.settings[[1]][6]
keep=model.settings[[2]]
dt=model.settings[[3]]
timesteps=model.settings[[4]]
us.fact=model.settings[[5]]
d=model.settings[[6]]
years=model.settings[[7]]
q=model.settings[[8]]

## Boundary layers
BoundaryInf=BoundaryNA
BoundaryInf[floor((ymax-6485000)/400):floor((ymax-ymin)/400),
            1:floor((435000-xmin)/400)]=NA
BoundaryInf[floor((ymax-6471200)/400):floor((ymax-ymin)/400),
            1:floor((xmax-xmin)/400)]=NA
BoundaryInf[floor((ymax-6474400)/400):floor((ymax-ymin)/400),
            floor((449000-xmin)/400):floor((xmax-xmin)/400)]=NA

## Recover random transects
NumRandTransects=no.rand.transects
row.ind=matrix(,y,x)
for(i in 1:y){
  row.ind[i,]=ifelse(!is.na(BoundaryInf[i,]),i,NA)
}
designs=sort(row.ind[!is.na(row.ind)])
set.seed(1)
design.tmp.rand=sort(sample(unique(designs[designs>30]),
                             NumRandTransects))
designRand=which(designs==design.tmp.rand[1])
for(i in 2:length(design.tmp.rand)){
  designRand=c(designRand,which(designs==design.tmp.rand[i]))
}

## Recover optimal random transects
NumOptTransects=20
set.seed(seed)
design.tmp.opt=sort(sample(
  unique(designs[!designs%in%design.tmp.rand]),NumOptTransects))
designOpt=which(designs==design.tmp.opt[1])
for(i in 2:length(design.tmp.opt)){
  designOpt=c(designOpt,which(designs==design.tmp.opt[i]))
}
design=sort(c(designRand,designOpt))

```

```

## Plot map and transects

param.values=apply(MCMCOutput[[best.score]],2,mean,na.rm=TRUE)
lambda=calc_lambda(param.values)
lambda2017=lambda[[25]]
lambda2017[!is.na(lambda2017[])] [design]=30.9999
plot(lambda2017)

```

6 Exchange algorithm

After the optimal random transects are identified, the design can be improved using an exchange algorithm. Each of the n optimal transects are exchanged with their neighbors (with the transect above it (to the North), and with the transect below it. After each transect is exchanged, the model is refit, and the design criterion is calculated for each exchange. If the design criterion is improved with the exchange, the new transect is kept, and the old transect is discarded. Then the next transect on the list is exchanged. The process repeats until the design criterion cannot be improved through exchange. Because each exchange requires refitting the model, and it must occur sequentially (except for examining the two immediate neighbors, which can occur in parallel), this requires fitting the model numerous times, and therefore, is time consuming. Below is script for an example of one exchange, where the first transect `t.id=1` is exchanged for both of its neighbors.

```

t.id=1      ## t.id will need to be examined for t.id=1,...,n
            ## and then repeated until no further exchanges are needed
transect.id=design.tmp.opt[t.id]
design.rows=list()

## Exchange the first transect with transect above it
design.rows[[1]]=design.tmp.opt
design.rows[[1]][t.id]=design.rows[[1]][t.id]+1

## Exchange the first transect with transect below it
design.rows[[2]]=design.tmp.opt
design.rows[[2]][t.id]=design.rows[[2]][t.id]-1

## Run MCMC with new designs (2 parallel chains)
## Fit model
no.designs=2
designs=design.rows[1:no.designs]
designID=1:no.designs
MCMCOutputExchange=run.exchange.2pl.list(n.iter,check.point,this.design,
                                         this.ID,C.ppd.tmp,C,CISU,N,Y,X,
                                         W,cell,ISU,BoundaryNA,new.data,

```

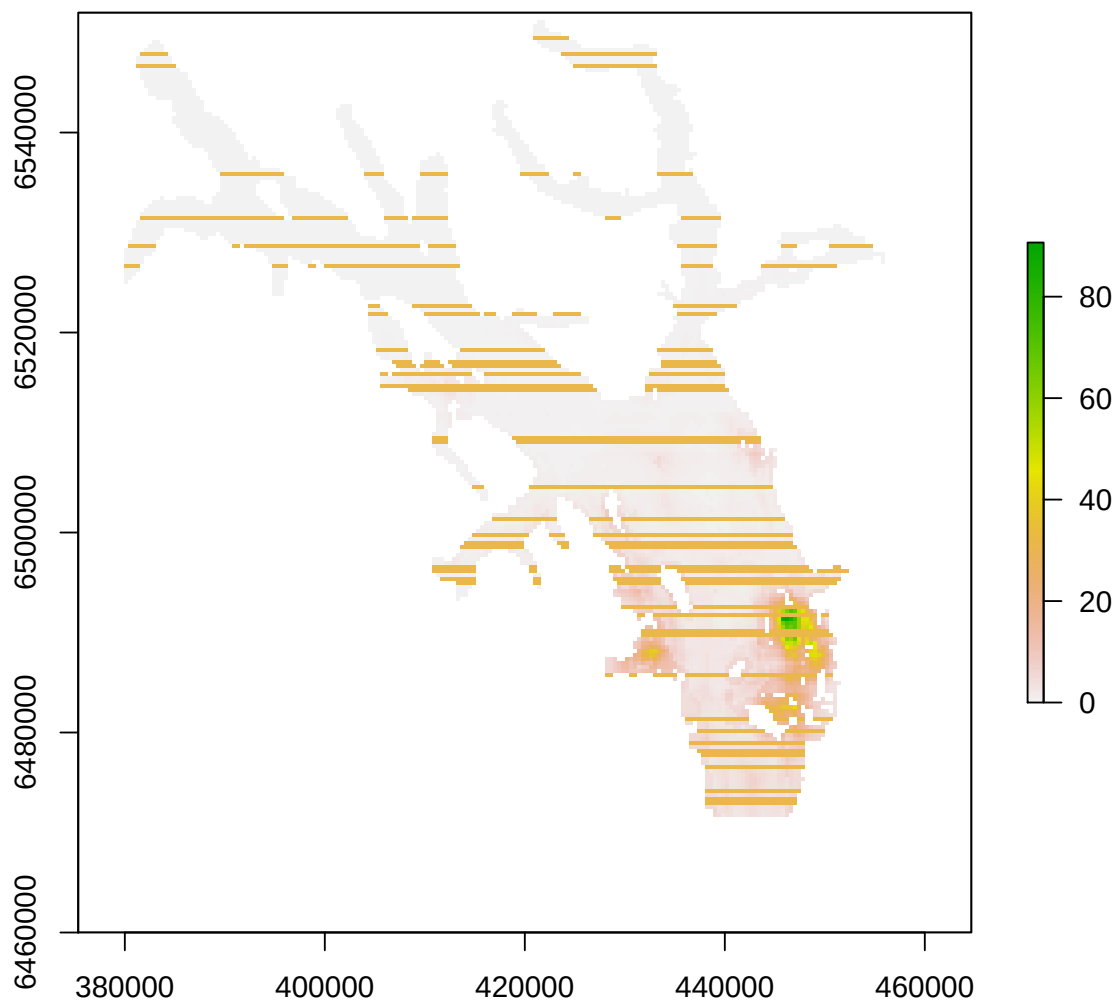


Figure 1: Optimal random transects

```
starting.values.l,model.settings,  
no.rand.transects)
```

After the models are fit, we can examine the design criteria and compare to the original design criteria `OptimalScore`.

```
## Calculate the design criterion  
ExchangeScores=numeric(no.designs)  
for(i in 1:no.designs){  
  ExchangeScores[i]=calc.q(MCMCOutputExchange[[i]][,1])  
}  
  
exchange_action(OptimalScore,ExchangeScores,transect.id)  
  
## [1] "No exchange"
```

When an exchange is necessary, the following script is used to exchange it. For example, if we were indicated to exchange transect 7 to transect 8:

```
ind=which(design.tmp.opt==7)  
design.tmp.opt[ind]=8
```

7 Obtaining coordinates of the optimal design

After the exchange algorithm has been completed, we can obtain the coordinates of the optimal design. We simplify the process by breaking Glacier Bay into three sections; the West Arm (WA), the East Arm (EA), and Lower Glacier Bay (LB).

7.1 West Arm of Glacier Bay

```
## Calculate coordinates in NAD27 UTM 8N  
  
## define the spatial extent of the West Arm  
ewa=extent(xmin,  
           427800,  
           6513000,  
           ymax)  
WA=crop(lambda2017,ewa)  
  
## Remove portion of West Arm  
WA[1:60,90:120]=NA
```

```

## Plot the results
plot(WA)

## Obtain coordinates
WA.coords.tmp=matrix(,3)
for(i in 1:dim(WA)[1]){
  if(length(which(WA[i,]==30.9999))!=0){
    x.start=head(which(WA[i,]==30.9999),1)*400+e@xmin-200
    x.end=tail(which(WA[i,]==30.9999),1)*400+e@xmin-200
    y=e@ymax-i*400+200
    WA.coords.tmp=rbind(WA.coords.tmp,c(x.start,x.end,y))
  }
}

## Remove first row of NAs and name columns
WA.coords=WA.coords.tmp[-1,]
colnames(WA.coords)=c("lon.west","lon.east","lat")

## Output coordinates in NAD27 UTM ZONE 8N
WA.coords

##      lon.west lon.east   lat
## [1,]   381800   384200 6508800
## [2,]   381400   385000 6507600
## [3,]   389800   412200 6496800
## [4,]   381800   412200 6492400
## [5,]   380600   413000 6489600
## [6,]   380200   413400 6487600
## [7,]   404600   414600 6483600
## [8,]   404600   425400 6482800
## [9,]   405400   421800 6479200
## [10,]  407000   423000 6478000
## [11,]  407400   423400 6477600
## [12,]  405800   425400 6476800
## [13,]  405800   426600 6475600
## [14,]  408600   427000 6475200

```

7.1.1 Coordinate conversion

To convert coordinates from NAD27 UTM Zone 8N to either Decimal Degrees, or Degrees Minutes Seconds, we can use the following script. Note that WA.coords has three columns: lon.west, lon.east, and lat. The coordinates lon.west and lon.east correspond to the the end-points of transects on the west and east side of Glacier Bay, respectively. The

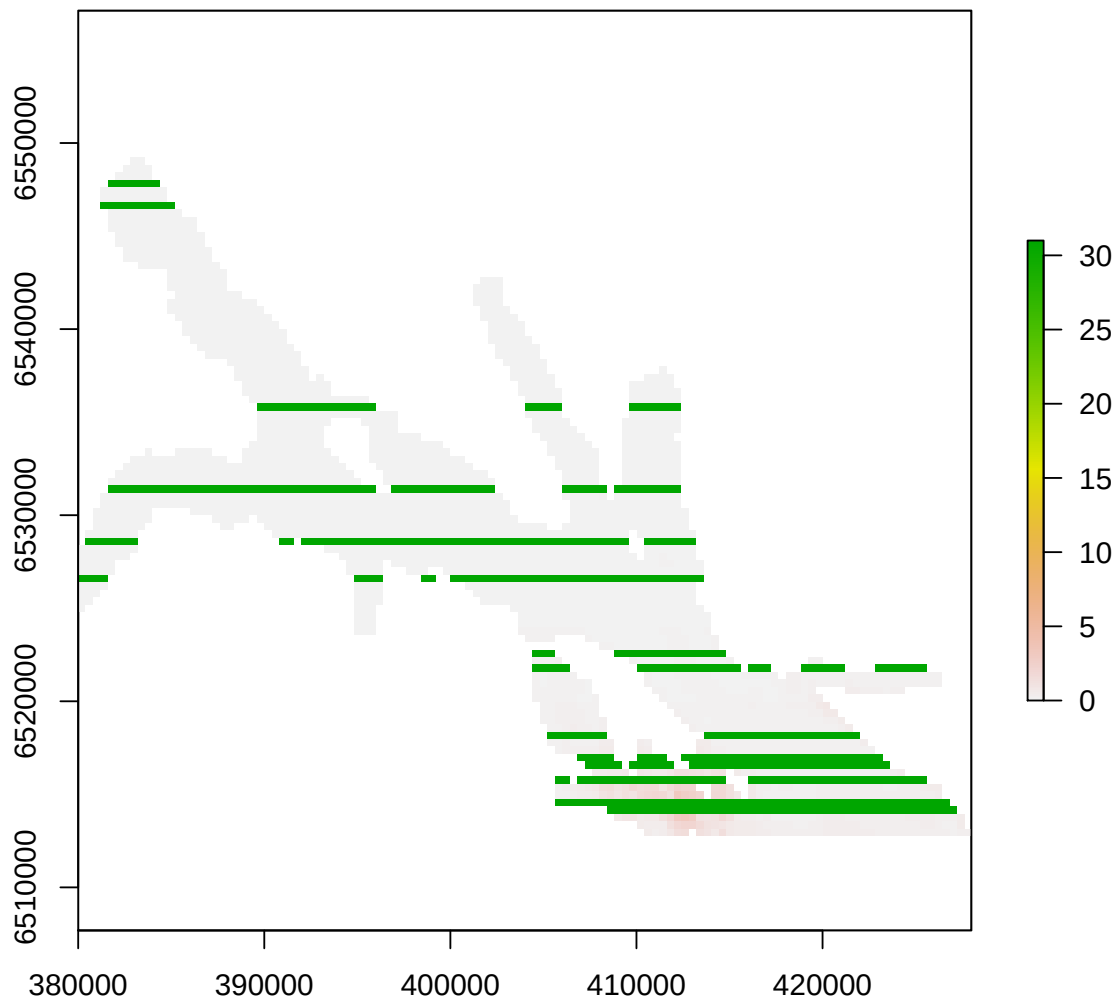


Figure 2: West Arm transects

latitude are similar, but not exact due to transformations among coordinate systems. ALL COORDINATES ARE FOR THE WEST ARM OF GLACIER BAY. Below `east` and `west` refer to the east and west side of the west arm.

```
## Convert to Decimal Degrees
west.pointsWA=data.frame(lon=WA.coords[,1],lat=WA.coords[,3])
east.pointsWA=data.frame(lon=WA.coords[,2],lat=WA.coords[,3])
coordinates(west.pointsWA)=c("lon", "lat")
proj4string(west.pointsWA)=
  CRS("+proj=utm +zone=8 +ellps=clrk66 +datum=NAD27 +units=m +no_defs") #26708
coordinates(east.pointsWA)=c("lon", "lat")
proj4string(east.pointsWA)=
  CRS("+proj=utm +zone=8 +ellps=clrk66 +datum=NAD27 +units=m +no_defs") #26708
CRS.new=CRS("+proj=longlat +datum=WGS84")
west.pointsWA=spTransform(west.pointsWA,CRS.new)
east.pointsWA=spTransform(east.pointsWA,CRS.new)
west.pointsWA@coords

##           lon      lat
## [1,] -137.0419 58.70467
## [2,] -137.0482 58.69379
## [3,] -136.8981 58.59908
## [4,] -137.0334 58.55749
## [5,] -137.0525 58.53203
## [6,] -137.0584 58.51398
## [7,] -136.6380 58.48409
## [8,] -136.6377 58.47691
## [9,] -136.6225 58.44476
## [10,] -136.5946 58.43433
## [11,] -136.5876 58.43083
## [12,] -136.6146 58.42330
## [13,] -136.6142 58.41253
## [14,] -136.5661 58.40953

east.pointsWA@coords

##           lon      lat
## [1,] -137.0006 58.70532
## [2,] -136.9861 58.69476
## [3,] -136.5128 58.60419
## [4,] -136.5111 58.56469
## [5,] -136.4963 58.53971
## [6,] -136.4886 58.52184
## [7,] -136.4666 58.48616
## [8,] -136.2811 58.48097
```

```
## [9,] -136.3416 58.44801
## [10,] -136.3206 58.43745
## [11,] -136.3137 58.43393
## [12,] -136.2792 58.42709
## [13,] -136.2582 58.41652
## [14,] -136.2513 58.41300
```

```
## Convert western lat long DD points of West Arm to DMS
convert_DD_to_DMS(west.pointsWA@coords)
```

```
##      D_lon M_lon      S_lon D_lat M_lat      S_lat
## [1,]  -137      2 31.012875    58    42 16.803398
## [2,]  -137      2 53.580307    58    41 37.642059
## [3,]  -136     53 53.175264    58    35 56.685504
## [4,]  -137      2  0.226539    58    33 26.964601
## [5,]  -137      3  9.151293    58    31 55.325582
## [6,]  -137      3 30.090150    58    30 50.315441
## [7,]  -136     38 16.821326    58    29  2.719404
## [8,]  -136     38 15.622193    58    28 36.864824
## [9,]  -136     37 20.911026    58    26 41.144827
## [10,] -136     35 40.519635    58    26  3.598866
## [11,] -136     35 15.286179    58    25 50.976940
## [12,] -136     36 52.710117    58    25 23.890791
## [13,] -136     36 50.941927    58    24 45.108028
## [14,] -136     33 57.901532    58    24 34.318739
```

```
## Convert eastern longitude points of West Arm to DMS
convert_DD_to_DMS(east.pointsWA@coords)
```

```
##      D_lon M_lon      S_lon D_lat M_lat      S_lat
## [1,]  -137      0  1.982230    58    42 19.138232
## [2,]  -136     59 10.101336    58    41 41.137498
## [3,]  -136     30 46.012348    58    36 15.075222
## [4,]  -136     30 39.896955    58    33 52.874796
## [5,]  -136     29 46.558555    58    32 22.962200
## [6,]  -136     29 19.100840    58    31 18.612124
## [7,]  -136     27 59.582861    58    29 10.186811
## [8,]  -136     16 52.005486    58    28 51.477416
## [9,]  -136     20 29.759180    58    26 52.839442
## [10,] -136     19 14.322039    58    26 14.821621
## [11,] -136     18 49.186307    58    26  2.146251
## [12,] -136     16 44.983759    58    25 37.537001
## [13,] -136     15 29.657833    58    24 59.482249
## [14,] -136     15  4.559135    58    24 46.794578
```

7.2 East Arm of Glacier Bay

```
## Calculate coordinates in NAD27 UTM 8N
```

```
## Define spatial extent of East arm
```

```
eea=extent(414800,  
           xmax,  
           6513000,  
           ymax)
```

```
EA=crop(lambda2017,eea)
```

```
## Remove portion of East Arm
```

```
EA[60:113,1:39]=NA
```

```
## Plot the results
```

```
plot(EA)
```

```
## Obtain coordinates
```

```
EA.coords.tmp=matrix(,3)
```

```
for(i in 1:dim(EA)[1]){
```

```
  if(length(which(EA[i,]==30.9999))!=0){
```

```
    x.start=head(which(EA[i,]==30.9999),1)*400+e@xmin-200
```

```
    x.end=tail(which(EA[i,]==30.9999),1)*400+e@xmin-200
```

```
    y=e@ymax-i*400+200
```

```
    EA.coords.tmp=rbind(EA.coords.tmp,c(x.start,x.end,y))
```

```
  }
```

```
}
```

```
## Remove first row of NAs and name columns
```

```
EA.coords=EA.coords.tmp[-1,]
```

```
colnames(EA.coords)=c("lon.west","lon.east","lat")
```

```
## Output coordinates in NAD27 UTM ZONE 8N
```

```
EA.coords
```

```
##      lon.west lon.east   lat  
## [1,]   386200   389400 6510400  
## [2,]   389000   398200 6508800  
## [3,]   390200   398200 6507600  
## [4,]   385000   401800 6496800  
## [5,]   393400   404600 6492400  
## [6,]   400600   419800 6489600  
## [7,]   401000   416200 6487600  
## [8,]   400200   406200 6483600
```

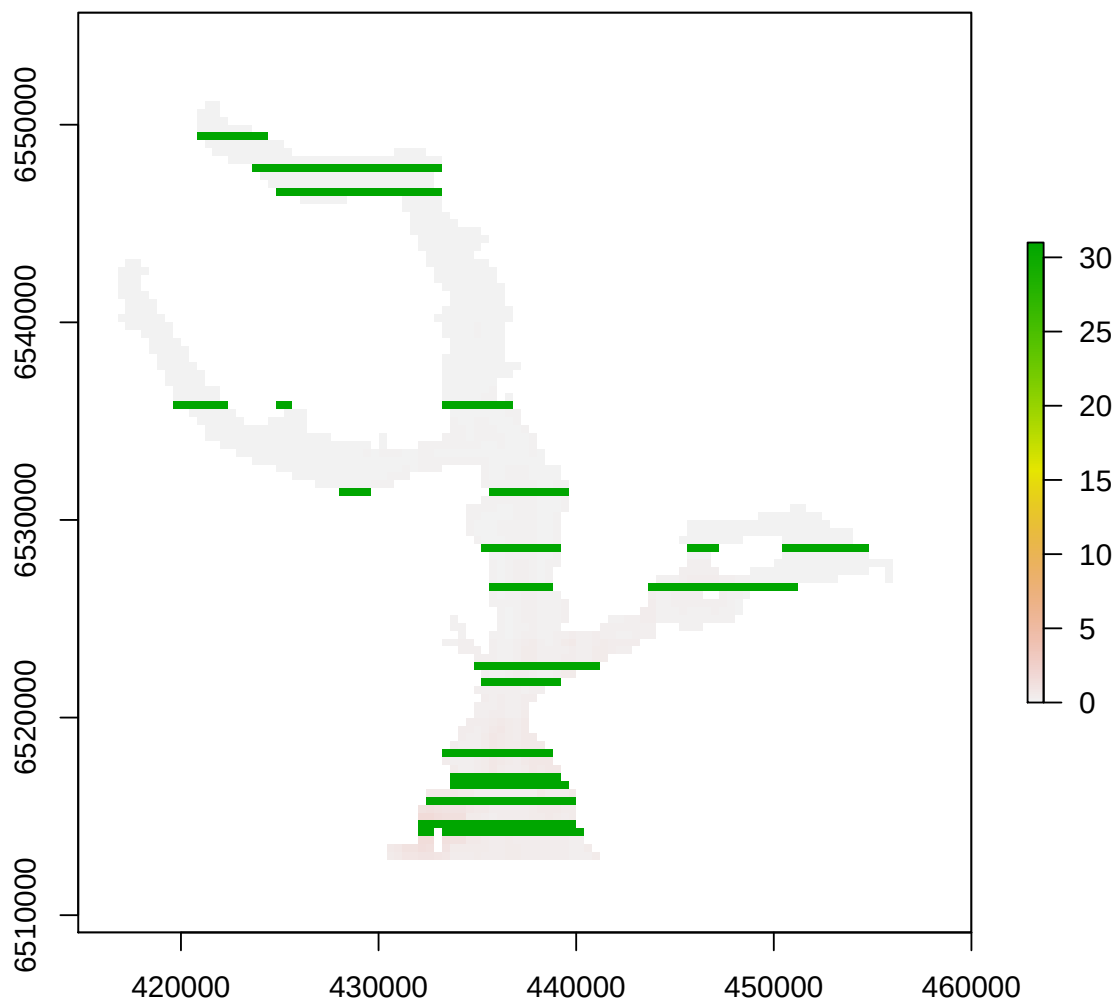


Figure 3: East Arm transects

```
## [9,] 400600 404200 6482800
## [10,] 398600 403800 6479200
## [11,] 399000 404200 6478000
## [12,] 399000 404600 6477600
## [13,] 397800 405000 6476800
## [14,] 397400 405000 6475600
## [15,] 397400 405400 6475200
```

7.2.1 Coordinate conversion

To convert coordinates from NAD27 UTM Zone 8N to either Decimal Degrees, or Degrees Minutes Seconds, we can use the following script. Note that `EA.coords` has three columns: `lon.west`, `lon.east`, and `lat`. The coordinates `lon.west` and `lon.east` correspond to the the end-points of transects on the west and east side of Glacier Bay, respectively. The latitude are similar, but not exact due to transformations among coordinate systems. ALL COORDINATES ARE FOR THE EAST ARM OF GLACIER BAY. Below `east` and `west` refer to the east and west side of the East Arm.

```
## Convert to Decimal Degrees
west.pointsEA=data.frame(lon=EA.coords[,1],lat=EA.coords[,3])
east.pointsEA=data.frame(lon=EA.coords[,2],lat=EA.coords[,3])
coordinates(west.pointsEA)=c("lon", "lat")
proj4string(west.pointsEA)=
  CRS("+proj=utm +zone=8 +ellps=clrk66 +datum=NAD27 +units=m +no_defs") #26708
coordinates(east.pointsEA)=c("lon", "lat")
proj4string(east.pointsEA)=
  CRS("+proj=utm +zone=8 +ellps=clrk66 +datum=NAD27 +units=m +no_defs") #26708
CRS.new=CRS("+proj=longlat +datum=WGS84")
west.pointsEA=spTransform(west.pointsEA,CRS.new)
east.pointsEA=spTransform(east.pointsEA,CRS.new)
west.pointsEA@coords

##          lon      lat
## [1,] -136.9669 58.72021
## [2,] -136.9178 58.70658
## [3,] -136.8965 58.69611
## [4,] -136.9807 58.59784
## [5,] -136.8341 58.56048
## [6,] -136.7092 58.53706
## [7,] -136.7015 58.51919
## [8,] -136.7134 58.48310
## [9,] -136.7062 58.47601
## [10,] -136.7389 58.44324
## [11,] -136.7315 58.43256
```

```

## [12,] -136.7314 58.42897
## [13,] -136.7516 58.42151
## [14,] -136.7579 58.41064
## [15,] -136.7577 58.40705

east.pointsEA@coords

##          lon      lat
## [1,] -136.9116 58.72104
## [2,] -136.7590 58.70884
## [3,] -136.7585 58.69807
## [4,] -136.6917 58.60196
## [5,] -136.6417 58.56309
## [6,] -136.3795 58.54102
## [7,] -136.4406 58.52239
## [8,] -136.6106 58.48444
## [9,] -136.6445 58.47682
## [10,] -136.6499 58.44441
## [11,] -136.6425 58.43373
## [12,] -136.6355 58.43022
## [13,] -136.6283 58.42313
## [14,] -136.6278 58.41236
## [15,] -136.6208 58.40885

## Convert western lat long DD points of East Arm to DMS
convert_DD_to_DMS(west.pointsEA@coords)

##      D_lon M_lon      S_lon D_lat M_lat      S_lat
## [1,] -136    58  0.6935201    58    43 12.743053
## [2,] -136    55  3.9009587    58    42 23.670405
## [3,] -136    53 47.2728036    58    41 46.001639
## [4,] -136    58 50.3518882    58    35 52.207843
## [5,] -136    50  2.8491558    58    33 37.745776
## [6,] -136    42 33.1020727    58    32 13.399520
## [7,] -136    42  5.2528870    58    31  9.097277
## [8,] -136    42 48.3938843    58    28 59.177064
## [9,] -136    42 22.4563753    58    28 33.652105
## [10,] -136    44 20.1370730    58    26 35.657086
## [11,] -136    43 53.5780362    58    25 57.210352
## [12,] -136    43 52.9450614    58    25 44.283648
## [13,] -136    45  5.6111700    58    25 17.427216
## [14,] -136    45 28.3276111    58    24 38.311056
## [15,] -136    45 27.6854751    58    24 25.384500

## Convert eastern longitude points of East Arm to DMS
convert_DD_to_DMS(east.pointsEA@coords)

```

##		D_lon	M_lon	S_lon	D_lat	M_lat	S_lat
##	[1,]	-136	54	41.887574	58	43	15.734688
##	[2,]	-136	45	32.508625	58	42	31.823617
##	[3,]	-136	45	30.555439	58	41	53.048071
##	[4,]	-136	41	30.109912	58	36	7.048921
##	[5,]	-136	38	30.059196	58	33	47.106057
##	[6,]	-136	22	46.200577	58	32	27.674251
##	[7,]	-136	26	26.102270	58	31	20.587919
##	[8,]	-136	36	38.065732	58	29	3.968785
##	[9,]	-136	38	40.306281	58	28	36.549483
##	[10,]	-136	38	59.555790	58	26	39.887669
##	[11,]	-136	38	33.093409	58	26	1.421863
##	[12,]	-136	38	7.842112	58	25	48.809208
##	[13,]	-136	37	42.001099	58	25	23.267657
##	[14,]	-136	37	40.217655	58	24	44.485185
##	[15,]	-136	37	14.988146	58	24	31.869691

7.3 Lower Glacier Bay

```
## Calculate coordinates in NAD27 UTM 8N
## Define spatial extent of Lower Glacier Bay
```

```
elb=extent(xmin,
           xmax,
           ymin,
           6513000)
LB=crop(lambda2017,elb)
```

```
## Remove portion of Glacier Bay
LB[60:113,1:39]=NA
```

```
## Plot the results
plot(LB)
```

```
## Obtain the coordinates
LB.coords.tmp=matrix(,3)
for(i in 1:dim(LB)[1]){
  if(length(which(LB[i,]==30.9999))!=0){
    x.start=head(which(LB[i,]==30.9999),1)*400+e@xmin-200
    x.end=tail(which(LB[i,]==30.9999),1)*400+e@xmin-200
    y=e@ymax-i*400+200
    LB.coords.tmp=rbind(LB.coords.tmp,c(x.start,x.end,y))
  }
}
```

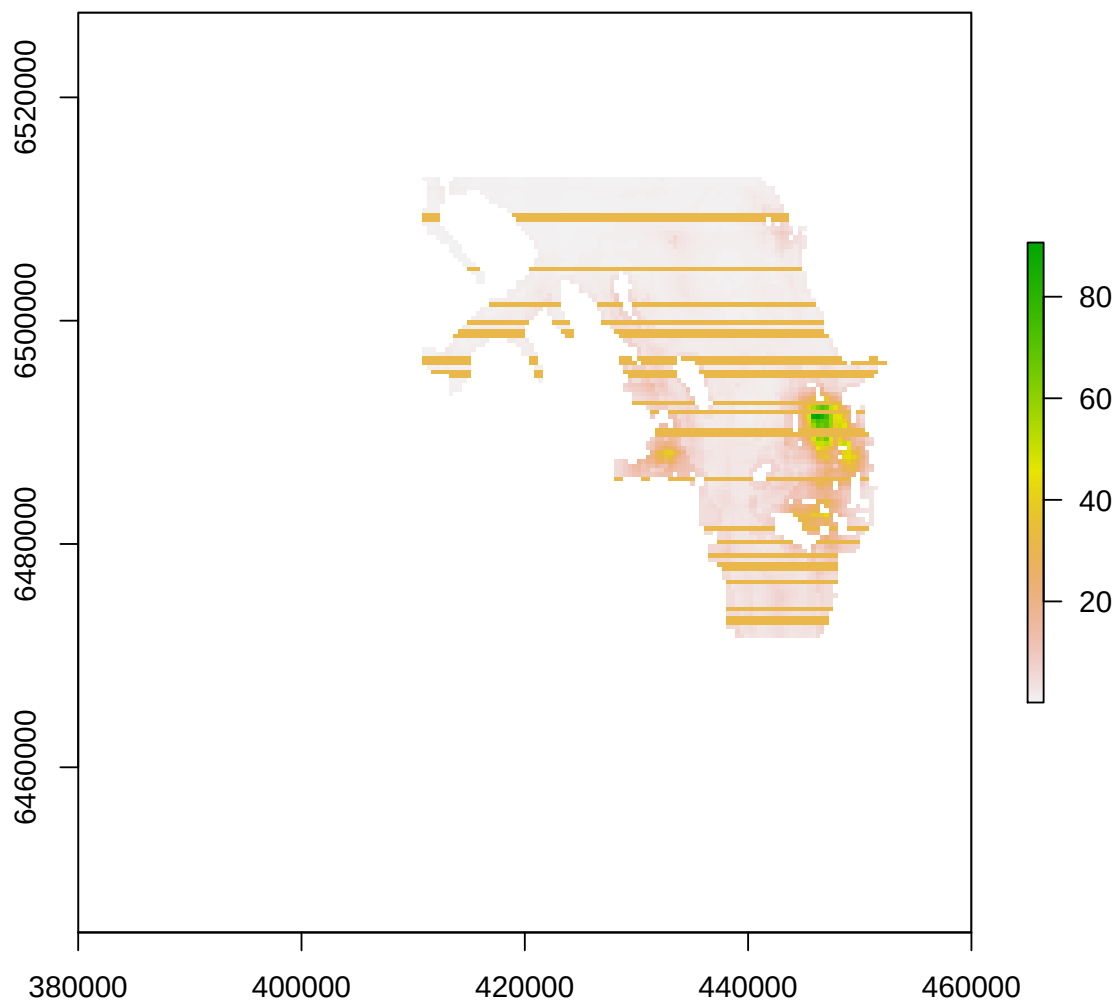


Figure 4: Lower Glacier Bay transects

```

    }
}

## Remove first row of NAs and name columns
LB.coords=LB.coords.tmp[-1,]
colnames(LB.coords)=c("lon.west", "lon.east", "lat")

## Output coordinates in NAD27 UTM ZONE 8N
LB.coords

##      lon.west lon.east   lat
## [1,]    411000    443400 6509600
## [2,]    411000    443400 6509200
## [3,]    415000    444600 6504800
## [4,]    417000    445800 6501600
## [5,]    415000    446600 6500000
## [6,]    414200    446600 6499200
## [7,]    413800    447000 6498800
## [8,]    411000    451400 6496800
## [9,]    411000    452200 6496400
## [10,]   411800    451400 6495600
## [11,]   413400    451000 6495200
## [12,]   429800    448200 6492800
## [13,]   431400    450200 6492000
## [14,]   431800    450200 6490400
## [15,]   431800    450600 6490000
## [16,]   428200    450600 6486000
## [17,]   436200    450600 6481600
## [18,]   437400    449800 6480400
## [19,]   436600    447800 6479200
## [20,]   437400    447800 6478400
## [21,]   437800    447800 6478000
## [22,]   438200    447800 6476800
## [23,]   438200    447400 6474400
## [24,]   438200    447000 6473600
## [25,]   438200    447000 6473200

```

7.3.1 Coordinate conversion

To convert coordinates from NAD27 UTM Zone 8N to either Decimal Degrees, or Degrees Minutes Seconds, we can use the following script. Note that EA.coords has three columns: `lon.west`, `lon.east`, and `lat`. The coordinates `lon.west` and `lon.east` correspond to the end-points of transects on the west and east side of Glacier Bay, respectively. The latitudes are similar, but not exact due to transformations among coordinate systems. ALL

COORDINATES ARE FOR LOWER GLACIER BAY. Below `east` and `west` refer to the east and west side of Lower Glacier Bay.

```
## Convert to Decimal Degrees
west.pointsLB=data.frame(lon=LB.coords[,1],lat=LB.coords[,3])
east.pointsLB=data.frame(lon=LB.coords[,2],lat=LB.coords[,3])
coordinates(west.pointsLB)=c("lon", "lat")
proj4string(west.pointsLB)=
  CRS("+proj=utm +zone=8 +ellps=clrk66 +datum=NAD27 +units=m +no_defs") #26708
coordinates(east.pointsLB)=c("lon", "lat")
proj4string(east.pointsLB)=
  CRS("+proj=utm +zone=8 +ellps=clrk66 +datum=NAD27 +units=m +no_defs") #26708
CRS.new=CRS("+proj=longlat +datum=WGS84")
west.pointsLB=spTransform(west.pointsLB,CRS.new)
east.pointsLB=spTransform(east.pointsLB,CRS.new)
west.pointsLB@coords

##          lon      lat
## [1,] -136.5385 58.71885
## [2,] -136.5383 58.71526
## [3,] -136.4676 58.67656
## [4,] -136.4320 58.64823
## [5,] -136.4658 58.63347
## [6,] -136.4793 58.62613
## [7,] -136.4860 58.62246
## [8,] -136.5334 58.60394
## [9,] -136.5333 58.60035
## [10,] -136.5192 58.59333
## [11,] -136.4915 58.59007
## [12,] -136.2088 58.57148
## [13,] -136.1810 58.56456
## [14,] -136.1737 58.55025
## [15,] -136.1735 58.54666
## [16,] -136.2341 58.51017
## [17,] -136.0956 58.47191
## [18,] -136.0747 58.46131
## [19,] -136.0881 58.45042
## [20,] -136.0742 58.44335
## [21,] -136.0672 58.43982
## [22,] -136.0600 58.42910
## [23,] -136.0594 58.40755
## [24,] -136.0592 58.40036
## [25,] -136.0591 58.39677

east.pointsLB@coords
```

```

##          lon      lat
## [1,] -135.9792 58.72432
## [2,] -135.9791 58.72073
## [3,] -135.9573 58.68138
## [4,] -135.9359 58.65280
## [5,] -135.9217 58.63853
## [6,] -135.9215 58.63135
## [7,] -135.9145 58.62780
## [8,] -135.8383 58.61036
## [9,] -135.8245 58.60686
## [10,] -135.8381 58.59959
## [11,] -135.8449 58.59595
## [12,] -135.8925 58.57407
## [13,] -135.8579 58.56712
## [14,] -135.8576 58.55276
## [15,] -135.8506 58.54921
## [16,] -135.8497 58.51329
## [17,] -135.8488 58.47378
## [18,] -135.8622 58.46292
## [19,] -135.8962 58.45190
## [20,] -135.8961 58.44472
## [21,] -135.8960 58.44113
## [22,] -135.8957 58.43035
## [23,] -135.9020 58.40875
## [24,] -135.9086 58.40152
## [25,] -135.9085 58.39793

## Convert western lat long DD points of East Arm to DMS
convert_DD_to_DMS(west.pointsLB@coords)

##          D_lon M_lon      S_lon D_lat M_lat      S_lat
## [1,]   -136     32 18.5487428    58   43  7.8450713
## [2,]   -136     32 17.9769029    58   42 54.9186968
## [3,]   -136     28  3.4638010    58   40 35.6274228
## [4,]   -136     25 55.0995936    58   38 53.6114633
## [5,]   -136     27 56.9353828    58   38  0.5042373
## [6,]   -136     28 45.4209361    58   37 34.0819715
## [7,]   -136     29  9.6579625    58   37 20.8686831
## [8,]   -136     32  0.3354885    58   36 14.1944335
## [9,]   -136     31 59.7712268    58   36  1.2673064
## [10,] -136     31  9.1097643    58   35 36.0012596
## [11,] -136     29 29.4990528    58   35 24.2347608
## [12,] -136     12 31.5537315    58   34 17.3406611
## [13,] -136     10 51.6834605    58   33 52.4045048
## [14,] -136     10 25.2075771    58   33  0.9154174

```

```
## [15,] -136    10 24.7751271    58    32 47.9863535
## [16,] -136    14  2.8195563    58    30 36.6078026
## [17,] -136     5 44.2369754    58    28 18.8731934
## [18,] -136     4 29.0078228    58    27 40.7103984
## [19,] -136     5 17.1592510    58    27  1.5035830
## [20,] -136     4 27.0365639    58    26 36.0600260
## [21,] -136     4  1.9825009    58    26 23.3361876
## [22,] -136     3 36.1555048    58    25 44.7506481
## [23,] -136     3 33.8243386    58    24 27.1694781
## [24,] -136     3 33.0480123    58    24  1.3090267
## [25,] -136     3 32.6599857    58    23 48.3787894
```

```
## Convert eastern longitude points of East Arm to DMS
convert_DD_to_DMS(east.pointsLB@coords)
```

```
##      D_lon M_lon      S_lon D_lat M_lat      S_lat
## [1,] -135    58 45.215617    58    43 27.546200
## [2,] -135    58 44.851802    58    43 14.616730
## [3,] -135    57 26.366476    58    40 52.952868
## [4,] -135    56  9.101863    58    39 10.063692
## [5,] -135    55 18.118570    58    38 18.703110
## [6,] -135    55 17.436858    58    37 52.843355
## [7,] -135    54 52.304749    58    37 40.090708
## [8,] -135    50 18.018065    58    36 37.304626
## [9,] -135    49 28.145194    58    36 24.696432
## [10,] -135    50 17.085728    58    35 58.513495
## [11,] -135    50 41.548317    58    35 45.420126
## [12,] -135    53 32.966423    58    34 26.662435
## [13,] -135    51 28.550872    58    34  1.647571
## [14,] -135    51 27.280903    58    33  9.926067
## [15,] -135    51  2.223747    58    32 57.160863
## [16,] -135    50 59.080460    58    30 47.856136
## [17,] -135    50 55.631390    58    28 25.617923
## [18,] -135    51 44.049597    58    27 46.495235
## [19,] -135    53 46.446450    58    27  6.855593
## [20,] -135    53 45.785992    58    26 40.994160
## [21,] -135    53 45.455881    58    26 28.063432
## [22,] -135    53 44.466019    58    25 49.271203
## [23,] -135    54  7.128115    58    24 31.513494
## [24,] -135    54 31.099283    58    24  5.477614
## [25,] -135    54 30.765173    58    23 52.546842
```