

SOP 09: Estimating Occupancy and Abundance

Version 1.0

Contents

1	Summary	3
2	Setting up the R workspace	3
3	Preliminary Glacier Bay settings	4
4	Boundary and covariate information	5
5	Pre-2016 data	14
6	New data	21
7	MCMC settings and hyperparameters	22
8	Running the MCMC algorithm	23
9	Results	24
9.1	Model diagnostics	24
9.1.1	Tuning parameters	24
9.1.2	Trace plots	25
9.2	Model fit	26
9.2.1	Marginal posterior distributions	26
9.2.2	Results table	29
10	Posterior predictive distribution	31

List of Figures

1	Bathymetry data for Glacier Bay National Park	7
2	Depth covariate	8
3	Distance-to-shore covariate	10
4	Bottom slope covariate	11
5	Shoreline complexity covariate	12
6	Area of inference	13
7	Design based survey data	17
8	ISU data	19
9	Adaptive tuner	25
10	Trace-plots	27
11	Density plots	29

List of Tables

1 Results table 33

1 Summary

This SOP describes how to use available software to estimate spatio-temporal occupancy and abundance of 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 Williams et al. (2017c). Data for estimating spatio-temporal occupancy and abundance consist of eight design-based surveys, in which data were collected during 1999–2004, 2006, and 2012, and seven distributional surveys in which data were collected in 1993, 1995–1998, 2005, 2009–2010, and future aerial photograph data to be collected beginning in 2017. The design-based surveys and distributional surveys were conducted by the U.S. Geological Survey. 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.

```
rm(list=ls())

required.packages=c("coda",
                    "devtools",
                    "fBasics",
                    "fields",
                    "gridExtra",
                    "inline",
                    "raster",
                    "rasterVis",
                    "RColorBrewer",
                    "RcppArmadillo",
                    "roxygen2"
                    )

## install.packages(required.packages)
lapply(required.packages,library,character.only=TRUE)

## Loading required package:  timeDate
## Loading required package:  timeSeries
##
## Rmetrics Package fBasics
## Analysing Markets and calculating Basic Statistics
## Copyright (C) 2005-2014 Rmetrics Association Zurich
## Educational Software for Financial Engineering and Computational Science
## Rmetrics is free software and comes with ABSOLUTELY NO WARRANTY.
## https://www.rmetrics.org -- Mail to:  info@rmetrics.org
```

```

## Loading required package: spam
## Loading required package: grid
## Spam version 1.4-0 (2016-08-29) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
##
## Attaching package: 'spam'
## The following objects are masked from 'package:base':
##
##   backsolve, forwardsolve
## Loading required package: maps
## Loading required package: sp
## Loading required package: lattice
## Loading required package: latticeExtra
## Loading required package: RColorBrewer

## Local stoa package
setwd("~/Dropbox/Post-Doc/RFiles/EcologicalDiffusion/package")
install("stoa")

## Installing stoa
## '/Library/Frameworks/R.framework/Resources/bin/R' -no-site-file \
## -no-environ -no-save -no-restore -quiet CMD INSTALL \
## '/Users/pwill/Dropbox/Post-Doc/RFiles/EcologicalDiffusion/package/stoa' \
## -library='/Library/Frameworks/R.framework/Versions/3.3/Resources/library'
##
## -install-tests
##

library(stoa)

```

3 Preliminary Glacier Bay settings

The domain of inference is set by choosing minimum and maximum coordinates for latitude and longitude using the NAD27 datum, and UTM 8N projection. Glacier bay is partitioned into 400×400m grid cells. The total number of cells are represented with `q`, `cell` is raster object containing the unique identified of each cell, `d` is the epicenter of the sea otter colonization, `dt` are the number of discrete time steps per year, `timesteps` are the total number of time steps for the entire study period, `keep` is vector identifying which time steps to keep for inference, and `us.fact` is the upscale factor for homogenization. Note that the dimension of the study area, `x` and `y`, must be compatible with the upscale factor (the dimensions must be divisible by `us.fact` to a natural number).

```

inference.year=2017
xmin=380000 ## 404000
xmax=460000 ## 460000
ymin=6460000 ## 6460000
ymax=6552000 ## 6528000
x=(xmax-xmin)/400
y=(ymax-ymin)/400
cell=raster(,nrows=y,ncols=x,xmn=xmin,ymx=xmax,
            ymn=ymin,ymx=ymax,crs=NA)
q=x*y
cell[1:q]=1:q
d=c(442000,6465000)
dt=1/100
timesteps=1/dt*length(1993:inference.year)
keep=seq(1,timesteps,timesteps/length(1993:inference.year))
us.fact=10

## Check to see if size of domain is compatible with upscale factor
if((x/us.fact)%1==0&(y/us.fact)%1==0){
  "Compatible"
}else{"Not compatible"}

## [1] "Compatible"

```

4 Boundary and covariate information

Boundary layers are required for restricting movement of sea otters to marine environments, and for limiting inference to Glacier Bay National Park. The covariates used in the analysis include depth, distance to shore, slope of the ocean floor, and shoreline complexity. The boundary layers and the covariates are obtained using bathymetry data. The boundary layers and covariates are derived below. **Boundary** is a raster object that assigns marine environments a 1, and non-marine environments a 0, **BoundaryNA** is a raster object that assigns marine environments a 1, and non-marine environments a NA, **Boundary.us** is a raster object that assigns marine environments a 1, and non-marine environments a 0 with resolution equal to the up-scale factor for homogenization, and **BoundaryInf** restricts inference to Glacier Bay National Park.

```

###
### load covariate information
###

load("~/Dropbox/Post-Doc/RFiles/lutris/data/bath.raster.RData")
bath.tmp=crop(bath.raster, extent(xmin, xmax, ymin, ymax))

```

```

bath.depth=aggregate(bath.tmp, c(16, 16),na.rm=TRUE)
bath.dist=aggregate(bath.tmp, c(16, 16),na.rm=FALSE) #y, x
missing.x=c(423934.8,440637.5,434906.5)
missing.y=c(6506685.3,6491425,6515499)
for(i in 1:length(missing.x)){
  bath.dist[floor((ymax-missing.y[i])/400),
            floor((missing.x[i]-xmin)/400)]=NA
}

plot(bath.dist)

```

```

## Create Down-Scaled Boundary Layer (1 = water; 0 = land; NA set to 0)
Boundary=raster(,nrows=y,ncols=x,
  xmn=xmin,xmx=xmax,ymn=ymin,ymx=ymax,
  crs=NA)
Boundary[is.na(bath.depth)]=0
Boundary[bath.depth<0]=1
Boundary[bath.depth>=0]=0
BoundaryNA=Boundary
BoundaryNA[Boundary==0]=NA

## up-scaled boundary layer surrounded by zeros
Boundary.us=aggregate(Boundary,fact=us.fact,na.rm=TRUE,fun=max)
Boundary.us[1,]=0
Boundary.us[,1]=0
Boundary.us[dim(Boundary.us)[1],]=0
Boundary.us[, dim(Boundary.us)[2]]=0

```

```

## Depth
DepthCov=raster(,nrows=y,ncols=x,
  xmn=xmin,xmx=xmax,ymn=ymin,ymx=ymax,
  crs=NA)
depth=40
DepthCov[bath.depth< -depth]=0
DepthCov[bath.depth> -depth]=1
DepthCov[is.na(DepthCov)]=0
DepthCov=DepthCov*Boundary
plot(DepthCov)

```

```

## Distance to shore
BoundaryDist=raster(,nrows=y,ncols=x,

```

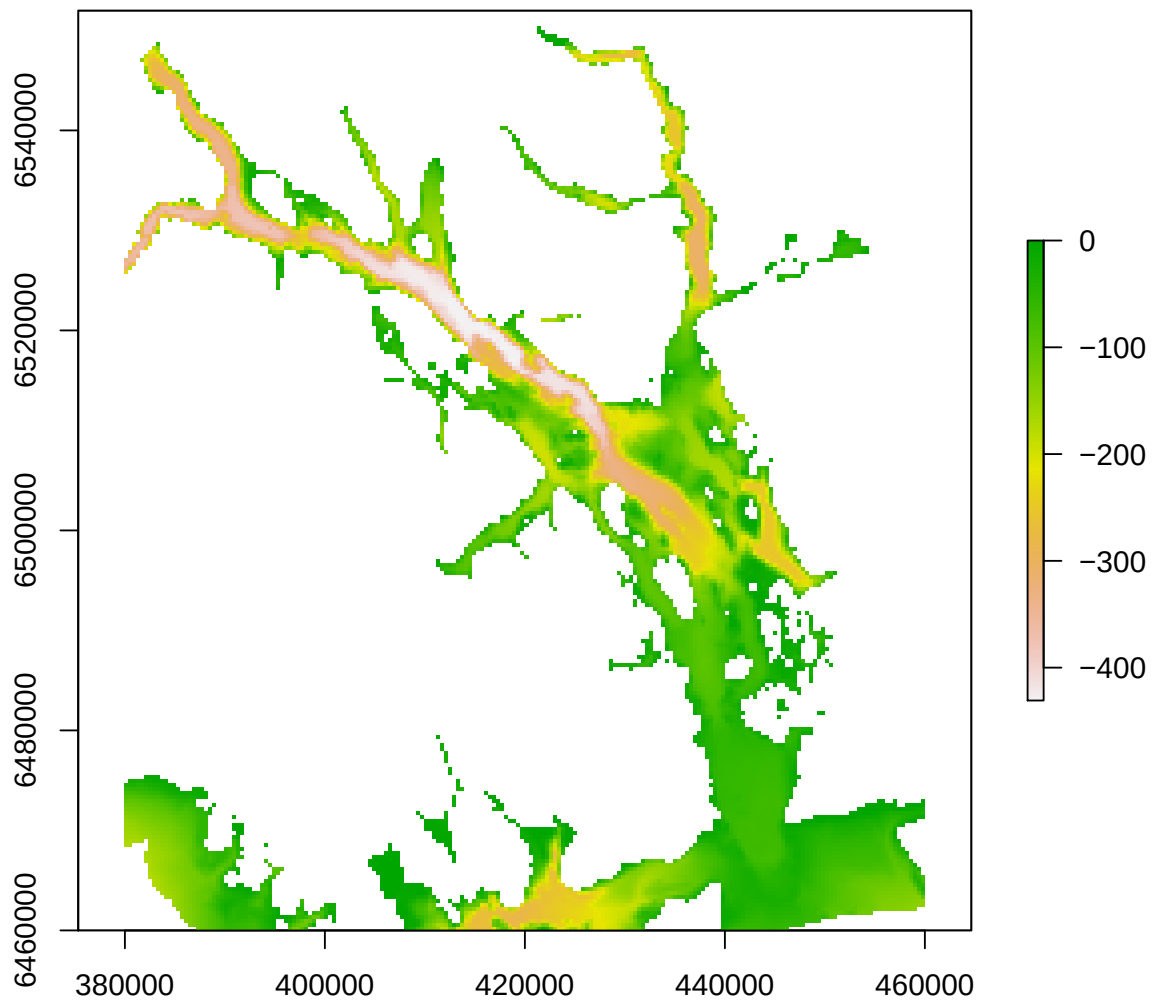


Figure 1: Bathymetry data for Glacier Bay National Park

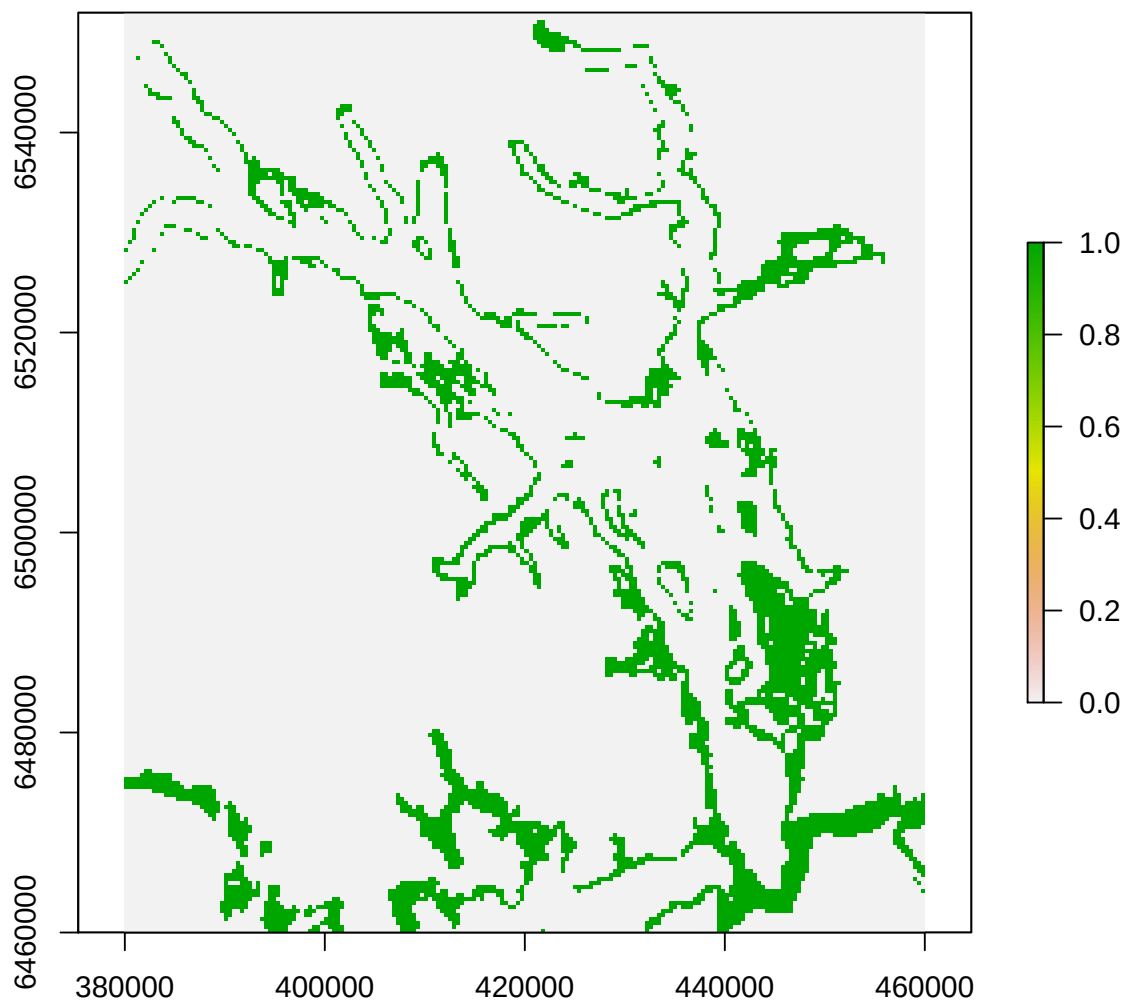


Figure 2: Depth covariate

```

    xmn=xmin,xmx=xmax,ymn=ymin,ymx=ymax,
    crs=NA)
BoundaryDist[is.na(bath.dist)]=0
BoundaryDist[bath.dist<0]=1
BoundaryDist[bath.dist>=0]=0
DistCov.tmp1=BoundaryDist
DistCov.tmp1[BoundaryDist==1]=NA
DistCov.tmp2=distance(DistCov.tmp1)
DistCov.tmp2[DistCov.tmp2==0]=NA
DistCov=scale(DistCov.tmp2,center=TRUE,scale=TRUE)
DistCov[is.na(DistCov[])]=min(DistCov[],na.rm=TRUE)
plot(DistCov)

```

```

## Bottom slope
SlopeCov.tmp=terrain(bath.depth, opt='slope', unit='degrees', neighbors=8)
SlopeCov.tmp[is.na(SlopeCov.tmp)]=0
SlopeCov=scale(SlopeCov.tmp, center=TRUE, scale=TRUE)
plot(SlopeCov)

```

```

## Shoreline complexity
ShoreCov.tmp=boundaries(DistCov.tmp1,type='inner')
ShoreCov.tmp[is.na(ShoreCov.tmp)]=0
ShoreCov=focal(ShoreCov.tmp,w=matrix(1,nr=11,nc=11))*Boundary
ShoreCov[is.na(ShoreCov)]=0
ShoreCov=scale(ShoreCov, center=TRUE, scale=TRUE)
plot(ShoreCov)

```

Compile covariate information for analysis.

```

X=cbind(1,DepthCov[],DistCov[],SlopeCov[],ShoreCov[])
W=matrix(1,nr=q,nc=1)

```

Inference area:

```

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

plot(BoundaryInf)

```

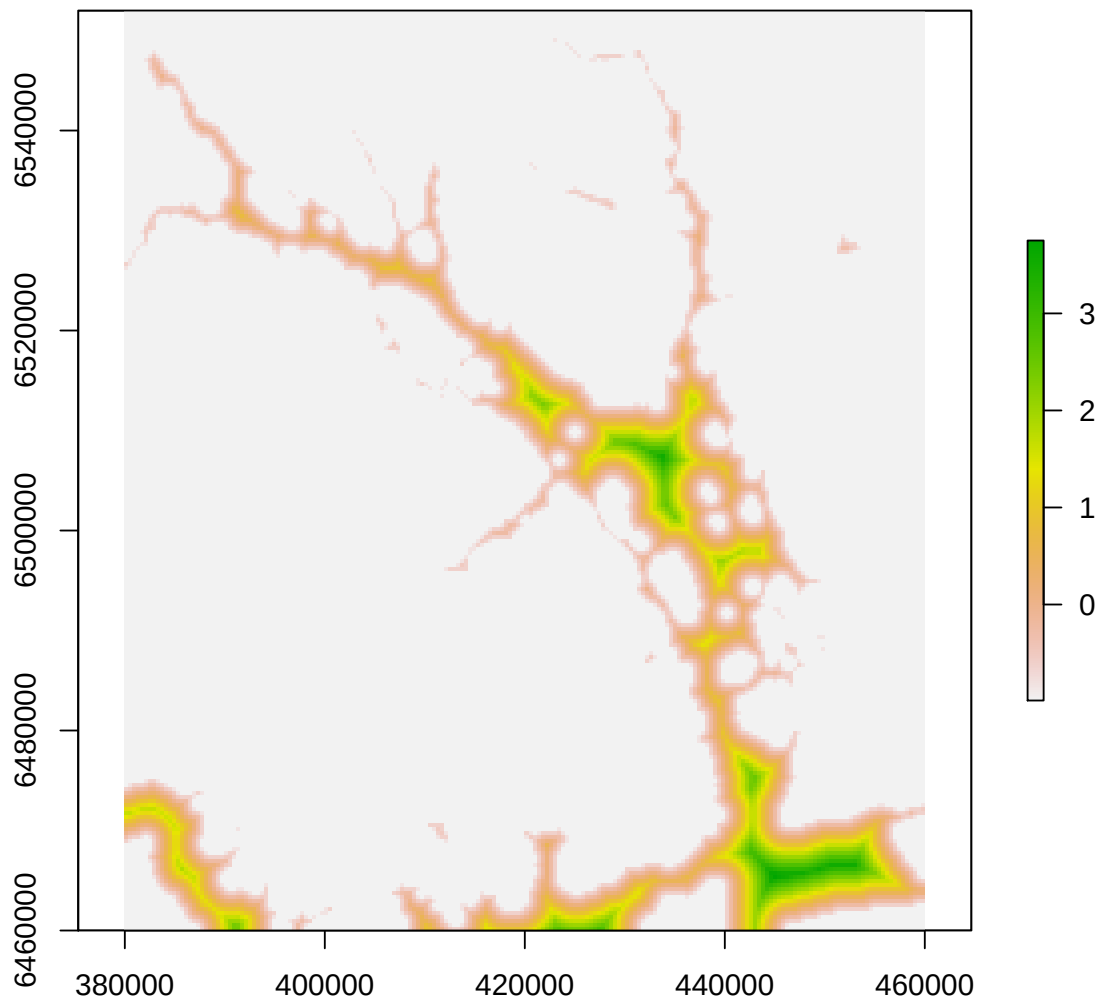


Figure 3: Distance-to-shore covariate

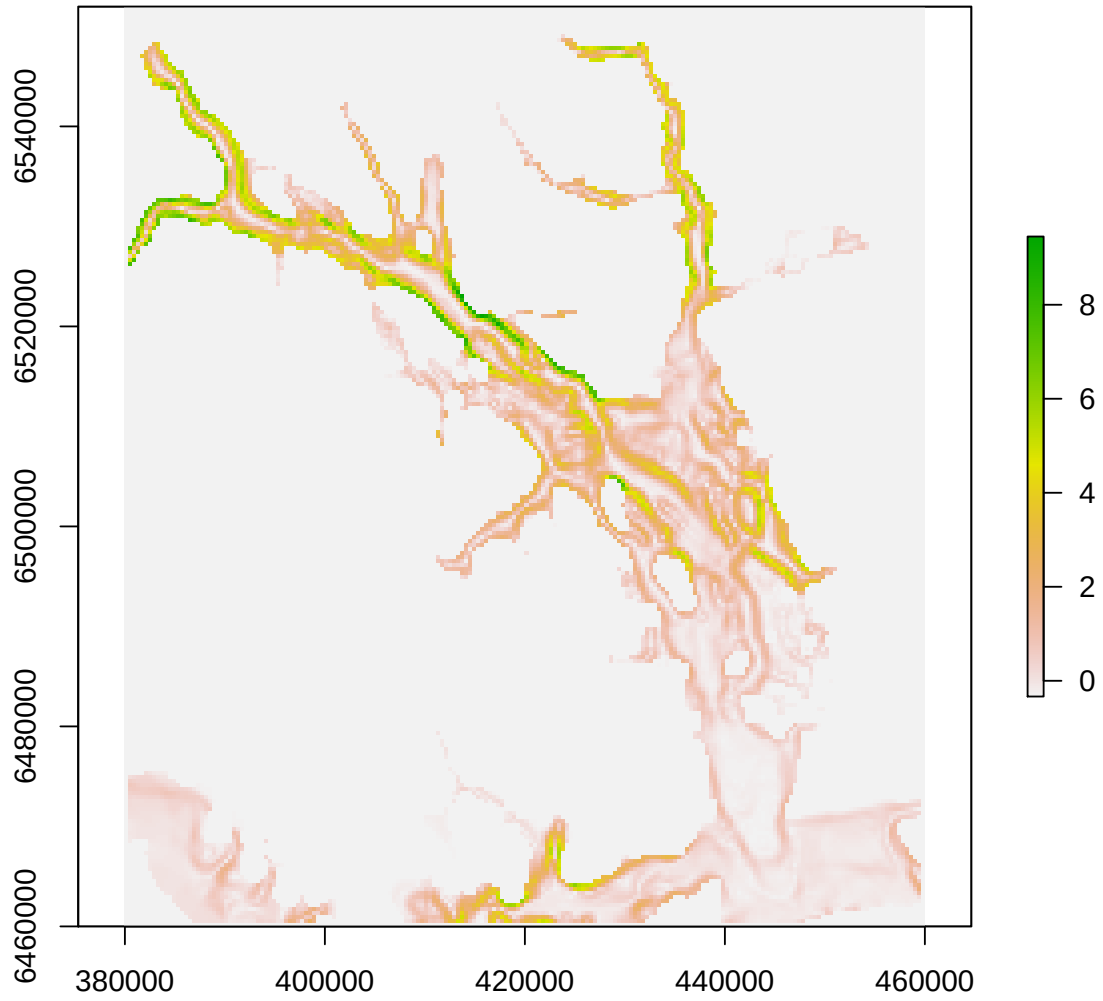


Figure 4: Bottom slope covariate

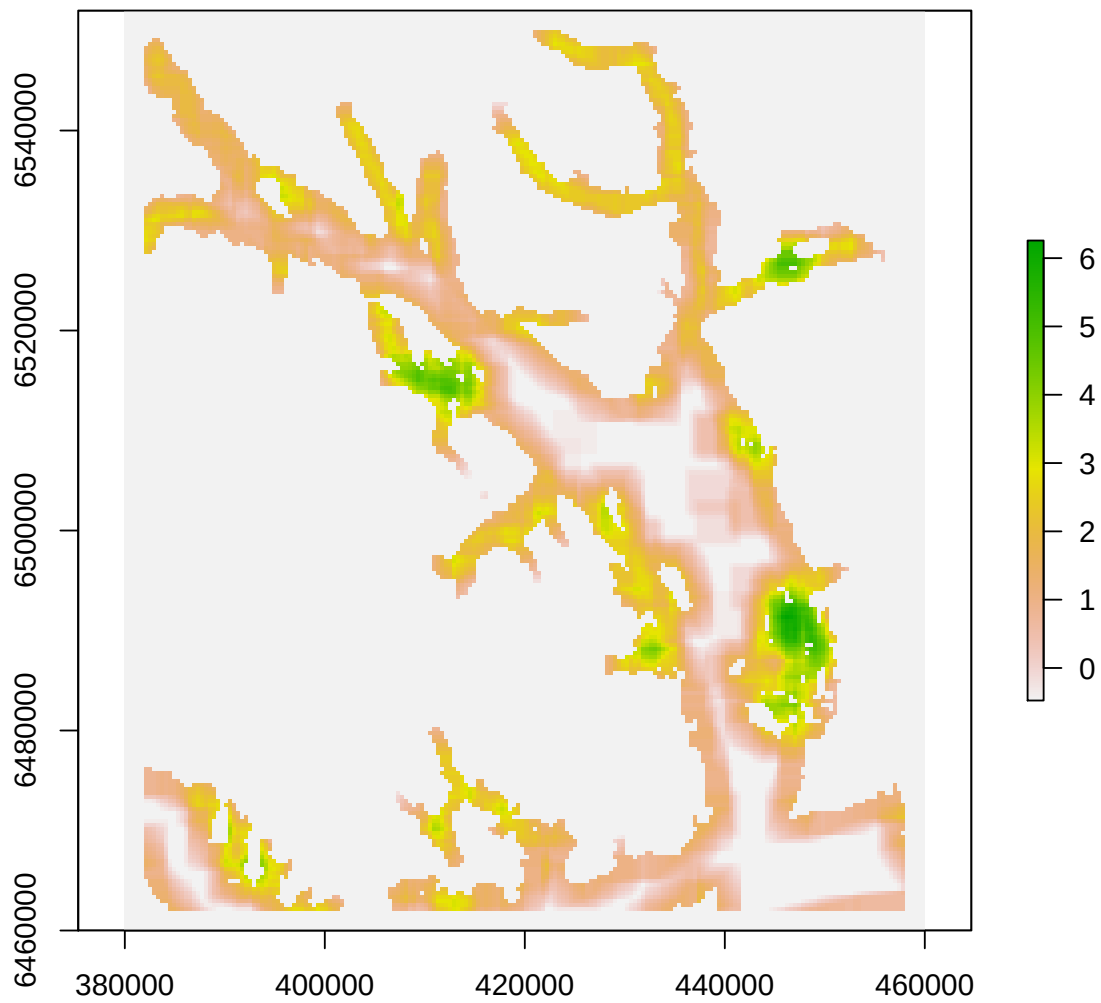


Figure 5: Shoreline complexity covariate

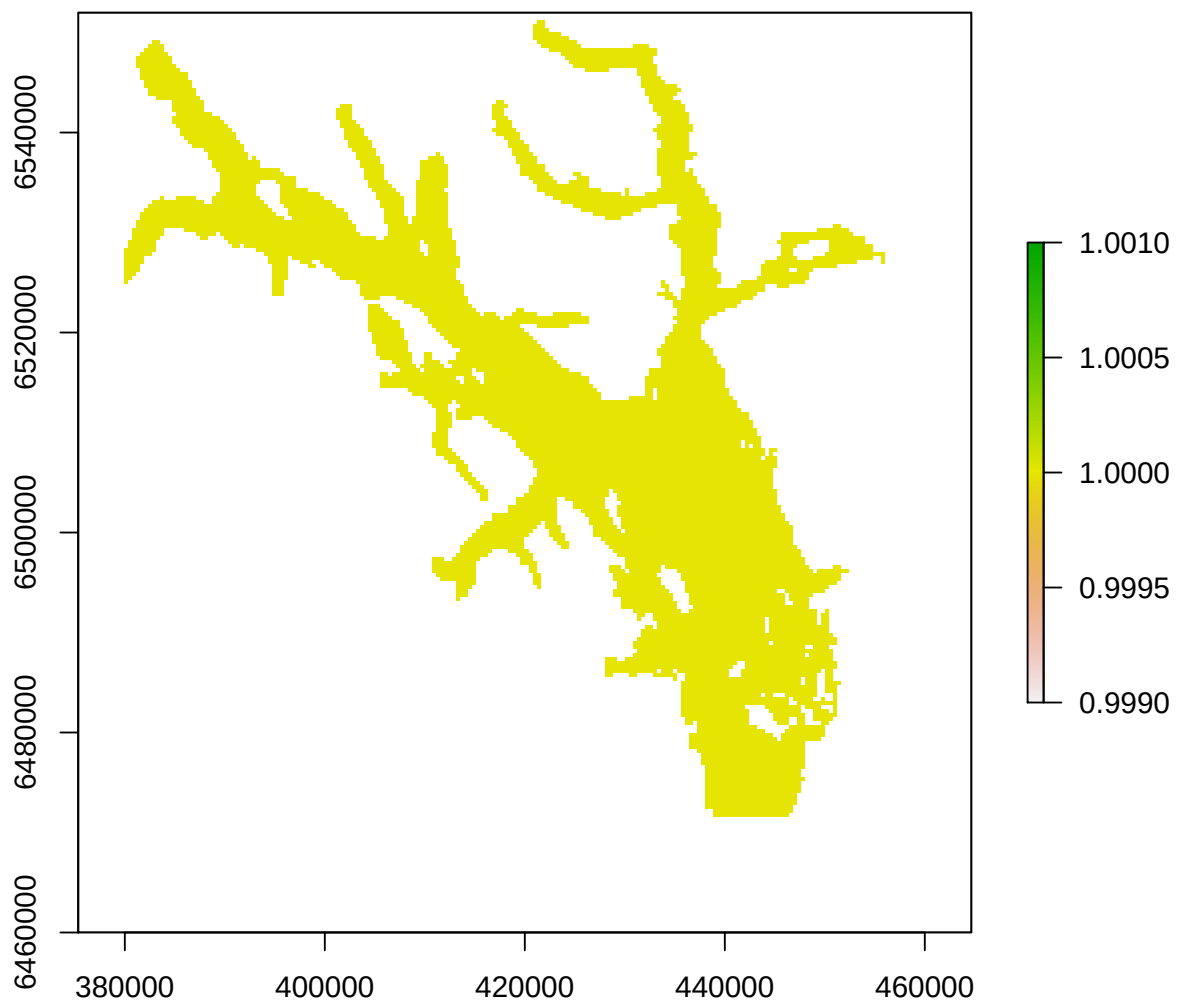


Figure 6: Area of inference

5 Pre-2016 data

Data from design-based surveys and distributional surveys collected by U.S. Geological Survey, between 1993 and 2012. Also collected during the design-based surveys were Intensive Sample Units (ISUs) to estimate detection probability.

```
###  
### Load data  
###  
  
data=read.csv("~/Dropbox/Post-Doc/OriginalDataFiles/DesignBasedData.csv")  
dataDistr=read.csv(paste("~/Dropbox/Post-Doc/OriginalDataFiles/",  
                          "DistributionalData.csv",sep=""))  
  
## Indicator for distributional data.  
ind.pred.tmp=1:dim(dataDistr)[1]  
  
## Use distributional data only when design-based data not available.  
ind.pred=ind.pred.tmp[dataDistr$year==2004|dataDistr$year==2006]  
dataDistr=dataDistr[-ind.pred,]  
ind=1993:2012  
  
## convert design-based data to raster  
data.r=raster(,nrows=y,ncols=x,xmn=xmin,  
              xmx=xmax,ymn=ymin,ymx=ymax,crs=0)  
data1.sub=subset(data,year=="1993")  
data1.sub=data1.sub[!is.na(data1.sub$GROUP_X),]  
data2.sub=subset(dataDistr,year=="1993")  
x_loc=c(data1.sub$Group_X,data2.sub$POINT_X)  
y_loc=c(data1.sub$Group_Y,data2.sub$POINT_Y)  
Counts.tmp=c(data1.sub$animals,data2.sub$animals)  
Counts.sub=rasterize(x=cbind(x_loc,y_loc),y=data.r,  
                     field=Counts.tmp,fun="sum",background=0)  
Counts=raster(,nrows=y,ncols=x,xmn=xmin,  
              xmx=xmax,ymn=ymin,ymx=ymax,crs=0)  
Counts[]=Counts.sub[]  
Counts=stack(mget(rep("Counts",20)))  
years=1994:2012  
ind=2  
for(t in years){  
  data1.sub=subset(data,year==t)  
  data1.sub=data1.sub[!is.na(data1.sub$GROUP_X),]  
  data2.sub=subset(dataDistr,year==t)  
  x_loc=c(data1.sub$GROUP_X,data2.sub$POINT_X)  
  y_loc=c(data1.sub$GROUP_Y,data2.sub$POINT_Y)
```

```

Counts.tmp=c(data1.sub$animals,data2.sub$animals)
if(length(x_loc)>0){
  Counts.sub=rasterize(x=cbind(x_loc,y_loc),
                      y=data.r,field=Counts.tmp,fun="sum",
                      background=0)

  Counts[[ind]][[]]=Counts.sub[]
}else{
  Counts[[ind]][[]]=rep(NA,q)
}
ind=ind+1
}

```

Create transect layer for flights based on start and end points. The function `rasterize` is slow, and these computations take a few minutes.

```

## Create transect layer
transect.sub=subset(data,year=="1999")
l=vector("list",nrow(transect.sub))
l.l=list()
begin.coord=data.frame(lon=transect.sub$west_long,
                       lat=transect.sub$west_lat)
end.coord=data.frame(lon=transect.sub$east_long,
                     lat=transect.sub$east_lat)
for(i in seq_along(l)){
  l[[i]]=Lines(list(Line(
    rbind(begin.coord[i,],
          end.coord[i,]))),as.character(i))
}
l.l[[1]]=l
transect.r=raster(,nrows=y,ncols=x,xmn=xmin,
                 xmx=xmax,ymn=ymin,ymx=ymax,crs=NA)
transect.sub=subset(data,year=="1999")
transect.sub.r=rasterize(x=SpatialLines(l),y=transect.r,
                       field=1,fun="last",background=NA)

transectAll=transect.sub.r
transectAll[]=transect.sub.r[]
transectAll[Counts[[1]][[]]>0]=1
transectAll=stack(mget(rep("transectAll",20)))
transectAll[[1]][[]]=rep(NA,q)
ind=2
for(t in years){
  transect.sub=subset(data,year==t)
  if(dim(transect.sub)[1]>0){
    l=vector("list",nrow(transect.sub))
    begin.coord=data.frame(lon=transect.sub$west_long,

```

```

                                lat=transect.sub$west_lat)
end.coord=data.frame(lon=transect.sub$east_long,
                      lat=transect.sub$east_lat)
for(i in seq_along(l)){
  l[[i]]=Lines(list(Line(rbind(
    begin.coord[i,],
    end.coord[i,]))),
    as.character(i))
}
l.l[[ind]]=l
transect.sub.r=rasterize(x=SpatialLines(l),
                        y=transect.r,field=1,fun="last",
                        background=NA)
transectAll[[ind]][]=transect.sub.r[]
transectAll[[ind]][Counts[[ind]][]>0]=1
}else{
  transectAll[[ind]][]=rep(NA,q)
}
ind=ind+1
}

```

Combine transect information with count data.

```

C.r=raster(,nrows=y,ncols=x,xmn=xmin,
           xmx=xmax,ymn=ymin,ymx=ymax,crs=NA)
C.r[]=Counts[[7]][]*transectAll[[7]][]
C.r=stack(mget(rep("C.r",20)))
for(i in 1:20){
  C.r[[i]][]=ifelse(Counts[[i]][]>0,Counts[[i]][],
                    transectAll[[i]][]-1)
}

## C data for analysis
C=vec(C.r[])

```

Plot survey data

```

##
levelplot(C.r,margin=FALSE,scales=list(draw=FALSE),
          col.regions=colorRampPalette(
            rev(brewer.pal(11,"Spectral"))),bias=1))

```

Load ISU data and prepare for analysis.

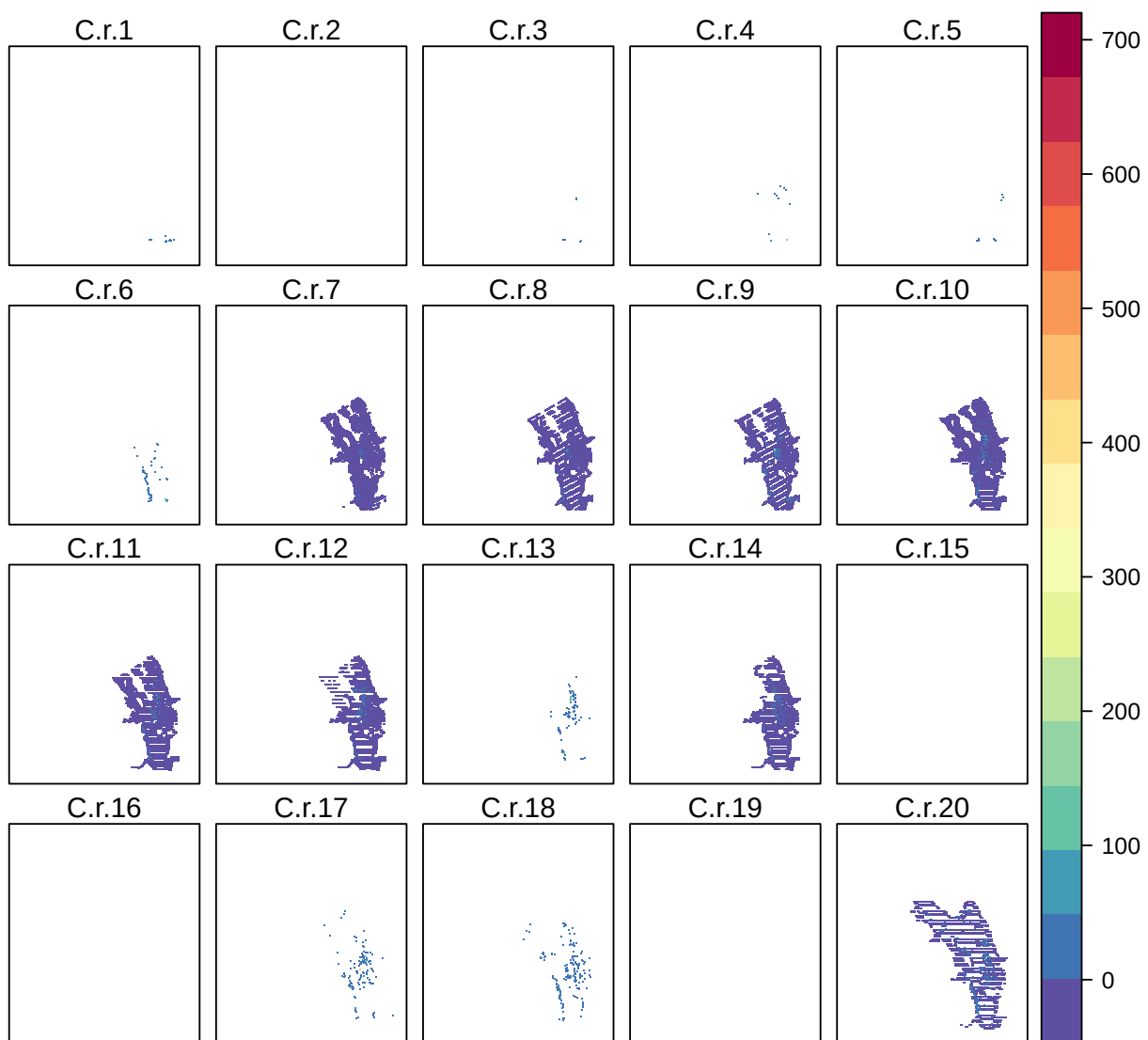


Figure 7: Design based survey data

```

## ISU grid cell indicator
load("~/Dropbox/Post-Doc/RFiles/EcologicalDiffusion/ISU.RData")
ISUind.r=raster(,nrows=y,ncols=x,xmn=xmin,
               xmx=xmax,ymn=ymin,ymx=ymax,crs=NA)

ISU.df=as.data.frame(ISU.all)
ISU.sub=subset(ISU.df,year==1999)
ISU.sub.r=rasterize(cbind(ISU.sub$x,ISU.sub$y),
                   ISUind.r,field=1,fun="last",background=0)

isuInd=ISU.sub.r
isuInd[]=ISU.sub.r[]
isuInd=stack(mget(rep("isuInd",20)))
ind=1
years=1993:2012
for(t in years){
  ISU.sub=subset(ISU.df,year==t)
  if(dim(ISU.sub)[1]>0){
    ISU.sub.r=rasterize(
      x=cbind(
        ISU.sub$x,
        ISU.sub$y),
      y=ISUind.r,
      field=1,
      fun="last",
      background=0)
    isuInd[[ind]][]=ISU.sub.r[]
  }else{
    isuInd[[ind]][]=rep(0,q)
  }
  ind=ind+1
}

```

Plot locations of ISU data

```

levelplot(isuInd,margin=FALSE,scales=list(draw=FALSE),
          col.regions=c("black","green"),colorkey=FALSE)

```

Convert ISU data for analysis.

```

## ISU counts for data for analysis
ISU.ind=vec(isuInd[])
indic=1:length(ISU.ind)
ISU=sort(indic[ISU.ind==1])

```

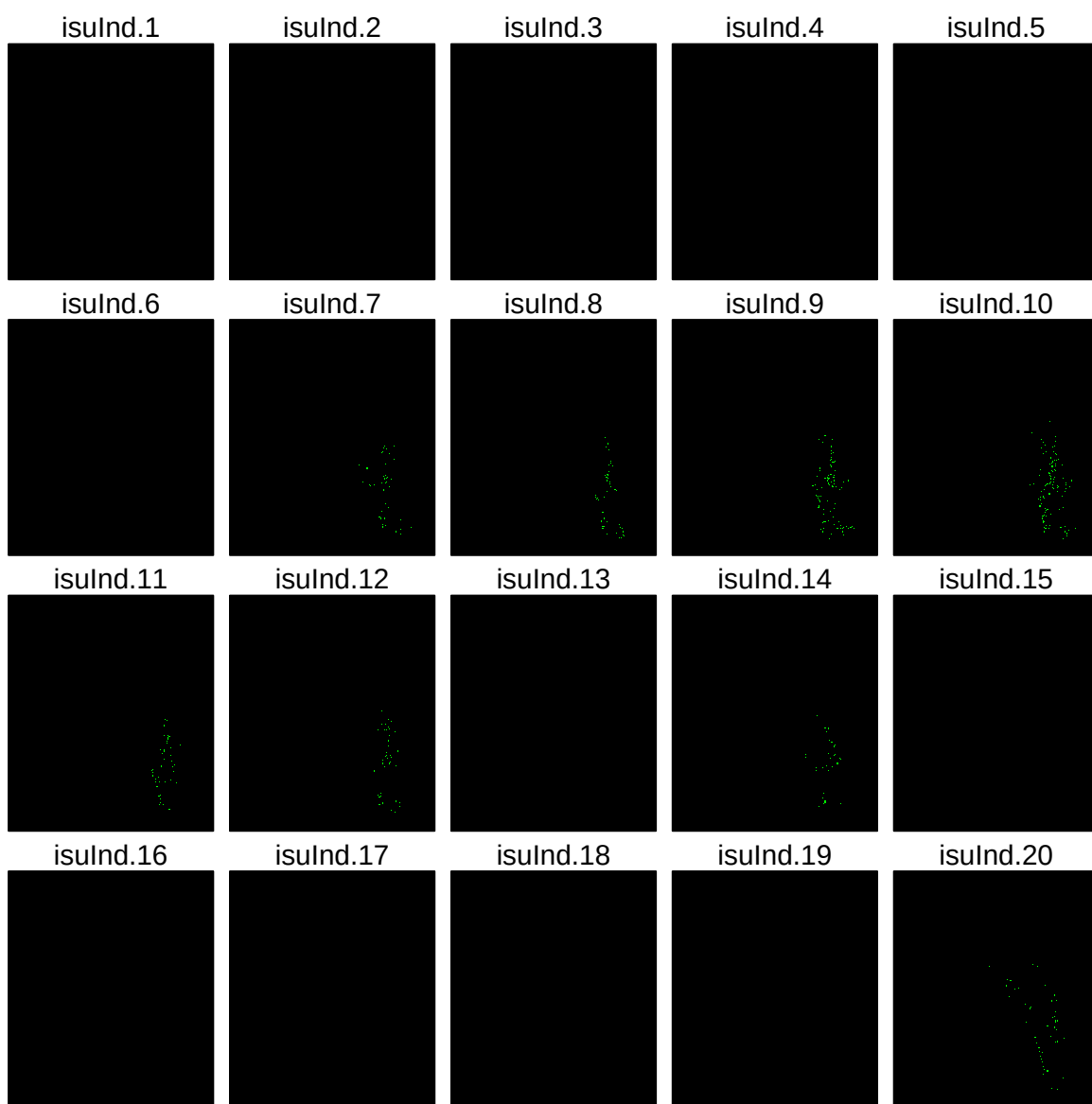


Figure 8: ISU data

```

## N data (known Ns from ISU)
N.r=raster(,nrows=y,ncols=x,xmn=xmin,
           xmx=xmax,ymn=ymin,ymx=ymax,crs=NA)

## Remove aggregation affect of Cs from ISU units
N.sub=subset(ISU.df,year==1999)
N.sub.r=rasterize(cbind(N.sub$x,N.sub$y),N.r,field=N.sub$N,
                  fun="sum",background=0)
CISU.sub.r=rasterize(cbind(N.sub$x,N.sub$y),N.r,field=N.sub$C,
                     fun="sum",background=0)

##
N=N.sub.r
CISU=CISU.sub.r
N=stack(mget(rep("N",20)))
CISU=stack(mget(rep("CISU",20)))
ind=1
for(t in years){
  N.sub=subset(ISU.df,year==t)
  if(dim(N.sub)[1]>0){
    N.sub.r=rasterize(x=cbind(N.sub$x,N.sub$y),
                      y=N.r,field=N.sub$N,fun="sum",
                      background=0)
    CISU.sub.r=rasterize(cbind(N.sub$x,N.sub$y),N.r,field=N.sub$C,
                         fun="sum",background=0)

    N[[ind]][]=N.sub.r[]
    CISU[[ind]][]=CISU.sub.r[]
  }else{
    N[[ind]][]=rep(0,q)
    CISU[[ind]][]=rep(0,q)
  }
  ind=ind+1
}

## Include distribution surveys in N
TransectVec.tmp=vec(transectAll[])
TransectVec=ifelse(C>=0,1,NA)
N.vec=vec(N[])*TransectVec
CISU=vec(CISU[])
CISU=CISU[ISU]
C[ISU]=CISU

## All other N greater than C for initial values
N.vec[-ISU]=ceiling(C[-ISU]/.75)

```

```

for(i in 1:20){
  N[[i]][]=N.vec[(q*i-(q-1)):(q*i),1]
}

## Create Y
Y=ifelse(C>0,1,0)

## Check required conditions
sum(Y[N.vec>0],na.rm=TRUE)==sum(Y,na.rm=TRUE)

## [1] TRUE

sum(C[Y==1],na.rm=TRUE)==sum(C,na.rm=TRUE)

## [1] TRUE

sum(C[N.vec>0],na.rm=TRUE)==sum(C,na.rm=TRUE)

## [1] TRUE

sum(N.vec[-ISU]>=C[-ISU],na.rm=TRUE)==sum(!is.na(C[-ISU]))

## [1] TRUE

sum(C[ISU]<=N.vec[ISU],na.rm=TRUE)==length(ISU)

## [1] TRUE

```

6 New data

New data can be obtained from the XXX database, by following the procedures described in SOP 11 to export a .csv file. After the .csv file has been exported to a local directory (e.g., “~/”), it can be converted to a format that can be used for analysis using the `process_data()` function from the `stoa` package.

```

input.data=read.csv(paste("~/Dropbox/Post-Doc/TechnicalReport/",
                          "input.data.csv",sep=""))
new.data.l=process_data(input.data)
new.data=do.call(rbind,new.data.l)

```

7 MCMC settings and hyperparameters

Several settings require specification prior to running the MCMC algorithm. The settings include the simulation settings, tuning parameters, starting values, hyperparameters for priors, and hold-out settings if cross-validation is going to be used. Additional details of these settings can be found in the `stoa` documentation by typing `?run.mcmc` in the R console.

```
###  
### MCMC settings  
###  
  
seed=2017  
hold.out=FALSE  
n.iter=100000  
burn=0  
N=vec(N[])  
  
## Tuning parameters  
beta.tune=c(0.01304513,  
            0.009,  
            0.004458039,  
            0.015784,  
            0.00531441  
            )  
alpha.tune=5.474963e-05  
theta.tune=4.33026  
kappa.tune=0.264  
  
## Starting values  
p.start=0.7683428  
p2.start=0.9453144  
theta.start=593.6343  
kappa.start=23.76023  
beta.start=c(16.0429,-0.6029451,-0.5674491,0.06574226,-0.7167277)  
alpha.start=0.2150435  
N.start=apply(new.data[,6:10],1,max,na.rm=TRUE)+1  
  
###  
### Model settings (priors)  
###  
  
q.p=1  
r.p=1  
q.r=1  
r.r=1
```

```

mu.kappa=5
sigma2.kappa=1
mu.theta=500
sigma2.theta=250^2
mu.beta=0
sigma2.beta=10^2
q.alpha=1
r.alpha=1

###
### Cross validation holdout samples (if any)
###

ind=1:length(Y)
set.seed(seed)
ho.ind=round(runif(length(Y),.5,8.5))
if(hold.out==TRUE){
  ho=ind[ho.ind==seed]
}else{ho=ind[ho.ind==0]}

```

8 Running the MCMC algorithm

Prior to running the MCMC algorithm, an output file for the data must be specified. This is where MCMC chains will be stored. Once a name is specified, the MCMC algorithm is used to fit the spatio-temporal occupancy abundance model. The algorithm is run using the `run.mcmc` function in the `stoa` package. Computation time is also monitored; the algorithm can take several days to complete.

```

name=~"/Dropbox/Post-Doc/TechnicalReport/MCMCOutput.RData"
N[N>65]=NA
C[C>65]=NA

sys.time=Sys.time()
run.mcmc(n.iter,seed,name,inference.year,
  alpha.start,beta.start,theta.start,kappa.start,N.start,
  q.p,r.p,q.r,r.r,mu.kappa,sigma2.kappa,
  mu.theta,sigma2.theta,mu.beta,sigma2.beta,
  q.alpha,r.alpha,
  beta.tune,alpha.tune,theta.tune,kappa.tune,
  C,CISU,N,Y,X,W,cell,ISU,BoundaryNA,
  keep,dt,timesteps,us.fact,d,
  ho,fix.theta=FALSE,fix.kappa=FALSE,new.data)
Sys.time()-sys.time

```

9 Results

The MCMC output can be loaded using the following command, where `name` is the path and file name where MCMC output was stored from section XXX. The status of the MCMC algorithm can also be checked, while the MCMC algorithm is running, provided > 100 MCMC iterations have occurred.

```
load(name)
(status=sum(out[[6]][,1]!="NA",na.rm=TRUE))
n.iter = status
beta=out[[3]][1:n.iter,]
alpha=out[[4]][1:n.iter]
theta=out[[5]][1:n.iter]
kappa=out[[6]][1:n.iter]
p=out[[2]][1:n.iter]
p2=out[[13]][1:n.iter]
```

9.1 Model diagnostics

9.1.1 Tuning parameters

The MCMC algorithm `run.mcmc` uses Metropolis-Hastings algorithm for sampling parameters whose full-conditional distribution is not conjugate (a Gibbs sampler is used for conjugate full-conditional distributions (i.e., N and p)). Each of the Metropolis-Hastings algorithms used in `run.mcmc` requires a tuning parameter. The function `run.mcmc` includes an adaptive tuning algorithm. The adaptive tuning algorithm must be constant for all parameters in the section of the MCMC chains that are used for inference (see Fig. 9). The following R script is used to examine the adaptive tuner. The acceptance rate influences the efficiency of the MCMC algorithm. Usually, an acceptance rate should be near 0.4. Script to produce a pdf of Fig. 10 is provided below.

```
load(paste("~/MCMCOutput.RData",sep=""))
(status=sum(out[[6]][,1]!="NA",na.rm=TRUE))
n.iter=status
par(mfrow=c(3,3))
plot(out[[11]][1:n.iter,1],type='l',col=1,main="beta0 tuner")
plot(out[[11]][1:n.iter,2],type='l',col=2,main="beta1 tuner")
plot(out[[11]][1:n.iter,3],type='l',col=3,main="beta2 tuner")
plot(out[[11]][1:n.iter,4],type='l',col=2,main="beta3 tuner")
plot(out[[11]][1:n.iter,5],type='l',col=3,main="beta4 tuner")
plot(out[[11]][1:n.iter,6],type='l',col=4,main="alpha tuner")
plot(out[[11]][1:n.iter,7],type='l',col=5,main="theta tuner")
plot(out[[11]][1:n.iter,8],type='l',col=6,main="kappa tuner")
## Acceptance Rate of Metropolis-Hastings samplers
(c("Acceptance Rate",out[[11]]/status))
```

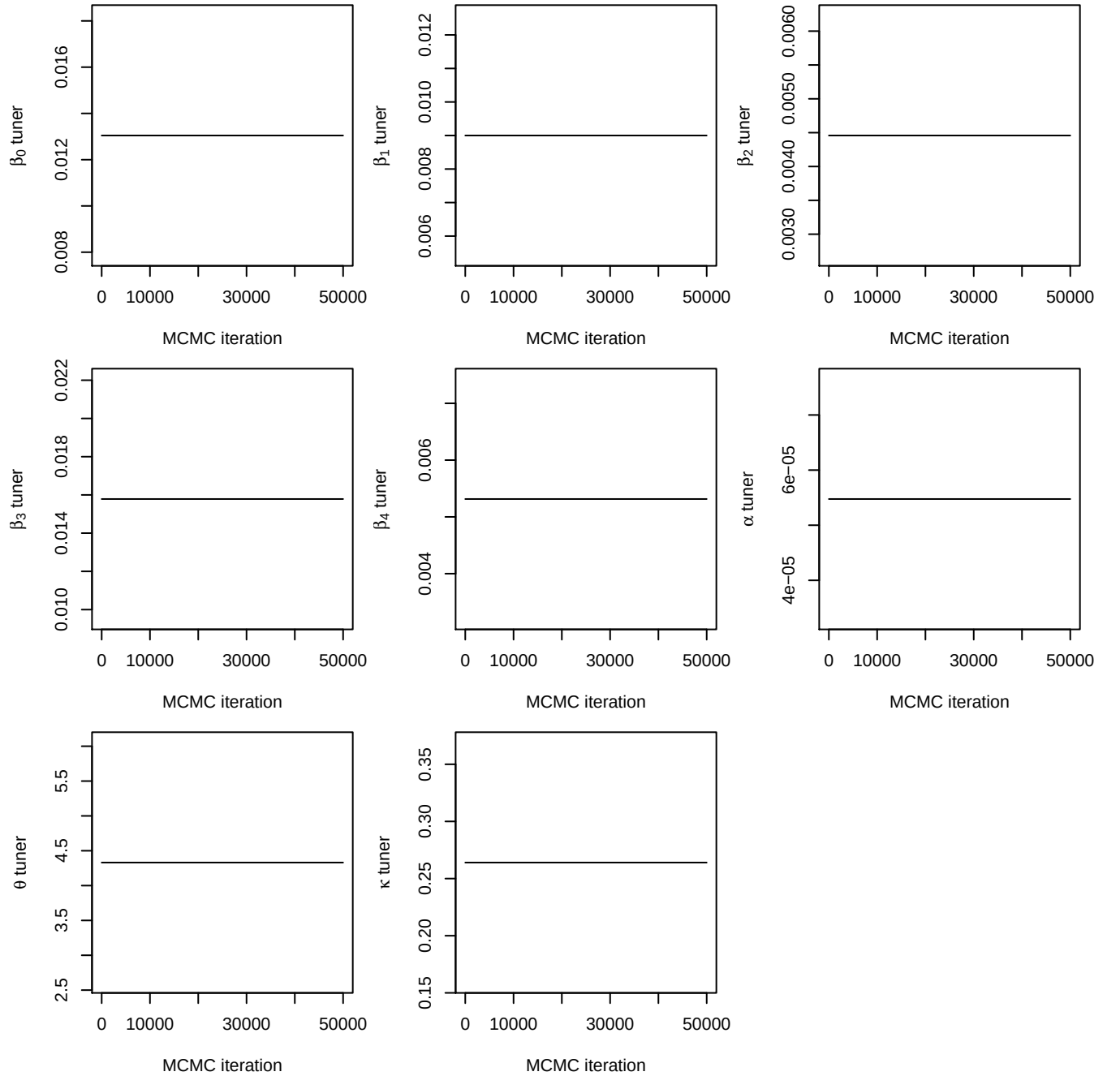


Figure 9: Plots of adaptive tuning values for parameters sampled using Metropolis-Hastings algorithm. The adaptive tuner should be constant for all MCMC iterations used for inference, as is shown in this figure.

9.1.2 Trace plots

Trace plots help identify whether MCMC chains have converged to the marginal posterior distributions, and should look like Fig. 10. Script to produce a pdf of Fig. 10 is provided below.

```

load(paste("~/MCMCOutput.RData", sep=""))
(status=sum(out[[6]][,1]!="NA", na.rm=TRUE))
n.iter=status
par(mfrow=c(3,3), mar=c(4,4,1,1))
plot(out[[3]][1:n.iter,1], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[0]))
plot(out[[3]][1:n.iter,2], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[1]))
plot(out[[3]][1:n.iter,3], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[2]))
plot(out[[3]][1:n.iter,4], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[3]))
plot(out[[3]][1:n.iter,5], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[4]))
plot(out[[4]][1:n.iter,1], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(alpha))
plot(out[[5]][1:n.iter,1], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(theta))
plot(out[[6]][1:n.iter,1], type='l', col=1, main = "",
      xlab="MCMC iteration", ylab=expression(kappa))
plot(out[[2]][1:n.iter,1], type='l', col=1, main="",
      xlab="MCMC iteration", ylab="p")

```

9.2 Model fit

9.2.1 Marginal posterior distributions

Script to produce Fig. 11 is provided below.

```

load(paste("~/MCMCOutput.RData", sep=""))
(status=sum(out[[6]][,1]!="NA", na.rm=TRUE))
n.iter=status
pdf("~/densityplots.pdf")
par(mfrow=c(3,3), mar=c(4,4,1,1))
plot(out[[3]][1:n.iter,1], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[0]))
plot(out[[3]][1:n.iter,2], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[1]))
plot(out[[3]][1:n.iter,3], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[2]))
plot(out[[3]][1:n.iter,4], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[3]))
plot(out[[3]][1:n.iter,5], type='l', col=1, main="",
      xlab="MCMC iteration", ylab=expression(beta[4]))

```

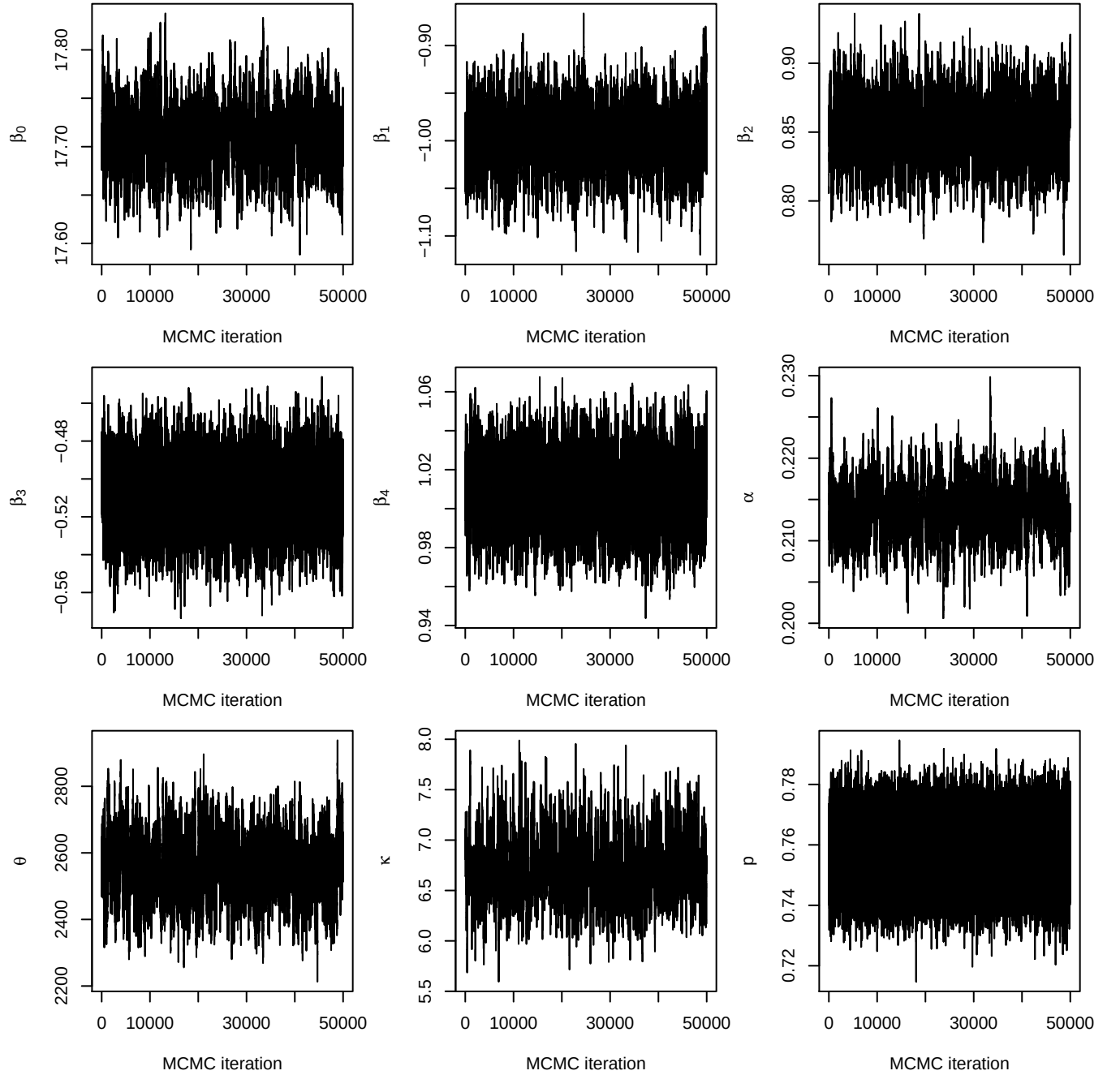


Figure 10: Trace-plots of model parameters.

```

plot(out[[4]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(alpha))
plot(out[[5]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(theta))
plot(out[[6]][1:n.iter,1],type='l',col=1,main = "",
      xlab="MCMC iteration",ylab=expression(kappa))
plot(out[[2]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab="p")

```

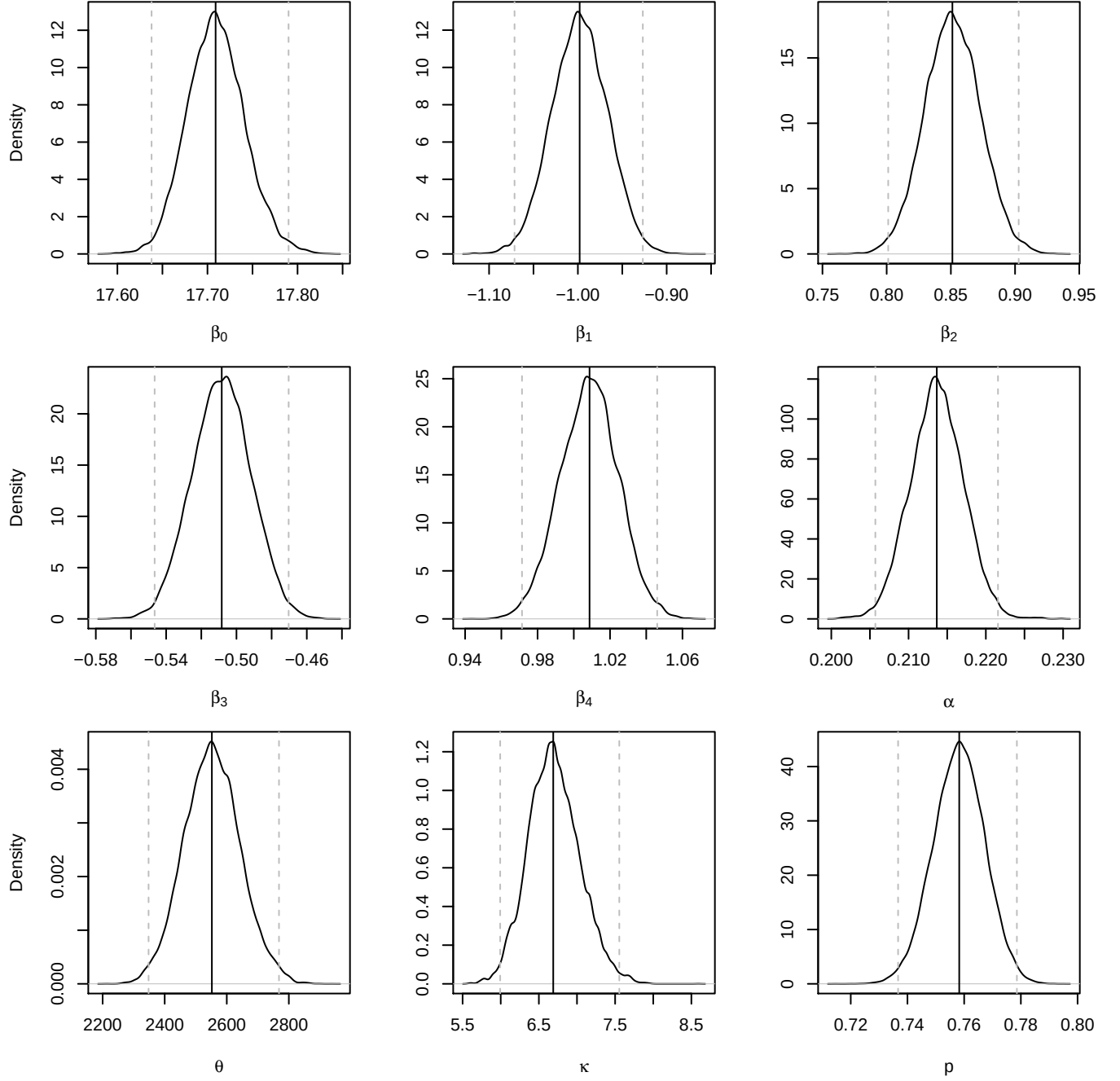


Figure 11: Density plot of MCMC values sampled from each parameters full-conditional posterior distribution. Solid gray line indicates the posterior mean, dashed-gray line indicate the 99% credible intervals.

9.2.2 Results table

The following script summarizes results of the model fit, using the estimated mean and 95% credible intervals of parameter marginal posterior distributions. The output from the R script can be included in a $\text{\LaTeX}$ document.

```

## install.packages("xtable")
library(xtable)

##
## Attaching package: 'xtable'
## The following object is masked from 'package:spam':
##
##      display
## The following object is masked from 'package:timeSeries':
##
##      align
## The following object is masked from 'package:timeDate':
##
##      align

load(paste("~/Dropbox/Post-Doc/RFiles/EcologicalDiffusion/",
           "Williams_etal_2017_UnpublishedSeaOtterEcology/",
           "FullDataModel/MCMCOutputChain4.RData", sep=""))
status=sum(out[[6]][,1]!="NA", na.rm=TRUE)
n.iter=status
betas=out[[3]][1:n.iter,]
alpha=out[[4]][1:n.iter,1]
theta=out[[5]][1:n.iter,1]
kappa=out[[6]][1:n.iter,1]
p=out[[2]][1:n.iter,1]

N.samps=matrix(
  rpois(status*dim(out[[7]])[2], #size=85,,
        out[[7]][1:n.iter,]),
  nr=status,
  nc=dim(out[[7]])[2]
)
Mean=apply(N.samps,2,mean)
quantile.abun=apply(N.samps,2,quantile,c(0.025,0.975))
LowerBound=apply(N.samps,2,quantile,.025)
UpperBound=apply(N.samps,2,quantile,.975)
tab.est=cbind(LowerBound,Mean,UpperBound)
m.beta=apply(betas,2,mean)
beta.lq=apply(betas,2,quantile,.025)
beta.uq=apply(betas,2,quantile,.975)
tab.est2=rbind(tab.est,cbind(beta.lq,m.beta,beta.uq))
m.alpha=mean(alpha)
alpha.lq=quantile(alpha,.025)
alpha.uq=quantile(alpha,.975)
tab.est3=rbind(tab.est2,c(alpha.lq,m.alpha,alpha.uq))

```

```

m.theta=mean(theta)
theta.lq=quantile(theta,.025)
theta.uq=quantile(theta,.975)
tab.est4=rbind(tab.est3,c(theta.lq,m.theta,theta.uq))
m.kappa=mean(kappa)
kappa.lq=quantile(kappa,.025)
kappa.uq=quantile(kappa,.975)
tab.est5=rbind(tab.est4,c(kappa.lq,m.kappa,kappa.uq))
m.p=mean(p)
p.lq=quantile(p,.025)
p.uq=quantile(p,.975)
tab.est6=rbind(tab.est5,c(p.lq,mean(p),p.uq))
colnames(tab.est6)=c("Lower Bound", "Mean", "Upper Bound")
rownames(tab.est6)=c("$\\sum_{i=1}^{q_s}n_{i,1993}$",
"$\\sum_{i=1}^{q_s}n_{i,1994}$",
"$\\sum_{i=1}^{q_s}n_{i,1995}$",
"$\\sum_{i=1}^{q_s}n_{i,1996}$",
"$\\sum_{i=1}^{q_s}n_{i,1997}$",
"$\\sum_{i=1}^{q_s}n_{i,1998}$",
"$\\sum_{i=1}^{q_s}n_{i,1999}$",
"$\\sum_{i=1}^{q_s}n_{i,2000}$",
"$\\sum_{i=1}^{q_s}n_{i,2001}$",
"$\\sum_{i=1}^{q_s}n_{i,2002}$",
"$\\sum_{i=1}^{q_s}n_{i,2003}$",
"$\\sum_{i=1}^{q_s}n_{i,2004}$",
"$\\sum_{i=1}^{q_s}n_{i,2005}$",
"$\\sum_{i=1}^{q_s}n_{i,2006}$",
"$\\sum_{i=1}^{q_s}n_{i,2007}$",
"$\\sum_{i=1}^{q_s}n_{i,2008}$",
"$\\sum_{i=1}^{q_s}n_{i,2009}$",
"$\\sum_{i=1}^{q_s}n_{i,2010}$",
"$\\sum_{i=1}^{q_s}n_{i,2011}$",
"$\\sum_{i=1}^{q_s}n_{i,2012}$",
"$\\beta_0$", "$\\beta_1$", "$\\beta_2$",
"$\\beta_3$", "$\\beta_4$", "$\\alpha$",
"$\\theta$", "$\\kappa$", "p")
print(xtable(tab.est6,caption='Results table'),caption.placement='top',
      sanitize.text.function = function(x){x})

```

10 Posterior predictive distribution

Critical model results for optimal sampling include the posterior predictive distribution (ppd) of future data, given past data. The ppd is used for selecting an optimal sample in SOP 10.

The ppd can be saved as an .RData object that can be loaded in SOP 10, using the following commands.

```
ppd=out[[14]]  
save(ppd,file=~/.RData)
```

Table 1: Results table

	Lower Bound	Mean	Upper Bound
$\sum_{i=1}^{q_s} n_{i,1993}$	641.00	715.69	793.00
$\sum_{i=1}^{q_s} n_{i,1994}$	672.00	739.35	809.00
$\sum_{i=1}^{q_s} n_{i,1995}$	661.00	724.00	790.00
$\sum_{i=1}^{q_s} n_{i,1996}$	697.00	761.33	828.00
$\sum_{i=1}^{q_s} n_{i,1997}$	766.00	832.41	900.00
$\sum_{i=1}^{q_s} n_{i,1998}$	861.00	930.16	1001.00
$\sum_{i=1}^{q_s} n_{i,1999}$	978.00	1051.20	1126.00
$\sum_{i=1}^{q_s} n_{i,2000}$	1117.00	1194.30	1274.00
$\sum_{i=1}^{q_s} n_{i,2001}$	1277.00	1360.31	1445.00
$\sum_{i=1}^{q_s} n_{i,2002}$	1462.00	1551.03	1641.00
$\sum_{i=1}^{q_s} n_{i,2003}$	1674.00	1768.80	1865.00
$\sum_{i=1}^{q_s} n_{i,2004}$	1914.00	2016.68	2120.00
$\sum_{i=1}^{q_s} n_{i,2005}$	2188.00	2298.66	2412.00
$\sum_{i=1}^{q_s} n_{i,2006}$	2497.00	2618.87	2744.00
$\sum_{i=1}^{q_s} n_{i,2007}$	2847.00	2983.52	3123.00
$\sum_{i=1}^{q_s} n_{i,2008}$	3243.00	3397.53	3554.00
$\sum_{i=1}^{q_s} n_{i,2009}$	3691.00	3868.98	4050.00
$\sum_{i=1}^{q_s} n_{i,2010}$	4198.00	4404.84	4615.00
$\sum_{i=1}^{q_s} n_{i,2011}$	4774.00	5014.88	5260.00
$\sum_{i=1}^{q_s} n_{i,2012}$	5420.00	5710.16	6006.00
β_0	17.65	17.71	17.78
β_1	-1.06	-1.00	-0.94
β_2	0.81	0.85	0.89
β_3	-0.54	-0.51	-0.48
β_4	0.98	1.01	1.04
α	0.21	0.21	0.22
θ	2374.51	2549.91	2733.73
κ	6.07	6.70	7.39
p	0.74	0.76	0.78