

SOP 09: Estimating Occupancy and Abundance

Version 2.0

Contents

1	Summary	3
2	Setting up the R workspace	3
3	Preliminary Glacier Bay settings	4
4	Boundary and covariate information	4
5	Pre-2019 data	12
6	New data	18
7	MCMC settings and hyperparameters	18
8	Running the MCMC algorithm	22
9	Results	24
9.1	Model diagnostics	25
9.1.1	Tuning parameters	25
9.1.2	Trace plots	26
9.2	Model fit	28
9.2.1	Marginal posterior distributions	28
9.2.2	Results table	30
10	Posterior predictive distribution	32
11	References	33

List of Figures

1	Bathymetry data for Glacier Bay National Park	6
2	Depth covariate	8
3	Distance-to-shore covariate	9
4	Bottom slope covariate	10
5	Shoreline complexity covariate	11
6	Area of inference	13
7	Design based survey data	17
8	Adaptive tuner	26
9	Trace-plots	28

10	Density plots	30
----	-------------------------	----

List of Tables

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 Lu et al. (2019) and Williams et al. (2017a, 2017b, 2018). 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 aerial photograph data, collection of which began in 2017 (see SOP XX and XX). The design-based surveys and distributional surveys were conducted by the U.S. Geological Survey. 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",
                    "ggmap",
                    "ggplot2",
                    "maptools",
                    "parallel",
                    "RCurl",
                    "rgdal",
                    "rgeos"
)
## install.packages(required.packages)
lapply(required.packages,library,character.only=TRUE,quietly=TRUE)

## set working directory; modify path as needed
setwd("~/Google Drive/otters/otter1/seot")
```

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 400×400 m 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, `time.steps` are the total number of time steps for the entire study period, `keep` is vector identifying which time steps to keep for inference, `us.fact` is the upscale factor for homogenization, and `res` is the dimension of the grid cells. 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=2019
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=ymax,
            ymn=ymin,ymx=ymax,crs=NA)
q=x*y
cell[1:q]=1:q
d=c(442000,6465000)
dt=1/100
time.steps=1/dt*length(1993:inference.year)
keep=seq(1,time.steps,time.steps/length(1993:inference.year))
us.fact=10
res=400

## 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 be 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 (e.g., shore, islands, etc.) 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("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,
```

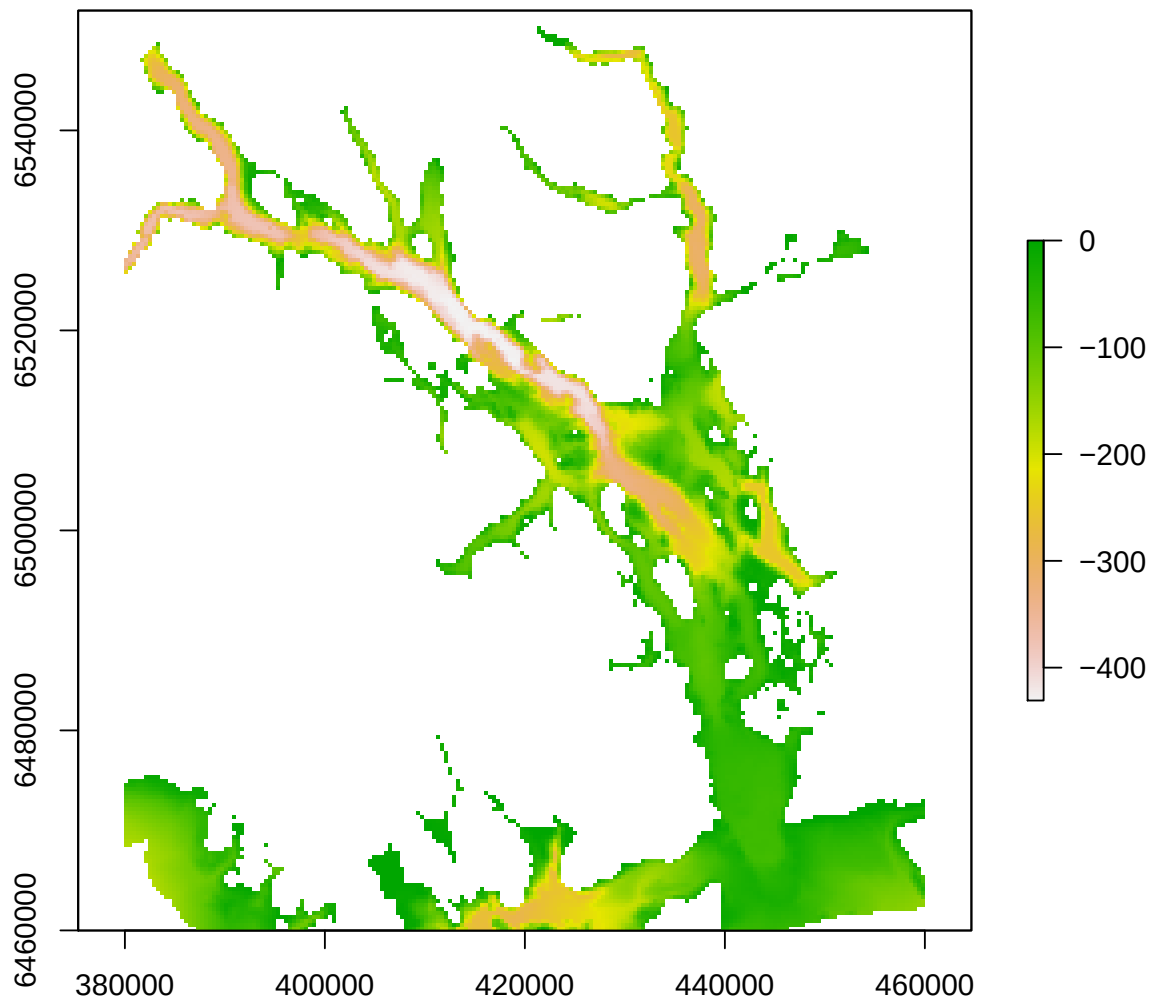


Figure 1: Bathymetry data for Glacier Bay National Park

```
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,  
                    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[])
```

Inference area:

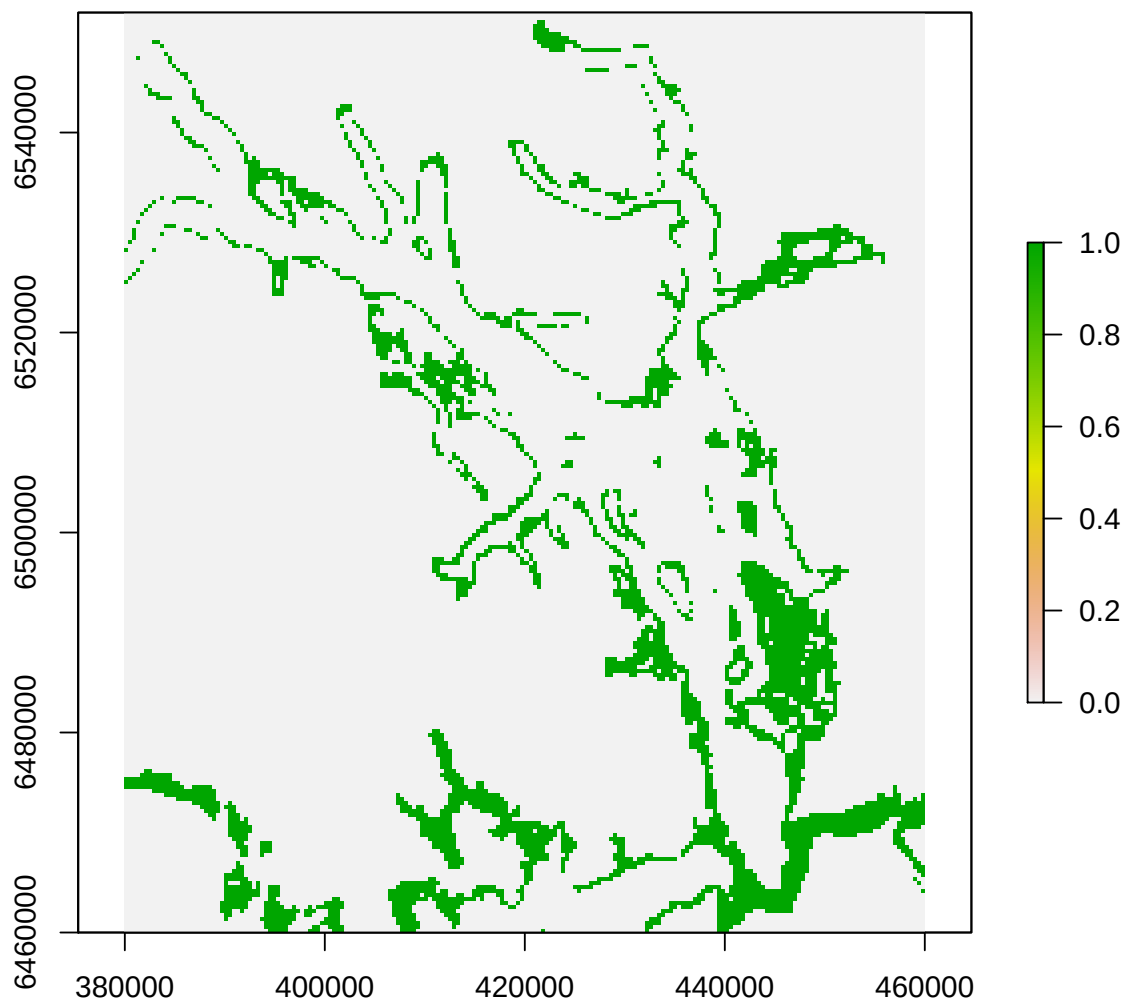


Figure 2: Depth covariate

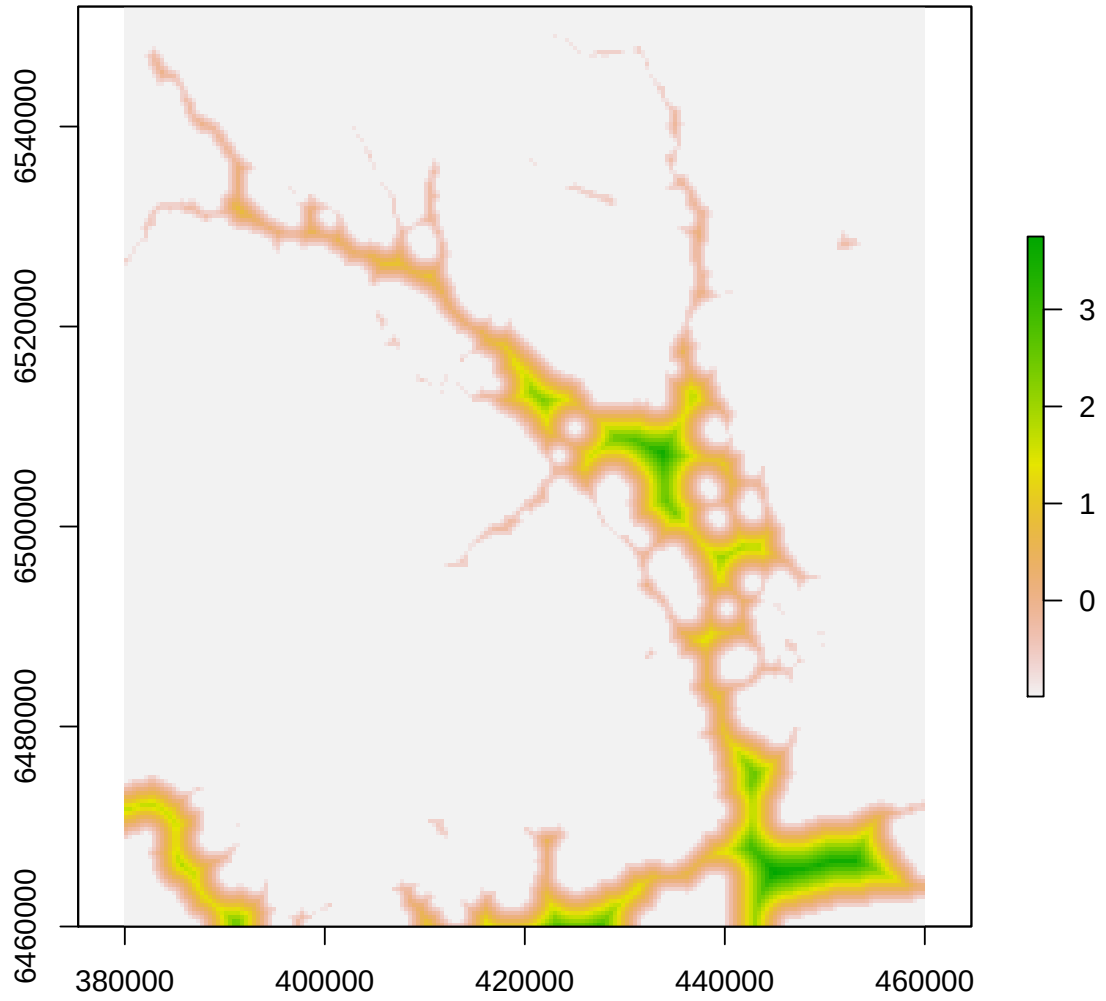


Figure 3: Distance-to-shore covariate

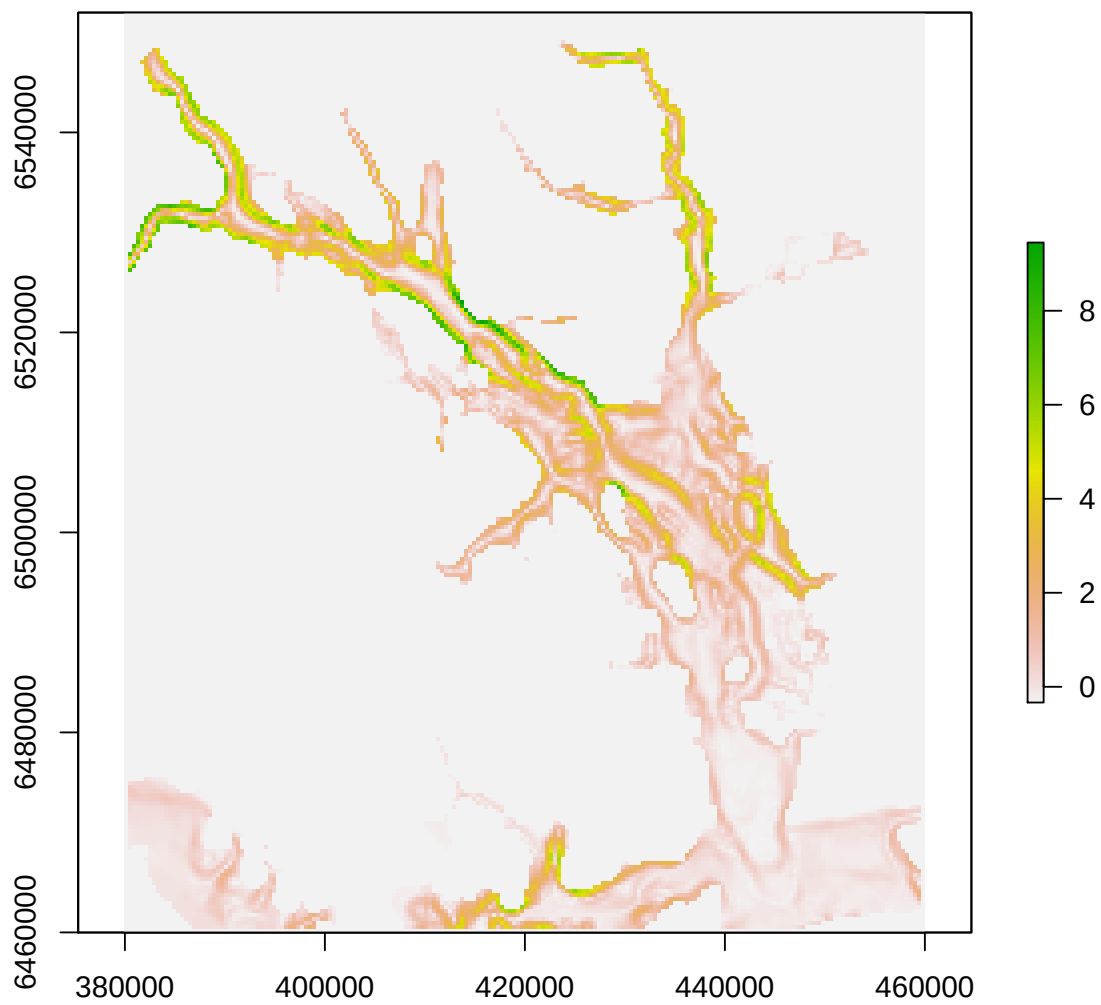


Figure 4: Bottom slope covariate

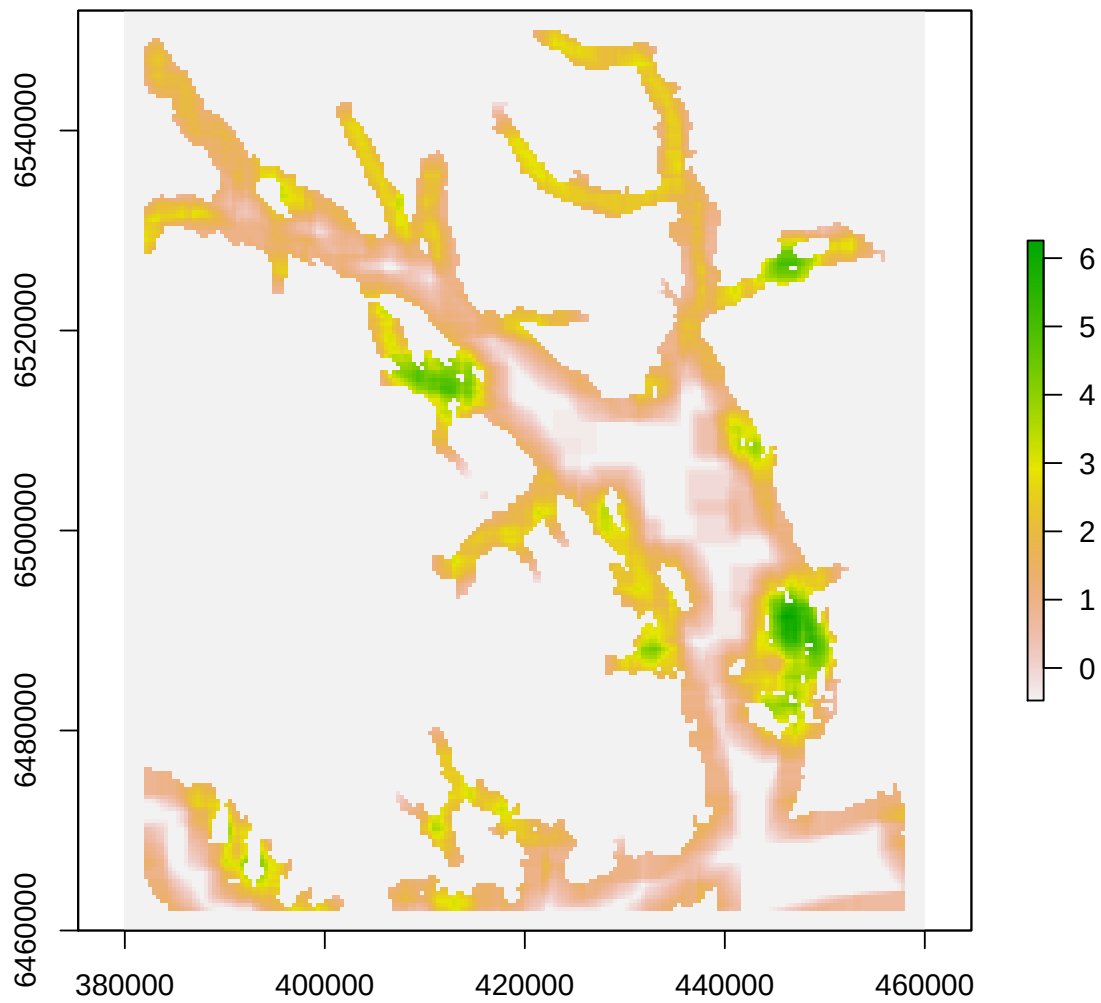


Figure 5: Shoreline complexity covariate

```

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
BoundaryInf.v=rep(BoundaryInf[],length(keep))

plot(BoundaryInf)

```

5 Pre-2019 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("GLBA_noDots.csv")
dataDistr=read.csv('DistribSurveysC.csv')

## 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[]

```

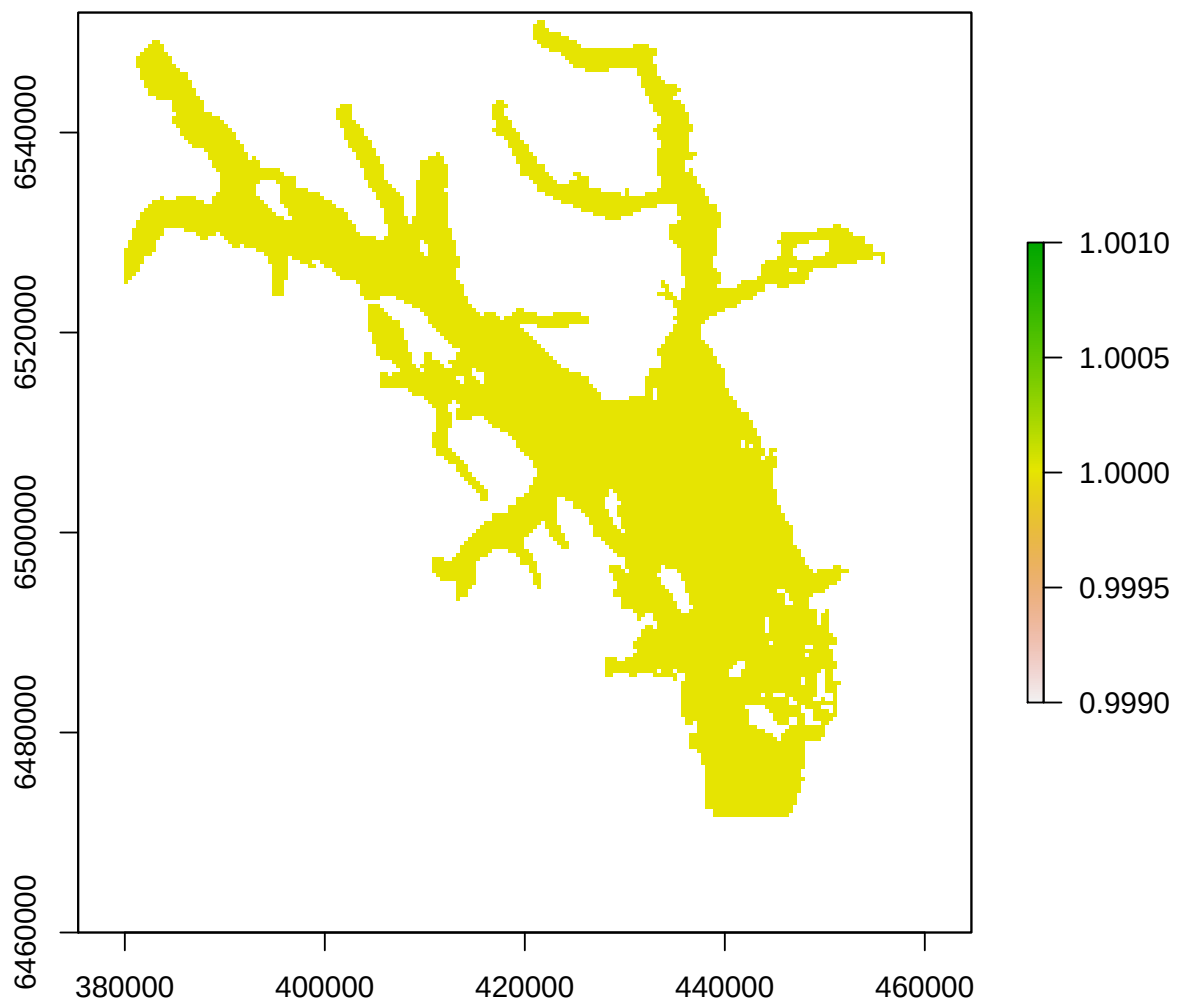


Figure 6: Area of inference

```

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,
                  mx=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]][>0]=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]][>0]=transect.sub.r
    transectAll[[ind]][Counts[[ind]][>0]=1
  }else{
    transectAll[[ind]][>0]=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]][>0]*transectAll[[7]][>0]
C.r=stack(mget(rep("C.r",20)))
for(i in 1:20){
  C.r[[i]][>0]=ifelse(Counts[[i]][>0,Counts[[i]][>0],
                      transectAll[[i]][>0]-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.

```
## Prep the ISU data
all.ISU.data=read.csv(paste("All ISUs 1999_2012_02032016.csv"))
ISU.N=all.ISU.data$circle.adt+
  all.ISU.data$circle.pup
ISU.Y=all.ISU.data$strip.adt+
  all.ISU.data$strip.pup
ISU.N[ISU.Y>ISU.N]=ISU.Y[ISU.Y>ISU.N]
sum(ISU.N<ISU.Y) # check no counts greater than true N

## [1] 0

ISU.year=all.ISU.data$year
ISU.data=data.frame(ISU.N,ISU.Y,ISU.year)
Y.1999=ISU.data$ISU.Y[ISU.data$ISU.year==1999]
N.1999=ISU.data$ISU.N[ISU.data$ISU.year==1999]
Y.2000=ISU.data$ISU.Y[ISU.data$ISU.year==2000]
N.2000=ISU.data$ISU.N[ISU.data$ISU.year==2000]
Y.2001=ISU.data$ISU.Y[ISU.data$ISU.year==2001]
N.2001=ISU.data$ISU.N[ISU.data$ISU.year==2001]
Y.2002=ISU.data$ISU.Y[ISU.data$ISU.year==2002]
N.2002=ISU.data$ISU.N[ISU.data$ISU.year==2002]
Y.2003=ISU.data$ISU.Y[ISU.data$ISU.year==2003]
N.2003=ISU.data$ISU.N[ISU.data$ISU.year==2003]
Y.2004=ISU.data$ISU.Y[ISU.data$ISU.year==2004]
N.2004=ISU.data$ISU.N[ISU.data$ISU.year==2004]
Y.2006=ISU.data$ISU.Y[ISU.data$ISU.year==2006]
N.2006=ISU.data$ISU.N[ISU.data$ISU.year==2006]
Y.2012=ISU.data$ISU.Y[ISU.data$ISU.year==2012]
N.2012=ISU.data$ISU.N[ISU.data$ISU.year==2012]
ISU=list(Y.1999=Y.1999,
         N.1999=N.1999,
         Y.2000=Y.2000,
         N.2000=N.2000,
         Y.2001=Y.2001,
         N.2001=N.2001,
         Y.2002=Y.2002,
         N.2002=N.2002,
         Y.2003=Y.2003,
```

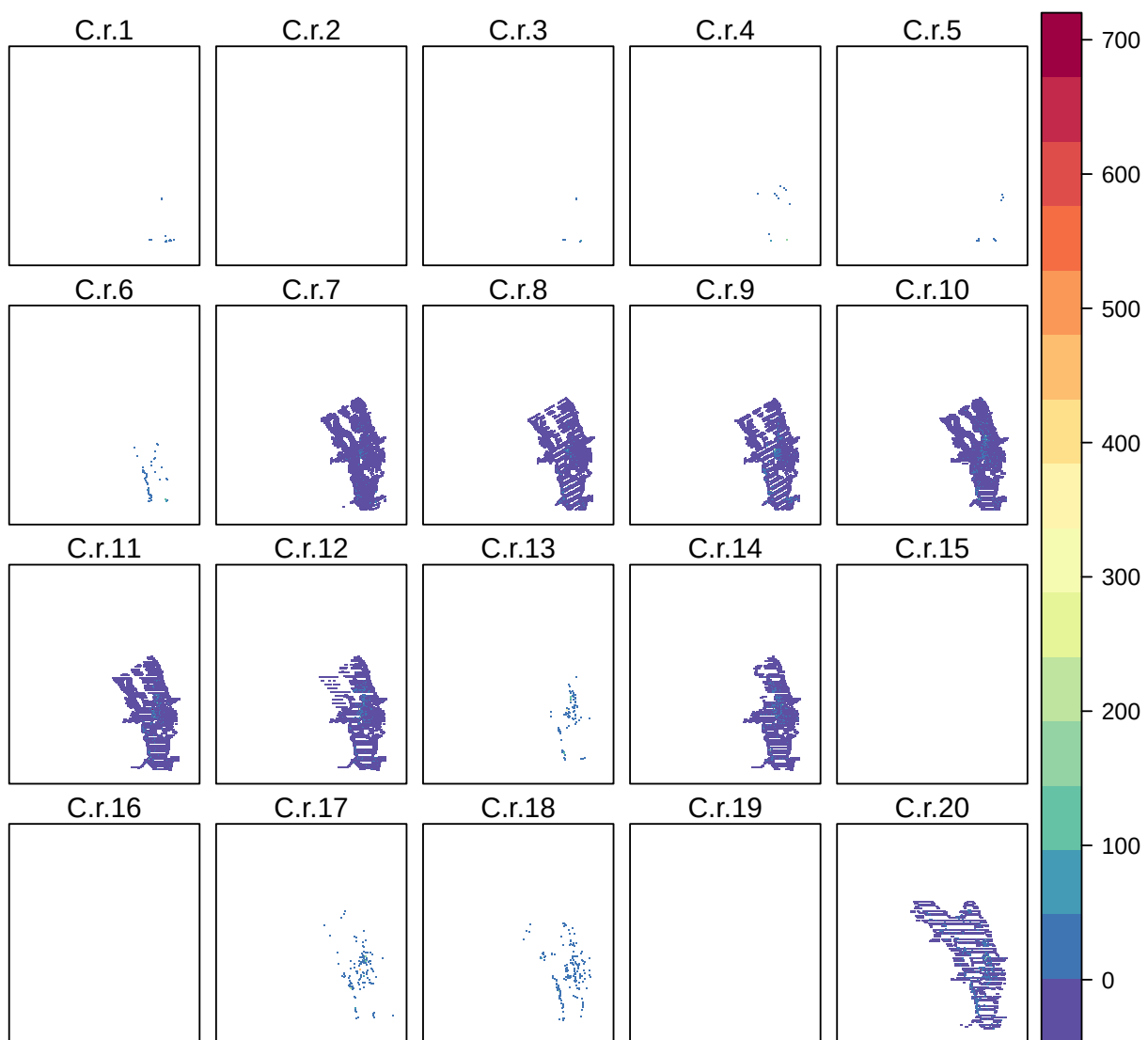


Figure 7: Design based survey data

```

N.2003=N.2003,
Y.2004=Y.2004,
N.2004=N.2004,
Y.2006=Y.2006,
N.2006=N.2006,
Y.2012=Y.2012,
N.2012=N.2012)

## Create Y
Y=ifelse(C>0,1,0)
## Check required conditions
sum(C[Y==1],na.rm=TRUE)==sum(C,na.rm=TRUE)

## [1] TRUE

sum(ISU.N<ISU.Y)

## [1] 0

```

Load and prepare aerial photograph data.

```

## load aerial photo data files and prep
files = c("SO_D_20170719_R.csv", "SO_D_20170721_O.csv",
          "SO_D_20170728_A.csv", "SO_D_20180720_R.CSV",
          "SO_D_20180719_O.csv", "SO_D_20180718_A.csv")
years=c(2017,2017,2017,2018,2018,2018)
source(paste0(getwd(), '/prep_photo.R'))
Y.p=prep.photo(files,years)

```

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 with the `prep.photo()` function, as above.

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 number of chains to be run in parallel.

```

###
### MCMC settings
###
No.chains=2
seed=c(1,99,651,760,1201,1223,1335,1416,2033,2132,2203,2977,3392,
       3514,4039,4221,5385,7039,7503,7959)[1:No.chains]
n.iter=1000
checkpoint=100
savepoint=100

###
### Bundle data
###
data=list(Y=C,
          Y.aerial=Y.p,
          ISU=ISU,
          X=X,
          seed=seed)
Background=list(bath=bath.raster,
                Boundary=Boundary,
                BoundaryNA=BoundaryNA,
                BoundaryInf=BoundaryInf,
                BoundaryInf.v=BoundaryInf.v,
                Boundary.us=Boundary.us)

###
### Spatio-temporal settings
###
extent=c(xmin,xmax,ymin,ymax)
st.info=list(dt=dt,
             time.frame=1993:inference.year,
             us.fact=us.fact,
             res=res,
             d=d,
             extent=extent
)

## Tuning parameters
gamma.tune=0.002835121
beta.tune=c(0.04,
            0.06,
            0.05,
            0.16,
            0.03)

```

```

theta.tune=100
kappa.tune=0.5
tau.tune=0.001
K.tune=0.1

tuners=list(gamma.tune=gamma.tune,
            beta.tune=beta.tune,
            theta.tune=theta.tune,
            kappa.tune=kappa.tune,
            tau.tune=tau.tune,
            K.tune=K.tune)

## Starting values
gamma=0.19
beta=c(15.75,
       -1.58,
        0.39,
       -0.4,
        1.21)
theta=800
kappa=12
p.1999=0.71
p.2000=0.81
p.2001=0.76
p.2002=0.84
p.2003=0.75
p.2004=0.76
p.2006=0.76
p.2012=0.59
p.2017=0.71
p.2018=0.71
p.ind=rep(NA,length(1993:inference.year))
p.ind[7]=p.1999
p.ind[8]=p.2000
p.ind[9]=p.2001
p.ind[10]=p.2002
p.ind[11]=p.2003
p.ind[12]=p.2004
p.ind[14]=p.2006
p.ind[20]=p.2012
p.ind[25]=p.2017
p.ind[26]=p.2018
tau=0.03
K=5.5

```

```

inits=list(gamma=gamma,
          beta=beta,
          theta=theta,
          kappa=kappa,
          tau=tau,
          p.ind=p.ind,
          K=K
)

###
### Model settings (priors)
###
## beta Normal(mu.beta,var.beta)
mu.beta=0
var.beta=10^2
## gamma Uniform(q.gamma,r.gamma)
q.gamma=0.12
r.gamma=0.30
## theta Normal+(mu.theta,var.theta)
mu.theta=600
var.theta=60^2
## kappa Normal+(mu.kappa,var.kappa)
mu.kappa=10
var.kappa=100^2
## p Beta(q.p,r.p)
q.p=1
r.p=1
## Aerial photograph priors Beta(p.alpha,p.beta)
## (moment matching from MEE paper -use Appendix S2 from ms)
p.alpha=44.04937
p.beta=13.40566
## tau Uniform(q.tau,r.tau)
q.tau=0
r.tau=1
## K Uniform(K.alpha,K.beta)
K.alpha=0
K.beta=30

priors=list(
  beta.prior=c(mu.beta,var.beta),
  gamma.prior=c(q.gamma,r.gamma),
  theta.prior=c(mu.theta,var.theta),
  kappa.prior=c(mu.kappa,var.kappa),
  p.prior=c(q.p,r.p),

```

```

p.aerial=c(p.alpha,p.beta),
tau.prior=c(q.tau,r.tau),
K.prior=c(K.alpha,K.beta)
)

```

8 Running the MCMC algorithm

Prior to running the MCMC algorithm, we select the parameters we want to monitor, remove anything in the workspace that is not needed, and specify an output file for the MCMC samples and diagnostics. This is where MCMC chains will be stored. Once a name is specified, the MCMC algorithm is run in parallel to fit the spatio-temporal occupancy abundance model. The algorithm is run using the `MCMC()` and `run.chain.2pl.list()` functions from the scripts `MCMC_Logistic_SOP.R` and `parallel_fun.R`. The latter is a wrapper to run MCMC in parallel to minimize computation time by spreading MCMC chains across CPU cores. Computation time is also monitored; the algorithm can take several days to complete. The number of iterations completed by each chain is given by the names of the empty files saved to the `progress` directory.

```

## name parameters to monitor
parameters=c("gamma",
             "beta",
             "kappa",
             "theta",
             "tau",
             "p",
             "n.tot",
             "n.tot.v",
             "tuners",
             "K",
             'accept')

## clear the workspace to minimize memory usage
rm(list=ls()[!ls() %in% c("data",
                        "priors",
                        "inits",
                        "parameters",
                        "st.info",
                        "Background",
                        "n.iter",
                        "checkpoint",
                        "savepoint",
                        "out.loc.l",
                        "run.chain.2pl.list",
                        'No.chains',

```

```

        'seed',
        'tuners']])
## load the function to run chains in parallel
source(paste0(getwd(), '/parallel_fun.R'))
## file output location
dir.create(paste0(getwd(), '/otter_out'))
out.loc.l=list()
for(i in 1:No.chains){
    out.loc.l[[i]]=path.expand(paste0(getwd(), "/otter_out/",
                                       "MCMC.Output.seed.",
                                       seed[i], ".RData"))
}
## create directory to monitor progress
dir.create(paste0(getwd(), '/progress'))
## load the MCMC function
source(paste0(getwd(), "/MCMC_Logistic_SOP.R"))
##+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
sys.time=Sys.time()
run.chain.2pl.list(data=data,
                   priors=priors,
                   inits=inits,
                   parameters=parameters,
                   st.info=st.info,
                   Background=Background,
                   n.iter=n.iter,
                   checkpoint=checkpoint,
                   savepoint=savepoint,
                   out.loc=out.loc.l,
                   tuners=tuners
)

## $`1`
## NULL
##
## $`99`
## NULL

Sys.time()-sys.time

## Time difference of 46.51169 mins

```

9 Results

The MCMC output can be loaded using the following command, where `otter_out` 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.

```
## check progress
list.files(path=paste0(getwd(), '/progress'))

## [1] "10" "100" "1000" "20" "200" "30" "300" "40" "400" "50"
## [11] "500" "60" "600" "70" "700" "80" "800" "90" "900"

## gather MCMC output
file_list=list.files(path=paste0(getwd(), "/otter_out"))
chains=list()
beta=list()
theta=list()
kappa=list()
gamma=list()
K=list()
tau=list()
p=list()
accept=list()
tuners=list()
N=list()
N.ppd=list()
for(i in 1:No.chains){
  load(paste0(getwd(), "/otter_out/", file_list[i]))
  chains[[i]]=MCMC.Chains
  beta[[i]]=MCMC.Chains$beta
  theta[[i]]=MCMC.Chains$theta
  kappa[[i]]=MCMC.Chains$kappa
  gamma[[i]]=MCMC.Chains$gamma
  K[[i]]=MCMC.Chains$K
  tau[[i]]=MCMC.Chains$tau
  p[[i]]=MCMC.Chains$p
  tuners[[i]]=MCMC.Chains$tuners
  accept[[i]]=MCMC.Chains$accept
  N[[i]]=MCMC.Chains$n.tot
  N.ppd[[i]]=MCMC.Chains$n.tot.v
}
```

9.1 Model diagnostics

9.1.1 Tuning parameters

The MCMC algorithm uses Metropolis-Hastings algorithm for sampling parameters whose full-conditional distribution are not conjugate, whereas a Gibbs sampler is used for conjugate full-conditional distributions (i.e., N and p). Each of the Metropolis-Hastings algorithms requires a tuning parameter. The function `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. 8). 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. 8 is provided below.

```
(status=max(as.numeric(list.files(path=paste0(getwd(), '/progress')))))

## [1] 1000

n.iter=status
par(mfrow=c(4,3))
# for chain k
k=1
plot(tuners[[k]][1:n.iter,1],type='l',col=4,main="gamma tuner")
plot(tuners[[k]][1:n.iter,2],type='l',col=1,main="beta0 tuner")
plot(tuners[[k]][1:n.iter,3],type='l',col=2,main="beta1 tuner")
plot(tuners[[k]][1:n.iter,4],type='l',col=3,main="beta2 tuner")
plot(tuners[[k]][1:n.iter,5],type='l',col=2,main="beta3 tuner")
plot(tuners[[k]][1:n.iter,6],type='l',col=3,main="beta4 tuner")
plot(tuners[[k]][1:n.iter,7],type='l',col=5,main="theta tuner")
plot(tuners[[k]][1:n.iter,8],type='l',col=6,main="kappa tuner")
plot(tuners[[k]][1:n.iter,9],type='l',col=7,main="tau tuner")
plot(tuners[[k]][1:n.iter,10],type='l',col=7,main="K tuner")
## Acceptance Rate of Metropolis-Hastings samplers
(c("Acceptance Rates",accept[[k]][status,]/status))

## [1] "Acceptance Rates" "0.426" "0.47" "0.483"
## [5] "0.455" "0.2" "0.497" "0.34"
## [9] "0.306" "0.379" "0.833"
```

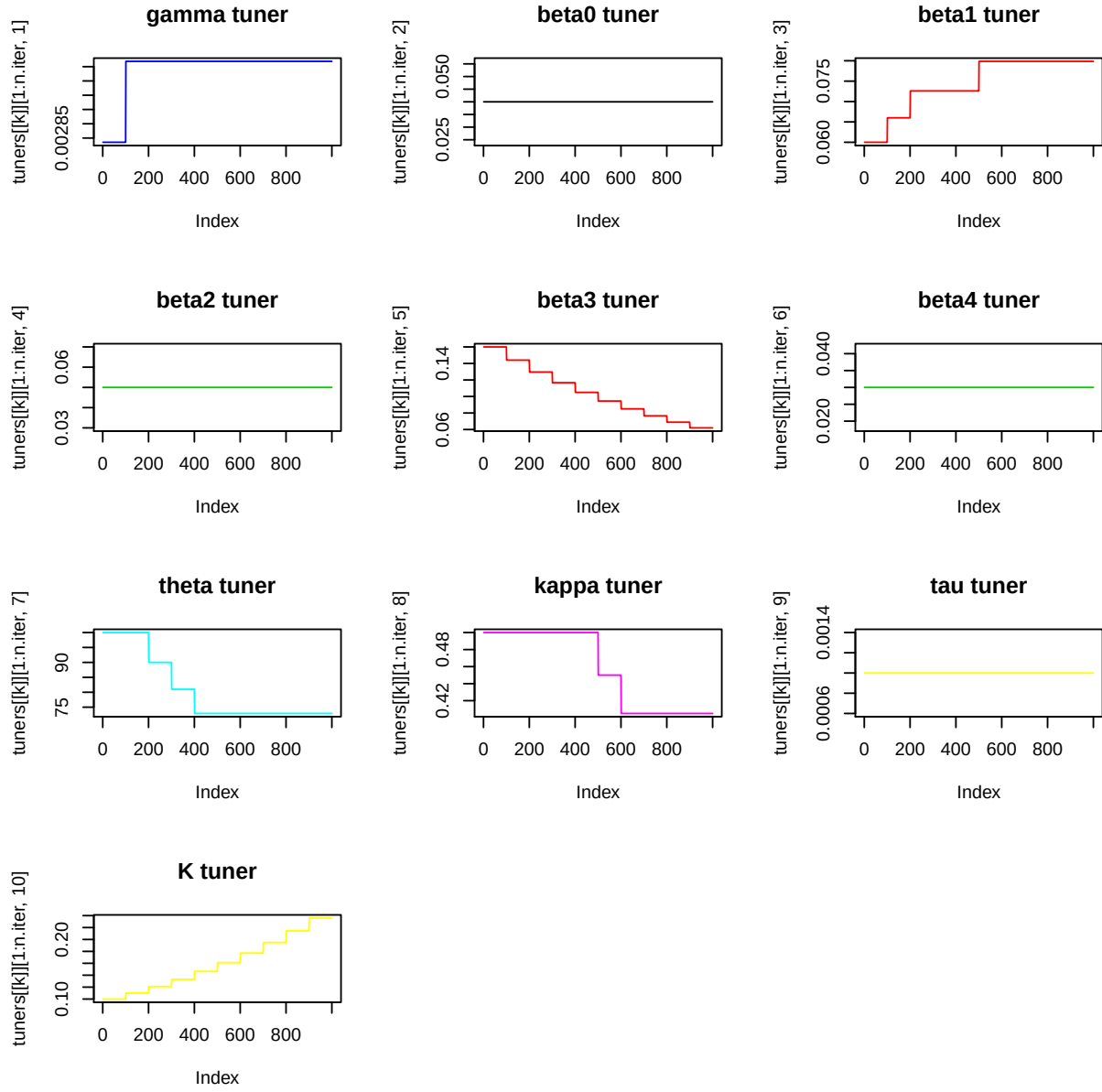


Figure 8: Plots of adaptive tuning values for parameters sampled using Metropolis-Hastings algorithm. The adaptive tuner should be constant for all MCMC iterations used for inference, as is shown in this figure.

9.1.2 Trace plots

Trace plots help identify whether MCMC chains have converged to the marginal posterior distributions, and should look like Fig. 9. Script to produce a pdf of Fig. 9 is provided below.

```

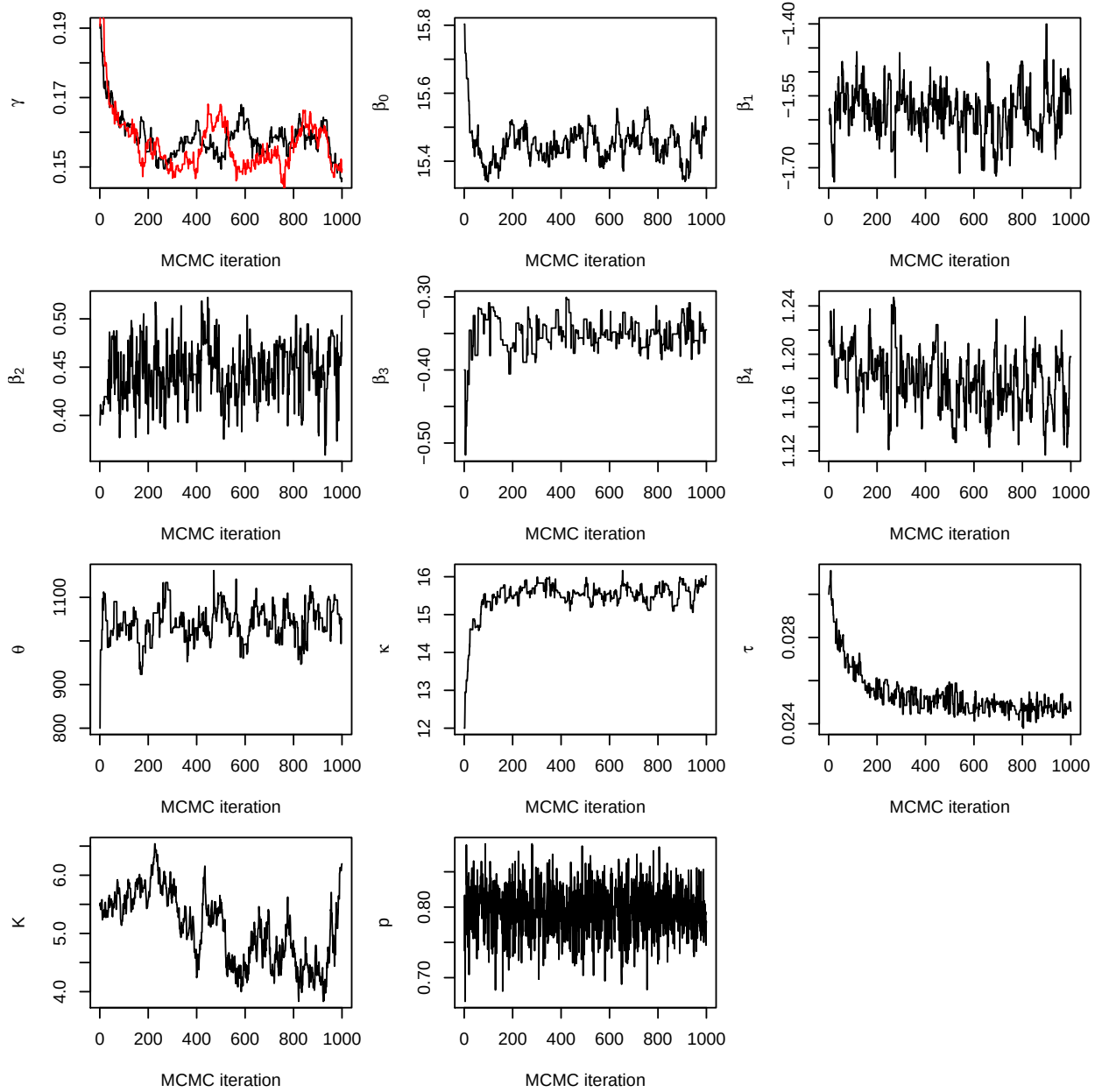
(status=max(as.numeric(list.files(path=paste0(getwd(), '/progress')))))

## [1] 1000

n.iter=status
# for chaine k; additional chains can be added to each plot with lines()
k=1
par(mfrow=c(4,3),mar=c(4,4,1,1))
plot(gamma[[k]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(gamma))
lines(gamma[[2]][1:n.iter,1],col=2)
plot(beta[[k]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[0]))
plot(beta[[k]][1:n.iter,2],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[1]))
plot(beta[[k]][1:n.iter,3],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[2]))
plot(beta[[k]][1:n.iter,4],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[3]))
plot(beta[[k]][1:n.iter,5],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[4]))
plot(theta[[k]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(theta))
plot(kappa[[k]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(kappa))
plot(tau[[k]][1:n.iter,1],type='l',col=1,main = "",
      xlab="MCMC iteration",ylab=expression(tau))
plot(K[[k]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab="K")
# p[,1:8] are ISU, and p[,9:10] are aerial
plot(p[[k]][1:n.iter,1],type='l',col=1,main="",
      xlab="MCMC iteration",ylab="p")

```

Figure 9: Trace-plots of model parameters.



9.2 Model fit

9.2.1 Marginal posterior distributions

Script to produce Fig. 10 is provided below.

```
(status=max(as.numeric(list.files(path=paste0(getwd(), '/progress')))))
## [1] 1000
```

```

n.iter=status
k=1 # for chain k
#pdf("~/densityplots.pdf")
par(mfrow=c(4,3),mar=c(4,4,1,1))
plot(density(gamma[[k]][1:n.iter,1]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[0]))
plot(density(beta[[k]][1:n.iter,1]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[0]))
plot(density(beta[[k]][1:n.iter,2]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[1]))
plot(density(beta[[k]][1:n.iter,3]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[2]))
plot(density(beta[[k]][1:n.iter,4]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[3]))
plot(density(beta[[k]][1:n.iter,5]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(beta[4]))
plot(density(tau[[k]][1:n.iter,1]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(tau))
plot(density(theta[[k]][1:n.iter,1]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab=expression(theta))
plot(density(kappa[[k]][1:n.iter,1]),type='l',col=1,main = "",
      xlab="MCMC iteration",ylab=expression(kappa))
plot(density(K[[k]][1:n.iter,1]),type='l',col=1,main = "",
      xlab="MCMC iteration",ylab=expression(K))
# p[,1:8] are ISU, and p[,9:10] are aerial
plot(density(p[[k]][1:n.iter,1]),type='l',col=1,main="",
      xlab="MCMC iteration",ylab="p")

```

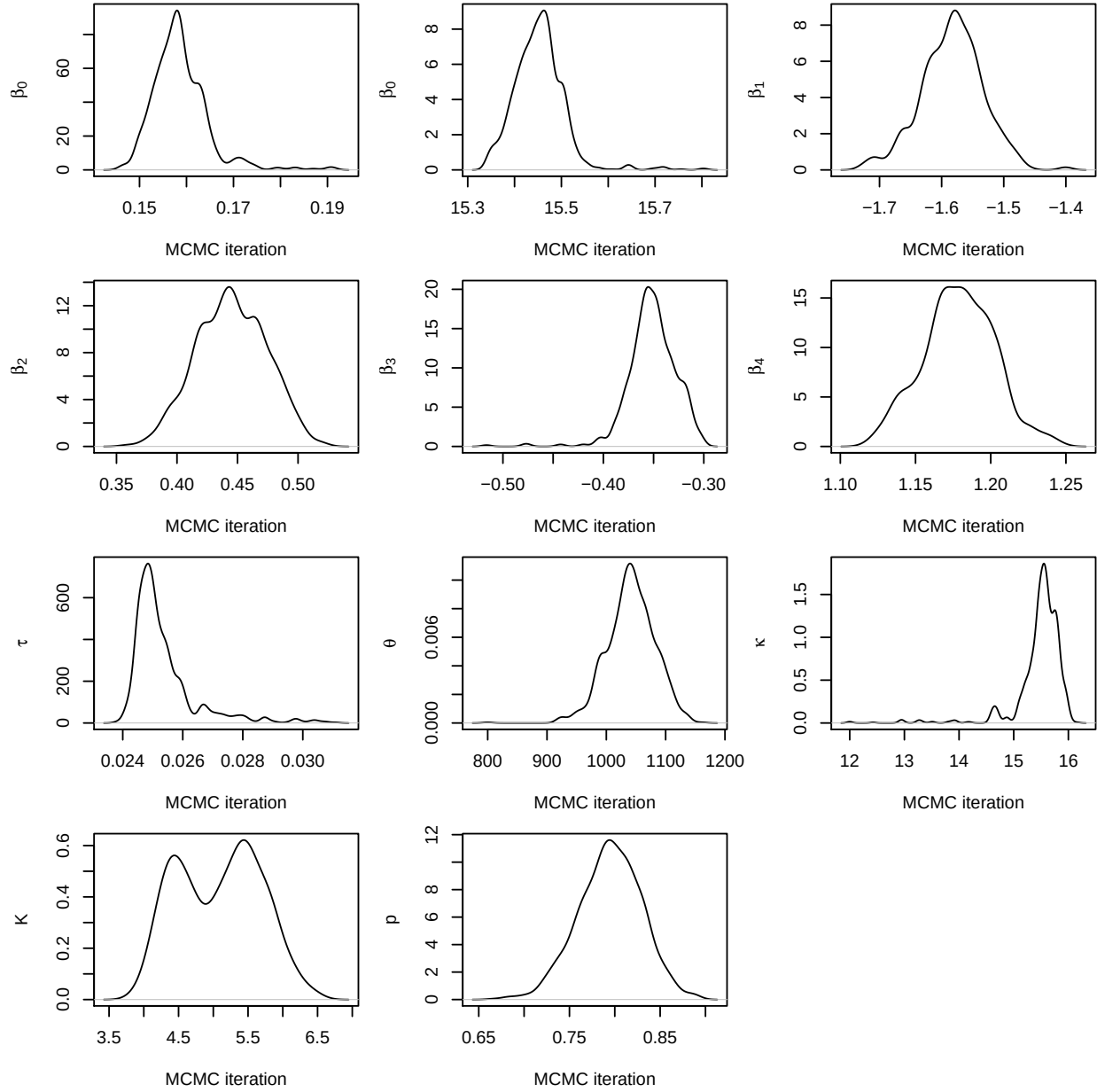


Figure 10: Density plot of MCMC values sampled from each parameter's full-conditional posterior distribution..

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)
```

```

(status=max(as.numeric(list.files(path=paste0(getwd(), '/progress')))))
n.iter=status
N.samps=do.call('rbind',N)

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)
beta=do.call('rbind',beta)
m.beta=apply(beta,2,mean)
beta.lq=apply(beta,2,quantile,.025)
beta.uq=apply(beta,2,quantile,.975)
tab.est2=rbind(tab.est,cbind(beta.lq,m.beta,beta.uq))
tau=do.call('rbind',tau)
m.tau=mean(tau)
tau.lq=quantile(tau,.025)
tau.uq=quantile(tau,.975)
tab.est3=rbind(tab.est2,c(tau.lq,m.tau,tau.uq))
theta=do.call('rbind',theta)
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))
kappa=do.call('rbind',kappa)
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))
p=do.call('rbind',p)
m.p=mean(p[,1])
p.lq=quantile(p[,1],.025)
p.uq=quantile(p[,1],.975)
tab.est6=rbind(tab.est5,c(p.lq,m.p,p.uq))
for(i in 2:ncol(p)){
  m.p=mean(p[,i])
  p.lq=quantile(p[,i],.025)
  p.uq=quantile(p[,i],.975)
  tab.est6=rbind(tab.est6,c(p.lq,m.p,p.uq))
}
colnames(tab.est6)=c("Lower Bound","Mean","Upper Bound")
rownames(tab.est6)=c("$\\sum_{i=1}^{q_s}n_{i,1993}$",
"$\\sum_{i=1}^{q_s}n_{i,1994}$",
"$\\sum_{i=1}^{q_s}n_{i,1995}$",

```

```

"$\\sum_{i=1}^{q_s}n_{i,1996}$",
"$\\sum_{i=1}^{q_s}n_{i,1997}$",
"$\\sum_{i=1}^{q_s}n_{i,1998}$",
"$\\sum_{i=1}^{q_s}n_{i,1999}$",
"$\\sum_{i=1}^{q_s}n_{i,2000}$",
"$\\sum_{i=1}^{q_s}n_{i,2001}$",
"$\\sum_{i=1}^{q_s}n_{i,2002}$",
"$\\sum_{i=1}^{q_s}n_{i,2003}$",
"$\\sum_{i=1}^{q_s}n_{i,2004}$",
"$\\sum_{i=1}^{q_s}n_{i,2005}$",
"$\\sum_{i=1}^{q_s}n_{i,2006}$",
"$\\sum_{i=1}^{q_s}n_{i,2007}$",
"$\\sum_{i=1}^{q_s}n_{i,2008}$",
"$\\sum_{i=1}^{q_s}n_{i,2009}$",
"$\\sum_{i=1}^{q_s}n_{i,2010}$",
"$\\sum_{i=1}^{q_s}n_{i,2011}$",
"$\\sum_{i=1}^{q_s}n_{i,2012}$",
"$\\sum_{i=1}^{q_s}n_{i,2013}$",
"$\\sum_{i=1}^{q_s}n_{i,2014}$",
"$\\sum_{i=1}^{q_s}n_{i,2015}$",
"$\\sum_{i=1}^{q_s}n_{i,2016}$",
"$\\sum_{i=1}^{q_s}n_{i,2017}$",
"$\\sum_{i=1}^{q_s}n_{i,2018}$",
"$\\sum_{i=1}^{q_s}n_{i,2019}$",
"$\\beta_0$", "$\\beta_1$", "$\\beta_2$",
"$\\beta_3$", "$\\beta_4$", "$\\alpha$",
"$\\theta$", "$\\kappa$", "p_1999", 'p_2000',
'p_2001', 'p_2002', 'p_2003', 'p_2004', 'p_2006',
'p_2012', 'p_2017', 'p_2018')
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.

```
save(N.ppd,file=~/.ppd.RData)
```

11 References

- Lu, X., P.J. Williams, M.B. Hooten, J.A. Powell, J.N. Womble, M.R. Bower. 2019. Nonlinear reaction–diffusion process models improve inference for population dynamics. *Environmetrics* e2604.
- Williams, P.J., M.B. Hooten, J.N. Womble, G.G. Esslinger, M.R. Bower, and T.J. Hefley. 2017a. An integrated data model to estimate spatio-temporal occupancy, abundance, and colonization dynamics. *Ecology* 98:328-336.
- Williams, P.J., M.B. Hooten, J.N. Womble, and M.R. Bower. 2017b. Estimating occupancy and abundance using aerial images with imperfect detection. *Methods in Ecology and Evolution* 8:1679-1689. DOI: 10.1111/2040-210X.12815.
- Williams, P.J., M.B. Hooten, J.N. Womble, G.G. Esslinger, and M.R. Bower. 2018. Monitoring dynamics spatio-temporal ecological processes optimally. *Ecology* 99:524-535.
- Williams, P.J., M.B. Hooten, G.G. Esslinger, J.N. Womble, J.L. Bodkin, and M.R. Bower. 2019. The rise of an apex predator following deglaciation. *Diversity and Distributions* 25:895-908.