Supplementary Document 1. R scripts to perform spatial point process analysis and covariate processing.

```
##### Script to Perform Spatial Point Process Analysis Presented in:

##### Sympatry and resource partitioning between the largest krill consumers around the Antarctic Peninsula
##### Ari S. Friedlaender1*, Trevor Joyce2, John W. Durban3, David W. Johnston4, Andrew J. Read4,
##### Douglas P. Nowacek4#, Jeremy A. Goldbogen5, and Nick Gales6

##### Elements of this analysis have been adapted from a script developed by Devin Johnson
##### for analyses presented in:
##### Hooten et al. 2017. Animal movement: statistical models for telemetry data

##### Script developed by Trevor Joyce
##### Version 3.0 (May 6, 2021)



############    TAGS    ############

#Import TAGS from ARGOSPRD.mdb queries
TAGS=read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
              "1_2016_Antarctic Whale Tracking/Data/",
              "Deployment Table.csv",sep = ""),stringsAsFactors=F)

#read date-time stamp from character string and reformat as POSIXct date time
TAGS$Deploy.date = as.POSIXct(TAGS$Deploy.date,format="%m/%d/%y %H:%M",tz="GMT")

############    ARGOS    ############

ARGOS = data.frame()

#### Import killer whale ARGOS data from .diag files provided by John (data on WC portal is not complete)
#### both were tagged in the pre-2012 field seasons (this data has been concatenated and seconds have been lost from time stamps)

for(i in list.files(paste("/Users/trevor.joyce/Grad School/Research/",
                  "1_2016_Antarctic Whale Tracking/Data/",
                  "Killer Whale Data/DIAG_kalman filter", sep = ""))[1:2])
{ARGOS_temp=read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
                  "1_2016_Antarctic Whale Tracking/Data/",
```

```
                    "Killer Whale Data/DIAG_kalman filter/",i,sep = ""),stringsAsFactors = F)

#rename columns to match the structure of Wildlife Computers Location.csv output files
colnames(ARGOS_temp)[which(colnames(ARGOS_temp)%in%c("PTT","Location.date","Location
.class",
                              "Semi.major.axis","Semi.minor.axis",
                              "Ellipse.orientation"))] = c("Ptt","Date","Quality",
                                         "Error.Semi.major.axis",
                                         "Error.Semi.minor.axis",
                                         "Error.Ellipse.orientation")
#assign a species id column (SPP) and Comment column where missing
ARGOS_temp$SPP = ""
ARGOS_temp$Comment = ""
#read date-time stamp from character string and reformat as POSIXct date time
ARGOS_temp$Date = as.POSIXct(strptime(ARGOS_temp$Date,format="%m/%d/%Y
%H:%M",tz="GMT"))

ARGOS_temp = ARGOS_temp[,c("Ptt","SPP","Date","Quality","Latitude","Longitude",
                    "Error.radius","Error.Semi.major.axis",
                    "Error.Semi.minor.axis","Error.Ellipse.orientation",
                    "Comment")]

#remove rows where location or time-stamp information is missing
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Latitude),]
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Date),]

#append rows to overall ARGOS data.frame
ARGOS = rbind(ARGOS,ARGOS_temp)
}


#### Import killer whale ARGOS data from .diag files provided by John (data on WC portal is not
complete)
#### both were tagged in the 2012-present field seasons

for(i in list.files(paste("/Users/trevor.joyce/Grad School/Research/",
                   "1_2016_Antarctic Whale Tracking/Data/",
                   "Killer Whale Data/DIAG_kalman filter", sep = ""))[-c(1:2)])
{ARGOS_temp=try(read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
                     "1_2016_Antarctic Whale Tracking/Data/",
                     "Killer Whale Data/DIAG_kalman filter/",i,sep = ""),stringsAsFactors = F))

#handle an error (duplicated row names) that arises from inconsistent formatting of Kalman
filter DIAG message files
```

```
if(class(ARGOS_temp)=="try-error")
{ARGOS_temp=read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Data/",
                    "Killer Whale Data/DIAG_kalman filter/",i,sep = ""),stringsAsFactors =
F,row.names = NULL)


#shift column names over by one (column names erroneously shifted by row.names = NULL
above)
colnames(ARGOS_temp) = c(colnames(ARGOS_temp)[-1],"X32")}


#handle an error (duplicated row names) that arises from inconsistent formatting of Kalman
filter DIAG message files
if(nrow(ARGOS_temp)==1)
{ARGOS_temp=read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Data/",
                    "Killer Whale Data/DIAG_kalman filter/",i,sep = ""),stringsAsFactors =
F,row.names = NULL)


#shift column names over by one (column names erroneously shifted by row.names = NULL
above)
colnames(ARGOS_temp) = c(colnames(ARGOS_temp)[-1],"X32")}

#rename columns to match the structure of Wildlife Computers Location.csv output files
colnames(ARGOS_temp)[which(colnames(ARGOS_temp)%in%c("PTT","Location.date","Location
.class",
                                   "Semi.major.axis","Semi.minor.axis",
                                   "Ellipse.orientation"))] = c("Ptt","Date","Quality",
                                                   "Error.Semi.major.axis",
                                                   "Error.Semi.minor.axis",
                                                   "Error.Ellipse.orientation")
#assign a species id column (SPP) and Comment column where missing
ARGOS_temp$SPP = ""
ARGOS_temp$Comment = ""

#read date-time stamp from character string and reformat as POSIXct date time
ARGOS_temp$Date = as.POSIXct(ARGOS_temp$Date,format="%Y/%m/%d
%H:%M:%S",tz="GMT")

#select a unified set of columns that are consistently found in all raw data formats
ARGOS_temp = ARGOS_temp[,c("Ptt","SPP","Date","Quality","Latitude","Longitude",
                    "Error.radius","Error.Semi.major.axis",
```

```
                    "Error.Semi.minor.axis","Error.Ellipse.orientation",
                    "Comment")]

#remove rows where location or time-stamp information is missing
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Latitude),]
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Date),]

#append rows to overall ARGOS data.frame
ARGOS = rbind(ARGOS,ARGOS_temp)
}

#remove non-Antarctic data included in DIAG Kalman folder
ARGOS = ARGOS[ARGOS$Ptt%in%TAGS$PTT,]

#assign species code (SPP) from TAGS to unlabeled datarows from DIAG Kalman folder
for(i in unique(ARGOS$Ptt))
{ARGOS[ARGOS$Ptt == i,c("SPP")] = TAGS[TAGS$PTT == i,c("SPP")]}


#### Import O. orca (KW-C) ARGOS data from -Location.csv files provided by Giancarlo and
Enrico
#### these are LIMPET tags) deployed by Giancarlo and Enrico in the 2014-15 field season
for(i in c(143823, 143824, 143825, 143826, 143828, 143830,
        143831, 143832, 143833, 143834))
{ARGOS_temp=read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Data/Killer Whale Data/",
                    i,"/",i,"-Locations.csv",sep = ""),stringsAsFactors = F)

#assign a species id column (SPP) and Comment column where missing
ARGOS_temp$SPP = "KWC"

#read date-time stamp from character string and reformat as POSIXct date time
ARGOS_temp$Date = as.POSIXct(ARGOS_temp$Date,format="%H:%M:%S %d-%b-
%Y",tz="GMT")

#select a unified set of columns that are consistently found in all raw data formats
ARGOS_temp = ARGOS_temp[,c("Ptt","SPP","Date","Quality","Latitude","Longitude",
                    "Error.radius","Error.Semi.major.axis",
                    "Error.Semi.minor.axis","Error.Ellipse.orientation",
                    "Comment")]

#remove rows where location or time-stamp information is missing
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Latitude),]
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Date),]
```

```
#append rows to overall ARGOS data.frame
ARGOS = rbind(ARGOS,ARGOS_temp)
}


#### Import B. bonarensis ARGOS data from WC portal -Location.csv files provided by Gin
#### these are implant (as opposed to LIMPET tags) deployed by Ari's group 2012-13 and 2016-
17 field seasons
for(i in c(112731, 112732, 112733, 112734, 112736, 112741,
      112745, 112747, 112748, 112750, 166115, 166116, 166118))
{ARGOS_temp=read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
              "1_2016_Antarctic Whale Tracking/Data/Minke Data/",i,"/",
              i,"-Locations.csv",sep = ""),stringsAsFactors = F)

#assign a species id column (SPP) and Comment column where missing
ARGOS_temp$SPP = "Bb"

#read date-time stamp from character string and reformat as POSIXct date time
ARGOS_temp$Date = as.POSIXct(ARGOS_temp$Date,format="%H:%M:%S %d-%b-
%Y",tz="GMT")

#select a unified set of columns that are consistently found in all raw data formats
ARGOS_temp = ARGOS_temp[,c("Ptt","SPP","Date","Quality","Latitude","Longitude",
              "Error.radius","Error.Semi.major.axis",
              "Error.Semi.minor.axis","Error.Ellipse.orientation",
              "Comment")]

#remove rows where location or time-stamp information is missing
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Latitude),]
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Date),]

#append rows to overall ARGOS data.frame
ARGOS = rbind(ARGOS,ARGOS_temp)
}

#### Import B. bonarensis ARGOS data from .diag files provided by John (data on WC portal is
not complete)
#### both were tagged in the 2013-14 field season
for(i in c(131101,131107))
{#handle an error (duplicated row names) that arises from inconsistent formatting of Kalman
filter DIAG message files
  ARGOS_temp = read.csv(paste("/Users/trevor.joyce/Grad School/Research/1_2016_Antarctic
Whale Tracking",
```

```
                "/Data/Minke Data/",i,"/",i,".csv",sep = ""),header = T,sep =
";",stringsAsFactors = F,row.names = NULL)

 #shift column names over by one (column names erroneously shifted by row.names = NULL
above)
 colnames(ARGOS_temp) = c(colnames(ARGOS_temp)[-1],"X32")

 #rename columns to match the structure of Wildlife Computers Location.csv output files

colnames(ARGOS_temp)[which(colnames(ARGOS_temp)%in%c("PTT","Location.date","Location
.class",
                             "Semi.major.axis","Semi.minor.axis",
                             "Ellipse.orientation"))] = c("Ptt","Date","Quality",
                                          "Error.Semi.major.axis",
                                          "Error.Semi.minor.axis",
                                          "Error.Ellipse.orientation")
 #assign a species id column (SPP) and Comment column where missing
 ARGOS_temp$SPP = "Bb"
 ARGOS_temp$Comment = ""

 #read date-time stamp from character string and reformat as POSIXct date time
 ARGOS_temp$Date = as.POSIXct(ARGOS_temp$Date,format="%Y/%m/%d
%H:%M:%S",tz="GMT")

 #select a unified set of columns that are consistently found in all raw data formats
 ARGOS_temp = ARGOS_temp[,c("Ptt","SPP","Date","Quality","Latitude","Longitude",
              "Error.radius","Error.Semi.major.axis",
              "Error.Semi.minor.axis","Error.Ellipse.orientation",
              "Comment")]

 #remove rows where location or time-stamp information is missing
 ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Latitude),]
 ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Date),]

 #append rows to overall ARGOS data.frame
 ARGOS = rbind(ARGOS,ARGOS_temp)
}

#### Import B. bonarensis ARGOS data from WC portal -Location.csv files provided by Gin
#### this includes the following PTT numbers (131108, 131117, 131118, 131120, 154184)
#### these are LIMPET tags deployed by John and Bob 2014-15 field season and by Ari's group
2015-16 field season
```

```
ARGOS_temp=read.csv(paste("/Users/trevor.joyce/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/Minke Data/",
              "WAP_minke_2015-16.csv",sep = ""),stringsAsFactors = F)

#rename columns to match the structure of Wildlife Computers Location.csv output files
colnames(ARGOS_temp)[which(colnames(ARGOS_temp)%in%c("Platform.ID.No.","Loc..quality"
,
                              "Loc..date","Semi.major.axis",
                              "Semi.minor.axis",
                              "Ellipse.orientation"))] = c("Ptt","Quality","Date",
                                             "Error.Semi.major.axis",
                                             "Error.Semi.minor.axis",
                                             "Error.Ellipse.orientation")

#assign a species id column (SPP) and Comment column where missing
ARGOS_temp$SPP = "Bb"
ARGOS_temp$Comment = ""

#read date-time stamp from character string and reformat as POSIXct date time
ARGOS_temp$Date = as.POSIXct(ARGOS_temp$Date,format="%Y-%m-%d
%H:%M:%S",tz="GMT")

#select a unified set of columns that are consistently found in all raw data formats
ARGOS_temp = ARGOS_temp[,c("Ptt","SPP","Date","Quality","Latitude","Longitude",
              "Error.radius","Error.Semi.major.axis",
              "Error.Semi.minor.axis","Error.Ellipse.orientation",
              "Comment")]

#remove rows where location or time-stamp information is missing
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Latitude),]
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Date),]

#append rows to overall ARGOS data.frame
ARGOS = rbind(ARGOS,ARGOS_temp)

#### Import M. novaeangliae ARGOS data from WC portal -Location.csv files provided by Gin
#### these are LIMPET tags deployed by Ari's group 2015-16 and 2016-17 field seasons
for(i in c(131111,131115,131116,131127,131128,131130,131132,131133,
      131134,131136,131142,131156,131158,131159,131162,154187))
{ARGOS_temp=read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
              "1_2016_Antarctic Whale Tracking/Data/Humpback Data/",i,"/",
              i,"-Locations.csv",sep = ""),stringsAsFactors = F)

#assign a species id column (SPP) and Comment column where missing
```

```
ARGOS_temp$SPP = "Mn"

#read date-time stamp from character string and reformat as POSIXct date time
ARGOS_temp$Date = as.POSIXct(ARGOS_temp$Date,format="%H:%M:%S %d-%b-
%Y",tz="GMT")

#select a unified set of columns that are consistently found in all raw data formats
ARGOS_temp = ARGOS_temp[,c("Ptt","SPP","Date","Quality","Latitude","Longitude",
                "Error.radius","Error.Semi.major.axis",
                "Error.Semi.minor.axis","Error.Ellipse.orientation",
                "Comment")]

#remove rows where location or time-stamp information is missing
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Latitude),]
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Date),]

#append rows to overall ARGOS data.frame
ARGOS = rbind(ARGOS,ARGOS_temp)
}

remove(ARGOS_temp)


#### Import additional M. novaeangliae ARGOS data from files provided by Ben Weinstein
#### these are LIMPET tags deployed by Ari's group 2012 - 2016 field seasons
#### (some overlap with individual files from Gin so need to remove duplicates)
ARGOS_temp=read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
                "1_2016_Antarctic Whale Tracking/Data/Humpback Data/",
                "Humpback.csv",sep = ""),
            stringsAsFactors = F)

colnames(ARGOS_temp) <- c("X","Date","Time","Longitude","Latitude","Quality",
                "Error.Ellipse.orientation","Error.Semi.major.axis",
                "Error.Semi.minor.axis","Ptt","Ptt.1","Common.Name","Species")

#assign a species id column (SPP) and Comment column where missing
ARGOS_temp$SPP = "Mn"

#read date-time stamp from character string and reformat as POSIXct date time
ARGOS_temp$Date <- paste(ARGOS_temp$Date, ARGOS_temp$Time, sep =" ")
ARGOS_temp$Date = as.POSIXct(ARGOS_temp$Date,format="%m/%d/%y
%H:%M:%S",tz="GMT")

#Add columns not included in this dataset
```

```
ARGOS_temp$Error.radius <- NA
ARGOS_temp$Comment <- NA


#remove records that duplicate information from PTTs directly imported from Location.csv files above
ARGOS_temp =
ARGOS_temp[!ARGOS_temp$Ptt%in%c(131111,131115,131116,131127,131128,131130,131132,131133,
                              131134,131136,131142,131156,131158,131159,131162,154187),]

#remove rows where location or time-stamp information is missing
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Latitude),]
ARGOS_temp = ARGOS_temp[!is.na(ARGOS_temp$Date),]

#select a unified set of columns that are consistently found in all raw data formats
ARGOS_temp = ARGOS_temp[,c("Ptt","SPP","Date","Quality","Latitude","Longitude",
                  "Error.radius","Error.Semi.major.axis",
                  "Error.Semi.minor.axis","Error.Ellipse.orientation",
                  "Comment")]

#append rows to overall ARGOS data.frame
ARGOS = rbind(ARGOS,ARGOS_temp)

#Clean-up
remove(ARGOS_temp,i)

#Shift column names upper case and adjust DATE column name to match subsequent CTCRW code
colnames(ARGOS) = toupper(colnames(ARGOS))
colnames(ARGOS)[which(colnames(ARGOS)%in%c("DATE"))] <- c("DATE_TIME")

#Order observations by PTT and then DATE_TIME (deals with mixed and unordered data from DIAG files)
ARGOS = ARGOS[order(ARGOS$PTT,ARGOS$DATE_TIME),]

# Flag records from prior to deployment date
# (these have been rounded to day because of inconsistent reporting of deployment times)
for(i in unique(ARGOS$PTT))
{ARGOS[ARGOS$PTT==i & ARGOS$DATE_TIME<lubridate::floor_date(TAGS[TAGS$PTT==i
,"Deploy.date"]),"DEPLOY"] <- "PRE-DEPLOYMENT"
ARGOS[ARGOS$PTT==i & ARGOS$DATE_TIME>lubridate::floor_date(TAGS[TAGS$PTT==i
,"Deploy.date"]),"DEPLOY"] <- "DEPLOYMENT"}
```

```
#assign a tagging REGION from TAGS to ARGOS records
for(i in unique(ARGOS$PTT))
{ARGOS[ARGOS$PTT == i,c("REGION")] = TAGS[TAGS$PTT == i,c("Region")]}

#Apply a pre-modeling Speed-Distance-Angle-filter (Freitas et al. 2008) to extract "erroneous"
values
#(vmax set to unrealistically high 28 m/s to remove only egregiously outlying points)
ARGOS$FILTERED = NA

for(i in unique(ARGOS[ARGOS$DEPLOY=="DEPLOYMENT","PTT"]))
{ARGOS[ARGOS$PTT==i &
      ARGOS$DEPLOY=="DEPLOYMENT","FILTERED"] <- argosfilter::sdafilter(lat =
ARGOS[ARGOS$PTT==i &

ARGOS$DEPLOY=="DEPLOYMENT","LATITUDE"],
                                                  lon = ARGOS[ARGOS$PTT==i &

ARGOS$DEPLOY=="DEPLOYMENT","LONGITUDE"],
                                                  dtime = ARGOS[ARGOS$PTT==i &

ARGOS$DEPLOY=="DEPLOYMENT","DATE_TIME"],
                                                  lc = ARGOS[ARGOS$PTT==i &
                                                         ARGOS$DEPLOY=="DEPLOYMENT","QUALITY"],
                                                  vmax = 28)}

ARGOS[,c("ln.sd.x","ln.sd.y","error.corr","diag.check")] = NA

# ARGOS[!is.na(ARGOS$ERROR.SEMI.MAJOR.AXIS) & !is.na(ARGOS$ERROR.SEMI.MINOR.AXIS)
&
#      ARGOS$ERROR.SEMI.MAJOR.AXIS==0 & ARGOS$ERROR.SEMI.MINOR.AXIS==0 ,
c("ERROR.SEMI.MAJOR.AXIS",
#                                                  "ERROR.SEMI.MINOR.AXIS",
#                                                  "ERROR.ELLIPSE.ORIENTATION")] = NA


#### Calculate the variance-covariance matrix used in CTCRW models for ARGOS
#### records that contain Kalman filter error ellipse information

for(i in unique(ARGOS[!is.na(ARGOS$ERROR.SEMI.MAJOR.AXIS),"PTT"]))
{## place Kalman filter error ellipse information in a model matrix that can be used by
 ## crawl::argosDiag2Cov to back out a variance-covariance matrix used in CTCRW model
 ## (rows with ERROR.SEMI.MAJOR.AXIS and ERROR.SEMI.MINOR.AXIS values of 0 removed
 ## to avoid "faulty Argos error correlation" error)
```

```
  DIAG_DATA_temp = stats::model.matrix(~ ERROR.SEMI.MAJOR.AXIS +
ERROR.SEMI.MINOR.AXIS + ERROR.ELLIPSE.ORIENTATION,
                              model.frame(ARGOS[ARGOS$PTT == i
                                           & !is.na(ARGOS$ERROR.SEMI.MAJOR.AXIS)
                                           & !is.na(ARGOS$ERROR.SEMI.MINOR.AXIS)
                                           & ARGOS$ERROR.SEMI.MAJOR.AXIS!=0
                                           & ARGOS$ERROR.SEMI.MINOR.AXIS!=0,], na.action =
na.pass))[,-1]

  ## calculate variance-covariance matrix used in CTCRW model
  DIAG_DATA_temp =
crawl::argosDiag2Cov(DIAG_DATA_temp[,1],DIAG_DATA_temp[,2],DIAG_DATA_temp[,3])

  ## add variance and correlation information to ARGOS
  ARGOS[ARGOS$PTT == i
      & !is.na(ARGOS$ERROR.SEMI.MAJOR.AXIS)
      & !is.na(ARGOS$ERROR.SEMI.MINOR.AXIS)
      & ARGOS$ERROR.SEMI.MAJOR.AXIS!=0
      & ARGOS$ERROR.SEMI.MINOR.AXIS!=0,c("ln.sd.x","ln.sd.y","error.corr","diag.check")] =
DIAG_DATA_temp

}

remove(DIAG_DATA_temp)

## Preserve LONGITUDE and LATITUDE in geographic coordinates that will not be overwritten
by CTCRW
ARGOS$LONGITUDE_RAW <- ARGOS$LONGITUDE
ARGOS$LATITUDE_RAW <- ARGOS$LATITUDE

## Preserve LONGITUDE and LATITUDE in geographic coordinates before transformation to
projected coordinates
ARGOS$LONGITUDE_GCS <- ARGOS$LONGITUDE
ARGOS$LATITUDE_GCS <- ARGOS$LATITUDE

## Save as set of coordinates in 0:360 longitude for plotting accross the dateline
ARGOS$LONGITUDE_GCS_360 <-
ifelse(ARGOS$LONGITUDE_GCS<0,ARGOS$LONGITUDE_GCS+360,ARGOS$LONGITUDE_GCS)
ARGOS$LATITUDE_GCS_360 <- ARGOS$LATITUDE_GCS


## convert to sp object and project to standard GPS/Argos projection
library(sp)
coordinates(ARGOS) <- ~ LONGITUDE + LATITUDE
```

```
proj4string(ARGOS) <- "+init=epsg:4326"


## separate West Antarctic Peninsular records from Ross Sea records
## which will be projected separately using locally specific projections
ARGOS_WAP_temp = ARGOS[ARGOS$REGION %in% c("WAP","Weddell"),]

## Create a custom Projected Coordinate system projection for movement modeling
## specific to the bounding dimensions of each study area
PROJ_WAP= paste("+proj=aea",
            " +lat_1=",-60,
            " +lat_2=",-70,
            " +lat_0=",-65,
            " +lon_0=",-60,
            "+x_0=0 +y_0=0 +ellps=GRS80 +datum=NAD83 +units=m",sep="")

## Transform coordinates to new custom projection
ARGOS_WAP_temp <- spTransform(ARGOS_WAP_temp, CRS(PROJ_WAP))

## separate Ross Sea records from West Antarctic Peninsular records
## which will be projected separately using locally specific projections
ARGOS_ROSS_temp = ARGOS[ARGOS$REGION == "Ross",]

## Create a custom Projected Coordinate system projection for movement modeling
## specific to the bounding dimensions of each study area
PROJ_ROSS= paste("+proj=aea",
            " +lat_1=",-70,
            " +lat_2=",-80,
            " +lat_0=",-75,
            " +lon_0=",165,
            "+x_0=0 +y_0=0 +ellps=GRS80 +datum=NAD83 +units=m",sep="")

## Transform coordinates to new custom projection
ARGOS_ROSS_temp <- spTransform(ARGOS_ROSS_temp, CRS(PROJ_ROSS))

## convert from a SpatialPointsDataFrame back to a dataframe (expected input for crwMLE)
ARGOS_WAP_temp <- as.data.frame(ARGOS_WAP_temp)
ARGOS_ROSS_temp <- as.data.frame(ARGOS_ROSS_temp)


## bring the two parts back together and clean up
ARGOS = rbind(ARGOS_WAP_temp,ARGOS_ROSS_temp)

remove(ARGOS_WAP_temp,ARGOS_ROSS_temp)
```

```
## create a time (hours) to next ARGOS fix column (TIME_DIFF) to identify duty cycle periods
for(i in unique(ARGOS$PTT))
{ARGOS[ARGOS$PTT == i,"TIME_DIFF"] <- as.numeric(difftime(time1 = ARGOS[ARGOS$PTT ==
i,"DATE_TIME"][c(2:nrow(ARGOS[ARGOS$PTT == i,]),
                                                        nrow(ARGOS[ARGOS$PTT == i,]))],
                              time2 = ARGOS[ARGOS$PTT ==
i,"DATE_TIME"][1:nrow(ARGOS[ARGOS$PTT == i,])],
                              units = "hours"))}

## Update TAGS data.frame with ARGOS transmission information

for(i in unique(ARGOS$PTT)){

 ## add DATE_TIME of first ARGOS transmission
 if(ARGOS[ARGOS$PTT == i,"DATE_TIME"][1] > TAGS[TAGS$PTT == i,"Deploy.date"])
 {TAGS[TAGS$PTT == i,"ARGOS_START"] <- ARGOS[ARGOS$PTT == i,"DATE_TIME"][1]}
 else {TAGS[TAGS$PTT == i,"ARGOS_START"] <- TAGS[TAGS$PTT == i,"Deploy.date"]}

 ## add DATE_TIME of last ARGOS transmission
 TAGS[TAGS$PTT == i,"ARGOS_END"] <- rev(ARGOS[ARGOS$PTT == i,"DATE_TIME"])[1]

 ## add start of duty cycling TAGS data.frame
 if(length(which(ARGOS[ARGOS$PTT == i,"TIME_DIFF"]>24))>0)
 {TAGS[TAGS$PTT == i,"ARGOS_START_DUTY"] <- ARGOS[ARGOS$PTT == i &
ARGOS$TIME_DIFF>24
                          & ARGOS$DATE_TIME > TAGS[TAGS$PTT ==
i,"Deploy.date"],"DATE_TIME"][1]}

 ## add start of migration to TAGS data.frame
 if(length(which(ARGOS[ARGOS$PTT == i,"REGION"] %in% c("WAP","Weddell") &
ARGOS[ARGOS$PTT == i,"LATITUDE_GCS"]> -60))>0)
 {TAGS[TAGS$PTT == i,"ARGOS_START_MIG"] <- ARGOS[ARGOS$PTT == i &
ARGOS$LATITUDE_GCS> -60
                          & ARGOS$DATE_TIME > TAGS[TAGS$PTT ==
i,"Deploy.date"],"DATE_TIME"][1]}

 if(length(which(ARGOS[ARGOS$PTT == i,"REGION"]=="Ross" & ARGOS[ARGOS$PTT ==
i,"LATITUDE_GCS"]> -71))>0)
 {TAGS[TAGS$PTT == i,"ARGOS_START_MIG"] <- ARGOS[ARGOS$PTT == i &
ARGOS$LATITUDE_GCS> -71
                          & ARGOS$DATE_TIME > TAGS[TAGS$PTT ==
i,"Deploy.date"],"DATE_TIME"][1]}
```

```
}

TAGS$ARGOS_START <- crawl::intToPOSIX(TAGS$ARGOS_START)
TAGS$ARGOS_END <- crawl::intToPOSIX(TAGS$ARGOS_END)
TAGS$ARGOS_START_DUTY <- crawl::intToPOSIX(TAGS$ARGOS_START_DUTY)
TAGS$ARGOS_START_MIG <- crawl::intToPOSIX(TAGS$ARGOS_START_MIG)


# Create a date corresponding to the format of raster names (timestamps)
ARGOS$YEAR=as.numeric(as.character(lubridate::year(ARGOS$DATE_TIME)))
ARGOS$MONTH=sprintf("%02d",as.numeric(lubridate::month(ARGOS$DATE_TIME)))
ARGOS$DAY=sprintf("%02d",as.numeric(as.character(lubridate::day(ARGOS$DATE_TIME))))
ARGOS$RAST_DATE=paste("X",ARGOS$YEAR,"_",ARGOS$MONTH,"_",ARGOS$DAY,sep="")


Sys.setenv(TZ="GMT")
ARGOS$SUNRISE=maptools::sunriset(as.matrix(cbind(ARGOS$LONGITUDE_GCS,ARGOS$LATITU
DE_GCS)),
                    as.POSIXct(ARGOS[,"DATE_TIME"],tz="GMT"),
                    direction="sunrise", POSIXct.out=TRUE)$time
#restore sunset to chron times object
ARGOS$SUNSET=maptools::sunriset(as.matrix(cbind(ARGOS$LONGITUDE_GCS,ARGOS$LATITUD
E_GCS)),
                   as.POSIXct(ARGOS[,"DATE_TIME"],tz="GMT"),
                   direction="sunset", POSIXct.out=TRUE)$time
ARGOS$SUN_ELEV=maptools::solarpos(as.matrix(cbind(ARGOS$LONGITUDE_GCS,ARGOS$LATIT
UDE_GCS)),
                    as.POSIXct(ARGOS[,"DATE_TIME"],tz="GMT"),
                    direction="sunset", POSIXct.out=TRUE)[,2]
ARGOS$SUN_AZI=maptools::solarpos(as.matrix(cbind(ARGOS$LONGITUDE_GCS,ARGOS$LATITU
DE_GCS)),
                    as.POSIXct(ARGOS[,"DATE_TIME"],tz="GMT"),
                    direction="sunset", POSIXct.out=TRUE)[,1]

###########   MIGRATION  ############

#Import manually scored migration timing landmarks (departure, turning, return, end)

MIGRATION = read.csv(paste("/Users/trevor.joyce/Grad School/Research/",
                 "1_2016_Antarctic Whale Tracking/Data/",
                 "MIGRATION_DATES.csv",sep = ""),stringsAsFactors=F)

MIGRATION$OUT_MIG= paste(MIGRATION$OUT_MIG_DATE,MIGRATION$OUT_MIG_TIME)
MIGRATION$TURN = paste(MIGRATION$TURN_DATE,MIGRATION$TURN_TIME)
```

```
MIGRATION$FURTHEST_N =
paste(MIGRATION$FURTHEST_N_DATE,MIGRATION$FURTHEST_N_TIME)
MIGRATION$IN_MIG = paste(MIGRATION$IN_MIG_DATE,MIGRATION$IN_MIG_TIME)
MIGRATION$END_MIG = paste(MIGRATION$END_MIG_DATE,MIGRATION$END_MIG_TIME)

MIGRATION$OUT_MIG = as.POSIXct(MIGRATION$OUT_MIG,format="%m/%d/%y
%H:%M:%S",tz="GMT")
MIGRATION$TURN = as.POSIXct(MIGRATION$TURN,format="%m/%d/%y
%H:%M:%S",tz="GMT")
MIGRATION$FURTHEST_N = as.POSIXct(MIGRATION$FURTHEST_N,format="%m/%d/%y
%H:%M:%S",tz="GMT")
MIGRATION$IN_MIG = as.POSIXct(MIGRATION$IN_MIG,format="%m/%d/%y
%H:%M:%S",tz="GMT")
MIGRATION$END_MIG = as.POSIXct(MIGRATION$END_MIG,format="%m/%d/%y
%H:%M:%S",tz="GMT")

MIGRATION = subset(MIGRATION,select = -c(OUT_MIG_DATE, OUT_MIG_TIME, TURN_DATE,
TURN_TIME,
                         FURTHEST_N_DATE, FURTHEST_N_TIME,
                         IN_MIG_DATE, IN_MIG_TIME, END_MIG_DATE, END_MIG_TIME))


for(i in unique(MIGRATION$PTT)){

  ## add DATE_TIME of first ARGOS transmission
  if(ARGOS[ARGOS$PTT == i,"DATE_TIME"][1] > TAGS[TAGS$PTT == i,"Deploy.date"])
  {MIGRATION[MIGRATION$PTT == i,"ARGOS_START"] <- ARGOS[ARGOS$PTT ==
i,"DATE_TIME"][1]}
  else {MIGRATION[MIGRATION$PTT == i,"ARGOS_START"] <- TAGS[TAGS$PTT ==
i,"Deploy.date"]}

  ## add DATE_TIME of last ARGOS transmission
  MIGRATION[MIGRATION$PTT == i,"ARGOS_END"] <- rev(ARGOS[ARGOS$PTT ==
i,"DATE_TIME"])[1]}

Sys.setenv(TZ='GMT')
MIGRATION$ARGOS_START = as.POSIXct(MIGRATION$ARGOS_START,origin = "1970-01-01"
,tz="GMT")
MIGRATION$ARGOS_END = as.POSIXct(MIGRATION$ARGOS_END,origin = "1970-01-01"
,tz="GMT")


for(i in 1:nrow(MIGRATION))
{MIGRATION$J_ARGOS_START[i]=as.numeric(base::julian(MIGRATION$ARGOS_START[i],
```

```
                                          origin =
as.POSIXct(paste(lubridate::year(MIGRATION$ARGOS_START[i]),
                                                               01,01,sep = "-"),
                                                          tz = "GMT")))+1} #Calculate Day of Year - doesn't work
unless looped because origin only first value in DATES frame
MIGRATION$J_OUT_MIG = NA
for(i in which(!is.na(MIGRATION$OUT_MIG)))
{MIGRATION$J_OUT_MIG[i]=as.numeric(base::julian(MIGRATION$OUT_MIG[i],
                                       origin = as.POSIXct(paste(lubridate::year(MIGRATION$OUT_MIG[i]),
                                                            01,01,sep = "-"),
                                                       tz = "GMT")))+1} #Calculate Day of Year - doesn't work
unless looped because origin only first value in DATES frame




MIGRATION$DURATION_PRE_MIG =
as.numeric(difftime(MIGRATION$OUT_MIG,MIGRATION$ARGOS_START,
                                        units = "days"))
MIGRATION$DURATION_OUT_MIG =
as.numeric(difftime(MIGRATION$FURTHEST_N,MIGRATION$OUT_MIG,
                                        units = "days"))
MIGRATION$DURATION_IN_MIG =
as.numeric(difftime(MIGRATION$END_MIG,MIGRATION$FURTHEST_N,
                                       units = "days"))
MIGRATION$DURATION_POST_MIG =
as.numeric(difftime(MIGRATION$ARGOS_END,MIGRATION$END_MIG,
                                        units = "days"))

MIGRATION[MIGRATION$PTT == 129722,"DURATION_PRE_MIG"][2] =
as.numeric(difftime(MIGRATION[MIGRATION$PTT == 129722,"OUT_MIG"][2],
                                                               MIGRATION[MIGRATION$PTT ==
129722,"END_MIG"][1],
                                                               units = "days"))
MIGRATION[MIGRATION$PTT == 129722,"DURATION_POST_MIG"][1] =
as.numeric(difftime(MIGRATION[MIGRATION$PTT == 129722,"OUT_MIG"][2],
                                                                MIGRATION[MIGRATION$PTT ==
129722,"END_MIG"][1],
                                                                units = "days"))


# Assign MIGRATION status based on manually scored
# migration timing landmarks (departure, turning, return, end)
```

```
ARGOS$MIGRATION = "PRE_MIG"

for(i in which(!is.na(MIGRATION$OUT_MIG)))
{ ARGOS[ARGOS$PTT == MIGRATION$PTT[i]
     & ARGOS$DATE_TIME >= MIGRATION$OUT_MIG[i],"MIGRATION"] = "OUT_MIG"

if(!is.na(MIGRATION$TURN[i]))
{ARGOS[ARGOS$PTT == MIGRATION$PTT[i]
    & ARGOS$DATE_TIME >= MIGRATION$TURN[i],"MIGRATION"] = "TURN_N"}

if(!is.na(MIGRATION$FURTHEST_N[i]))
{ARGOS[ARGOS$PTT == MIGRATION$PTT[i]
    & ARGOS$DATE_TIME >= MIGRATION$FURTHEST_N[i],"MIGRATION"] = "TURN_S"}

if(!is.na(MIGRATION$IN_MIG[i]))
{ARGOS[ARGOS$PTT == MIGRATION$PTT[i]
    & ARGOS$DATE_TIME >= MIGRATION$IN_MIG[i],"MIGRATION"] = "IN_MIG"}

if(!is.na(MIGRATION$END_MIG[i]))
{ARGOS[ARGOS$PTT == MIGRATION$PTT[i]
    & ARGOS$DATE_TIME >= MIGRATION$END_MIG[i],"MIGRATION"] = "POST_MIG"}
}


# for(i in 1:nrow(MIGRATION))
#
{MIGRATION$DISTANCE_PRE_MIG[i]=sum(ARGOS_CTCRW[ARGOS_CTCRW$PTT%in%MIGRATIO
N$PTT[i]
#                                 &
ARGOS_CTCRW$MIGRATION%in%c("PRE_MIG"),"DIST_GEO"],na.rm = T)}
# for(i in 1:nrow(MIGRATION))
#
{MIGRATION$DISTANCE_OUT_MIG[i]=sum(ARGOS_CTCRW[ARGOS_CTCRW$PTT%in%MIGRATI
ON$PTT[i]
#                                 &
ARGOS_CTCRW$MIGRATION%in%c("OUT_MIG"),"DIST_GEO"],na.rm = T)}
# for(i in 1:nrow(MIGRATION))
#
{MIGRATION$DISTANCE_TURN_N[i]=sum(ARGOS_CTCRW[ARGOS_CTCRW$PTT%in%MIGRATIO
N$PTT[i]
#                                &
ARGOS_CTCRW$MIGRATION%in%c("TURN_N"),"DIST_GEO"],na.rm = T)}
# for(i in 1:nrow(MIGRATION))
```

```
#
{MIGRATION$DISTANCE_TURN_S[i]=sum(ARGOS_CTCRW[ARGOS_CTCRW$PTT%in%MIGRATIO
N$PTT[i]
#                          &
ARGOS_CTCRW$MIGRATION%in%c("TURN_S"),"DIST_GEO"],na.rm = T)}
# for(i in 1:nrow(MIGRATION))
#
{MIGRATION$DISTANCE_IN_MIG[i]=sum(ARGOS_CTCRW[ARGOS_CTCRW$PTT%in%MIGRATION
$PTT[i]
#                          &
ARGOS_CTCRW$MIGRATION%in%c("IN_MIG"),"DIST_GEO"],na.rm = T)}
# for(i in 1:nrow(MIGRATION))
#
{MIGRATION$DISTANCE_POST_MIG[i]=sum(ARGOS_CTCRW[ARGOS_CTCRW$PTT%in%MIGRATI
ON$PTT[i]
#                           &
ARGOS_CTCRW$MIGRATION%in%c("POST_MIG"),"DIST_GEO"],na.rm = T)}
#
#
#


############    ARGOS_CTCRW_FIT    ############

# List to house CTCRW model fit results
ARGOS_CTCRW_FIT = list()

#### ELLIPSE: Fit CTCRW model on tags that have Kalman filter ellipse information

## Set input values for model in Johnson et al. (2008) Ecology 89:1208-1215
## Start values for theta come from the estimates in Johnson et al. (2008)

# Define which model parameters will be fixed a priori
# and which will be estimated (NA -> free)
FIX_PAR_ELLIPSE = c(1, #ln.sd.x: x location error parameter (known from Argos error ellipse
estimates so fixed at 1)
          1, #ln.sd.y: y location error parameter (known from Argos error ellipse estimates so
fixed at 1)
          NA, #ln sigma (intercept): velocity variance parameter (free parameter)
          NA) #ln beta (intercept): velocity correlation parameter (free parameter)

# Establish starting estimates for free parameters
THETA_ELLIPSE = c(log(1.5*60*60), #ln sigma: starting estimate for velocity variance parameter
based on a 1.5 m/s velocity (transformed into m/hr)
```

```
                -4) #ln beta: starting estimate for velocity correlation parameter (sigma of >=4 implies
essentially Brownian motion)

# Define a Laplace prior (moderately peaked) for CTCRW model parameters
LAP_PRIOR = function(PAR){-abs(PAR[2]-4)/1}

for(i in unique(ARGOS[!is.na(ARGOS$ln.sd.x) & ARGOS$DEPLOY=="DEPLOYMENT","PTT"]))
{# Define initial state (position and velocity) for model
  INIT_ELLIPSE = list(a=c(ARGOS[ARGOS$PTT == i
                          & !is.na(ARGOS$ln.sd.x)
                          & ARGOS$DEPLOY%in%"DEPLOYMENT"
                          & ARGOS$FILTERED%in%"not","LONGITUDE"][1],#starting estimate of initial x
position
                      0,#starting estimate of initial x velocity
                      ARGOS[ARGOS$PTT == i
                          & !is.na(ARGOS$ln.sd.x)
                          & ARGOS$DEPLOY%in%"DEPLOYMENT"
                          & ARGOS$FILTERED%in%"not","LATITUDE"][1],#starting estimate of initial y
position
                      0),#starting estimate of initial y velocity
                    P=diag(c(5000^2,
                          10^2,
                          5000^2,
                          10^2)))#variance-covariance matrix for the initial location and velocity

  print(paste(i,ARGOS[ARGOS$PTT == i,"SPP"][1]))

  # Set-up loop that will fit crwMLE
  # and then will keep trying to refit crwMLE up to 10 times to see if it's possible to
  # obtain a fit that does not contain model fitting errors ("try-errors") and
  # where none of the parameter estimate standard errors are NaN

  for(j in 1:10){

    if(class(ARGOS_CTCRW_FIT[[paste("CTCRW_FIT_",i,sep="")]]) == "NULL" |
      class(ARGOS_CTCRW_FIT[[paste("CTCRW_FIT_",i,sep="")]])=="try-error"|
      any(is.nan(ARGOS_CTCRW_FIT[[paste("CTCRW_FIT_",i,sep="")]]$se) == T)){


      # Fit CTCRW model
      ARGOS_CTCRW_FIT[[paste("CTCRW_FIT_",i,sep="")]] <- try(crawl::crwMLE(mov.model=~1,
                                                  err.model=list(x=~ln.sd.x-1,
                                                            y=~ln.sd.y-1,
                                                            rho=~error.corr),
```

```
                                        data = ARGOS[ARGOS$PTT == i
                                             & !is.na(ARGOS$ln.sd.x)
                                             & ARGOS$DEPLOY%in%"DEPLOYMENT"
                                             & ARGOS$FILTERED%in%"not",],
                                        coord = c("LONGITUDE","LATITUDE"),
                                        Time.name = "DATE_TIME",
                                        initial.state=INIT_ELLIPSE,
                                        fixPar = FIX_PAR_ELLIPSE,
                                        theta=THETA_ELLIPSE,
                                        prior=LAP_PRIOR,
                                        control=list(REPORT=10, trace=1),
                                        initialSANN=list(maxit=1000), attempts=5))

  }
 }
}


#### QUALITY: Fit CTCRW model on tags that lack Kalman filter ellipse information (hence
based on location quality classes)

## Set input values for model in Johnson et al. (2008) Ecology 89:1208-1215
## Start values for theta come from the estimates in Johnson et al. (2008)

# Define which model parameters will be fixed a priori
# and which will be estimated (NA -> free)
FIX_PAR_QUALITY = c(log(250), #location class 3 error parameters (fixed parameter)
          log(500), #location class 2 error parameters (fixed parameter)
          log(1500), #location class 1 error parameters (fixed parameter)
          rep(NA,3), #location class 0, A, B error parameters (free parameters)
          NA, #ln sigma: velocity variance parameter (free parameter)
          NA) #ln beta: velocity correlation parameter (free parameter)


# Establish starting estimates for free parameters
THETA_QUALITY = c(rep(log(2000),3), #starting estimates of location class 0, A, B error in m
         log(2*60*60), #ln sigma: starting estimate for velocity variance parameter based on a
2m/s velocity (transformed into m/hr)
         0) #ln beta: starting estimate for velocity correlation parameter

# Provide constraints on free parameters
CONSTR_QUALITY=list(

 lower=c(rep(log(1500),3), #lower constraint for location class 0, A, B error in m
```

```
        -Inf, #lower constraint for ln sigma (velocity variance parameter)
        -Inf), #lower constraint for ln beta (velocity correlation parameter)

  upper=c(rep(Inf,3), #upper constraint for location class 0, A, B error in m
        Inf, #upper constraint for ln sigma (velocity variance parameter)
        Inf) #upper constraint for ln beta (velocity correlation parameter)
)

# Define a Laplace prior (moderately peaked) for CTCRW model parameters
LAP_PRIOR = function(PAR){-abs(PAR[2]-4)/1}


ARGOS$QUALITY = factor(ARGOS$QUALITY, levels=c("3","2","1","0","A","B"))

for(i in
unique(ARGOS$PTT)[!unique(ARGOS$PTT)%in%substr(names(ARGOS_CTCRW_FIT),11,16)])
{# Define initial state (position and velocity) for model
  INIT_QUALITY = list(a=c(ARGOS[ARGOS$PTT == i
                            & is.na(ARGOS$ln.sd.x)
                            & ARGOS$DEPLOY%in%"DEPLOYMENT"
                            & ARGOS$FILTERED%in%"not","LONGITUDE"][1],#starting estimate of initial x
position
                       0,#starting estimate of initial x velocity
                       ARGOS[ARGOS$PTT == i
                            & is.na(ARGOS$ln.sd.x)
                            & ARGOS$DEPLOY%in%"DEPLOYMENT"
                            & ARGOS$FILTERED%in%"not","LATITUDE"][1],#starting estimate of initial y
position
                       0),#starting estimate of initial y velocity
                     P=diag(c(5000^2,
                            10^2,
                            5000^2,
                            10^2))) #variance-covariance matrix for the initial location and velocity

  if(nrow(ARGOS[ARGOS$PTT == i
            & is.na(ARGOS$ln.sd.x)
            & ARGOS$DEPLOY%in%"DEPLOYMENT"
            & ARGOS$FILTERED%in%"not",])==0){next}

  print(paste(i,ARGOS[ARGOS$PTT == i,"SPP"][1]))

  # Set-up loop that will fit crwMLE
  # and then will keep trying to refit crwMLE up to 10 times to see if it's possible to
  # obtain a fit that does not contain model fitting errors ("try-errors") and
```

```
 # where none of the parameter estimate standard errors are NaN

 for(j in 1:10){

  if(class(ARGOS_CTCRW_FIT[[paste("CTCRW_FIT_",i,sep="")]]) == "NULL" |
     class(ARGOS_CTCRW_FIT[[paste("CTCRW_FIT_",i,sep="")]])=="try-error"|
     any(is.nan(ARGOS_CTCRW_FIT[[paste("CTCRW_FIT_",i,sep="")]]$se) == T)){

   # Fit CTCRW model
   ARGOS_CTCRW_FIT[[paste("CTCRW_FIT_",i,sep="")]] <- try(crawl::crwMLE(mov.model=~1,
                                                    err.model=list(x=~QUALITY-1),
                                                    data = ARGOS[ARGOS$PTT == i
                                                             & is.na(ARGOS$ln.sd.x)
                                                             & ARGOS$DEPLOY%in%"DEPLOYMENT"
                                                             & ARGOS$FILTERED%in%"not",],
                                                    coord = c("LONGITUDE","LATITUDE"),
                                                    Time.name = "DATE_TIME",
                                                    initial.state=INIT_QUALITY,
                                                    fixPar = FIX_PAR_QUALITY,
                                                    theta=THETA_QUALITY,
                                                    constr = CONSTR_QUALITY,
                                                    prior=LAP_PRIOR,
                                                    control=list(REPORT=10, trace=1),
                                                    initialSANN=list(maxit=1000), attempts=5))

  }
 }
}



#### Error Handling Routine: identify ARGOS records that producted NaNs in estimating CTCRW
process model parameters
#### and try re-fitting models with the beta parameter fixed at 4 indicating no autocorrelation
of velocity vectors


#ELLIPSE: re-run CTCRW model with fixed beta parameter on tags that have Kalman filter ellipse
information
for(i in names(ARGOS_CTCRW_FIT)){
 # for(i in c("CTCRW_FIT_112732","CTCRW_FIT_112733")){

 if(any(!is.na(ARGOS[ARGOS$PTT == substr(i,11,16),"ln.sd.x"]))){
```

```
  #print the PTT, SPP, and parameter standard errors for fit problematic fits
  print(paste(i,ARGOS[ARGOS$PTT == substr(i,11,16),"SPP"][1]))

  # Set-up loop that will re-fit crwMLE using a fixed beta parameter (set to essentially Brownian
motion)
  # and then will keep trying to refit crwMLE up to 10 times
  # obtain a fit that does not contain model fitting errors ("try-errors") and
  # where none of the parameter estimate standard errors are NaN
  for(j in 1:10){

    # Refit CTCRW model if there are try-errors from previous fit
    if(class(ARGOS_CTCRW_FIT[[i]]) == "NULL" |
      class(ARGOS_CTCRW_FIT[[i]])=="try-error"){

      # Define which model parameters will be fixed a priori
      # and which will be estimated (NA -> free)
      #Fix the ln.beta parameter at 4 indicating no autocorrelation of velocity vectors
      FIX_PAR_ELLIPSE = c(1, #ln.sd.x: x location error parameter (known from Argos error ellipse
estimates so fixed at 1)
                  1, #ln.sd.y: y location error parameter (known from Argos error ellipse
estimates so fixed at 1)
                  NA, #ln sigma (intercept): velocity variance parameter (free parameter)
                  4) #ln beta (intercept): velocity correlation parameter (free parameter)

      # Establish starting estimates for free parameters
      THETA_ELLIPSE = c(log(1.5*60*60)) #ln sigma: starting estimate for velocity variance
parameter based on a 1.5 m/s velocity (transformed into m/hr)

      # Define initial state (position and velocity) for model
      INIT_ELLIPSE = list(a=c(ARGOS[ARGOS$PTT == substr(i,11,16)
                      & !is.na(ARGOS$ln.sd.x)
                      & ARGOS$DEPLOY%in%"DEPLOYMENT"
                      & ARGOS$FILTERED%in%"not","LONGITUDE"][1],#starting estimate of
initial x position
                    0,#starting estimate of initial x velocity
                    ARGOS[ARGOS$PTT == substr(i,11,16)
                      & !is.na(ARGOS$ln.sd.x)
                      & ARGOS$DEPLOY%in%"DEPLOYMENT"
                      & ARGOS$FILTERED%in%"not","LATITUDE"][1],#starting estimate of initial
y position
                    0),#starting estimate of initial y velocity
                  P=diag(c(5000^2,
                       10^2,
                       5000^2,
```

```
                        10^2)))#variance-covariance matrix for the initial location and velocity
        # Fit CTCRW model
        ARGOS_CTCRW_FIT[[i]] <- try(crawl::crwMLE(mov.model=~1,
                                    err.model=list(x=~ln.sd.x-1,
                                          y=~ln.sd.y-1,
                                          rho=~error.corr),
                                    data = ARGOS[ARGOS$PTT == substr(i,11,16)
                                          & !is.na(ARGOS$ln.sd.x)
                                          & ARGOS$DEPLOY%in%"DEPLOYMENT"
                                          & ARGOS$FILTERED%in%"not",],
                                    coord = c("LONGITUDE","LATITUDE"),
                                    Time.name = "DATE_TIME",
                                    initial.state=INIT_ELLIPSE,
                                    fixPar = FIX_PAR_ELLIPSE,
                                    theta=THETA_ELLIPSE,
                                    prior=LAP_PRIOR,
                                    control=list(REPORT=10, trace=1),
                                    initialSANN=list(maxit=1000), attempts=5))
       }

       # Refit CTCRW model if any parameter estimate standard errors are NaN
       if(class(ARGOS_CTCRW_FIT[[i]])!="try-error"){
        if(any(is.nan(ARGOS_CTCRW_FIT[[i]]$se) == T)){
         # Define which model parameters will be fixed a priori
         # and which will be estimated (NA -> free)
         #Fix the ln.beta parameter at 4 indicating no autocorrelation of velocity vectors
         FIX_PAR_ELLIPSE = c(1, #ln.sd.x: x location error parameter (known from Argos error
ellipse estimates so fixed at 1)
                    1, #ln.sd.y: y location error parameter (known from Argos error ellipse
estimates so fixed at 1)
                    NA, #ln sigma (intercept): velocity variance parameter (free parameter)
                    4) #ln beta (intercept): velocity correlation parameter (free parameter)

         # Establish starting estimates for free parameters
         THETA_ELLIPSE = c(log(1.5*60*60)) #ln sigma: starting estimate for velocity variance
parameter based on a 1.5 m/s velocity (transformed into m/hr)

         # Define initial state (position and velocity) for model
         INIT_ELLIPSE = list(a=c(ARGOS[ARGOS$PTT == substr(i,11,16)
                        & !is.na(ARGOS$ln.sd.x)
                        & ARGOS$DEPLOY%in%"DEPLOYMENT"
                        & ARGOS$FILTERED%in%"not","LONGITUDE"][1],#starting estimate of
initial x position
                     0,#starting estimate of initial x velocity
```

```
                    ARGOS[ARGOS$PTT == substr(i,11,16)
                       & !is.na(ARGOS$ln.sd.x)
                       & ARGOS$DEPLOY%in%"DEPLOYMENT"
                       & ARGOS$FILTERED%in%"not","LATITUDE"][1],#starting estimate of
initial y position
                    0),#starting estimate of initial y velocity
                  P=diag(c(5000^2,
                      10^2,
                      5000^2,
                      10^2)))#variance-covariance matrix for the initial location and velocity
    # Fit CTCRW model
    ARGOS_CTCRW_FIT[[i]] <- try(crawl::crwMLE(mov.model=~1,
                           err.model=list(x=~ln.sd.x-1,
                                  y=~ln.sd.y-1,
                                  rho=~error.corr),
                           data = ARGOS[ARGOS$PTT == substr(i,11,16)
                                 & !is.na(ARGOS$ln.sd.x)
                                 & ARGOS$DEPLOY%in%"DEPLOYMENT"
                                 & ARGOS$FILTERED%in%"not",],
                           coord = c("LONGITUDE","LATITUDE"),
                           Time.name = "DATE_TIME",
                           initial.state=INIT_ELLIPSE,
                           fixPar = FIX_PAR_ELLIPSE,
                           theta=THETA_ELLIPSE,
                           prior=LAP_PRIOR,
                           control=list(REPORT=10, trace=1),
                           initialSANN=list(maxit=1000), attempts=5))
    }
   }
  }
 }
}


#QUALITY: re-run CTCRW model with fixed beta parameter on tags that lack Kalman filter
ellipse information
for(i in names(ARGOS_CTCRW_FIT)){
 # for(i in c("CTCRW_FIT_112732","CTCRW_FIT_112733")){

 if(any(!is.finite(ARGOS[ARGOS$PTT == substr(i,11,16),"ln.sd.x"]))){

  #print the PTT, SPP, and parameter standard errors for fit problematic fits
  print(paste(i,ARGOS[ARGOS$PTT == substr(i,11,16),"SPP"][1]))
```

```
  # Set-up loop that will re-fit crwMLE using a fixed beta parameter (set to essentially Brownian
motion)
  # and then will keep trying to refit crwMLE up to 10 times
  # to obtain a fit that does not contain model fitting errors ("try-errors") and
  # where none of the parameter estimate standard errors are NaN
  for(j in 1:10){

    # Refit CTCRW model if there are try-errors from previous fit
    if(class(ARGOS_CTCRW_FIT[[i]]) == "NULL" |
      class(ARGOS_CTCRW_FIT[[i]])=="try-error"){

      # Define which model parameters will be fixed a priori
      # and which will be estimated (NA -> free)
      #Fix the ln.beta parameter at 4 indicating no autocorrelation of velocity vectors
      FIX_PAR_QUALITY = c(log(250), #location class 3 error parameters (fixed parameter)
                 log(500), #location class 2 error parameters (fixed parameter)
                 log(1500), #location class 1 error parameters (fixed parameter)
                 rep(NA,3), #location class 0, A, B error parameters (free parameters)
                 NA, #ln sigma: velocity variance parameter (free parameter)
                 4) #ln beta: velocity correlation parameter (free parameter)

      # Establish starting estimates for free parameters
      THETA_QUALITY = c(rep(log(2000),3), #starting estimates of location class 0, A, B error in m
                log(2*60*60)) #ln sigma: starting estimate for velocity variance parameter based
on a 2m/s velocity (transformed into m/hr)

      # Provide constraints on free parameters
      CONSTR_QUALITY=list(

        lower=c(rep(log(1500),3), #lower constraint for location class 0, A, B error in m
            -Inf), #lower constraint for ln sigma (velocity variance parameter)

        upper=c(rep(Inf,3), #upper constraint for location class 0, A, B error in m
            Inf) #upper constraint for ln sigma (velocity variance parameter)
      )

      # Define initial state (position and velocity) for model
      INIT_QUALITY = list(a=c(ARGOS[ARGOS$PTT == substr(i,11,16)
                    & !is.na(ARGOS$ln.sd.x)
                    & ARGOS$DEPLOY%in%"DEPLOYMENT"
                    & ARGOS$FILTERED%in%"not","LONGITUDE"][1],#starting estimate of
initial x position
                0,#starting estimate of initial x velocity
                ARGOS[ARGOS$PTT == substr(i,11,16)
```

```
                       & !is.na(ARGOS$ln.sd.x)
                       & ARGOS$DEPLOY%in%"DEPLOYMENT"
                       & ARGOS$FILTERED%in%"not","LATITUDE"][1],#starting estimate of initial
y position
                     0),#starting estimate of initial y velocity
                   P=diag(c(5000^2,
                        10^2,
                        5000^2,
                        10^2)))#variance-covariance matrix for the initial location and velocity

   # Fit CTCRW model
   ARGOS_CTCRW_FIT[[i]] <- try(crawl::crwMLE(mov.model=~1,
                              err.model=list(x=~QUALITY-1),
                              data = ARGOS[ARGOS$PTT == substr(i,11,16)
                                    & is.na(ARGOS$ln.sd.x)
                                    & ARGOS$DEPLOY%in%"DEPLOYMENT"
                                    & ARGOS$FILTERED%in%"not",],
                              coord = c("LONGITUDE","LATITUDE"),
                              Time.name = "DATE_TIME",
                              initial.state=INIT_QUALITY,
                              fixPar = FIX_PAR_QUALITY,
                              theta=THETA_QUALITY,
                              constr = CONSTR_QUALITY,
                              prior=LAP_PRIOR,
                              control=list(REPORT=10, trace=1),
                              initialSANN=list(maxit=1000), attempts=5))
  }


  # Refit CTCRW model if any parameter estimate standard errors are NaN
  if(class(ARGOS_CTCRW_FIT[[i]])!="try-error"){
   if(any(is.nan(ARGOS_CTCRW_FIT[[i]]$se) == T)){

    # Define which model parameters will be fixed a priori
    # and which will be estimated (NA -> free)
    #Fix the ln.beta parameter at 4 indicating no autocorrelation of velocity vectors
    FIX_PAR_QUALITY = c(log(250), #location class 3 error parameters (fixed parameter)
               log(500), #location class 2 error parameters (fixed parameter)
               log(1500), #location class 1 error parameters (fixed parameter)
               rep(NA,3), #location class 0, A, B error parameters (free parameters)
               NA, #ln sigma: velocity variance parameter (free parameter)
               4) #ln beta: velocity correlation parameter (free parameter)

    # Establish starting estimates for free parameters
```

```
	THETA_QUALITY = c(rep(log(2000),3), #starting estimates of location class 0, A, B error in
m
			log(2*60*60)) #ln sigma: starting estimate for velocity variance parameter
based on a 2m/s velocity (transformed into m/hr)

	# Provide constraints on free parameters
	CONSTR_QUALITY=list(

	  lower=c(rep(log(1500),3), #lower constraint for location class 0, A, B error in m
		-Inf), #lower constraint for ln sigma (velocity variance parameter)

	  upper=c(rep(Inf,3), #upper constraint for location class 0, A, B error in m
		Inf) #upper constraint for ln sigma (velocity variance parameter)
	)

	# Define initial state (position and velocity) for model
	INIT_QUALITY = list(a=c(ARGOS[ARGOS$PTT == substr(i,11,16)
				& !is.na(ARGOS$ln.sd.x)
				& ARGOS$DEPLOY%in%"DEPLOYMENT"
				& ARGOS$FILTERED%in%"not","LONGITUDE"][1],#starting estimate of
initial x position
			0,#starting estimate of initial x velocity
			ARGOS[ARGOS$PTT == substr(i,11,16)
				& !is.na(ARGOS$ln.sd.x)
				& ARGOS$DEPLOY%in%"DEPLOYMENT"
				& ARGOS$FILTERED%in%"not","LATITUDE"][1],#starting estimate of
initial y position
			0),#starting estimate of initial y velocity
		P=diag(c(5000^2,
			10^2,
			5000^2,
			10^2)))#variance-covariance matrix for the initial location and velocity

	# Fit CTCRW model
	ARGOS_CTCRW_FIT[[i]] <- try(crawl::crwMLE(mov.model=~1,
				err.model=list(x=~QUALITY-1),
				data = ARGOS[ARGOS$PTT == substr(i,11,16)
					& is.na(ARGOS$ln.sd.x)
					& ARGOS$DEPLOY%in%"DEPLOYMENT"
					& ARGOS$FILTERED%in%"not",],
				coord = c("LONGITUDE","LATITUDE"),
				Time.name = "DATE_TIME",
				initial.state=INIT_QUALITY,
				fixPar = FIX_PAR_QUALITY,
```

```
                                theta=THETA_QUALITY,
                                constr = CONSTR_QUALITY,
                                prior=LAP_PRIOR,
                                control=list(REPORT=10, trace=1),
                                initialSANN=list(maxit=1000), attempts=5))
    }
   }
  }
 }
}

#### Error Removal Routine: identify ARGOS records that still produce errors after fixing the
#### beta parameter at 4 and trying to re-fit the models. Record these errors in a table and
remove from ARGOS_CTCRW_FIT

ARGOS_CTCRW_FIT_ERRORS = data.frame()
for(i in names(ARGOS_CTCRW_FIT))
{ if(class(ARGOS_CTCRW_FIT[[i]])=="try-error")
{ARGOS_CTCRW_FIT_ERRORS = rbind(ARGOS_CTCRW_FIT_ERRORS,data.frame(PTT =
as.numeric(substr(i,11,16)),
                                      ERROR_TYPE = ARGOS_CTCRW_FIT[[i]][1]))
ARGOS_CTCRW_FIT[[i]] = NULL}

 if(any(is.nan(ARGOS_CTCRW_FIT[[i]]$se) == T))
 {ARGOS_CTCRW_FIT_ERRORS = rbind(ARGOS_CTCRW_FIT_ERRORS,data.frame(PTT =
as.numeric(substr(i,11,16)),
                                       ERROR_TYPE = "S.E. NaN"))
 ARGOS_CTCRW_FIT[[i]] = NULL}

 if(any(is.na(ARGOS_CTCRW_FIT[[i]]$se[(length(ARGOS_CTCRW_FIT[[i]]$se)-
1):length(ARGOS_CTCRW_FIT[[i]]$se)]) == T))
 {ARGOS_CTCRW_FIT_ERRORS = rbind(ARGOS_CTCRW_FIT_ERRORS,data.frame(PTT =
as.numeric(substr(i,11,16)),
                                       ERROR_TYPE = "S.E. NA"))
 ARGOS_CTCRW_FIT[[i]] = NULL}
}

#Add Species information to ARGOS_CTCRW_FIT_ERRORS
ARGOS_CTCRW_FIT_ERRORS <-
merge(ARGOS_CTCRW_FIT_ERRORS,TAGS[,c("PTT","Species","SPP")],by = "PTT",all.x = T)

# Clean-up
remove(i)
```

```
##### PRED_TIME #####
## we'll setup our predTimes based on the date_hour
## column in the fit$data
PRED_TIME = list()
for(i in names(ARGOS_CTCRW_FIT))
{# set up a sequence of POSIXct times, starting by rounding up to the nearest hour
  # from the first time-stamp used in CTCRW model fit and ending with the nearest hour
  # rounding down from the last time-stamp used in CTCRW model fit
  PRED_TIME[[i]] <- seq(from =
lubridate::ceiling_date(min(ARGOS_CTCRW_FIT[[i]]$data$DATE_TIME, na.rm = TRUE), unit =
"hours"),
                      to = lubridate::floor_date(max(ARGOS_CTCRW_FIT[[i]]$data$DATE_TIME, na.rm =
TRUE), unit = "hours"),
                      by = '30 min')
  #                    by = '10 min')
}

# Clean-up
remove(i)

##### ARGOS_CTCRW_PRED #####

ARGOS_CTCRW_PRED = list()
ARGOS_CTCRW_PRED_ERROR = data.frame()

# returns numeric to a POSIXct vector
for(i in names(ARGOS_CTCRW_FIT)){
  ARGOS_CTCRW_PRED[[i]] <- try(crawl::crwPredict(ARGOS_CTCRW_FIT[[i]], predTime =
PRED_TIME[[i]]))

  if(class(ARGOS_CTCRW_PRED[[i]]) == "try-error"){
    ARGOS_CTCRW_PRED_ERROR = rbind(ARGOS_CTCRW_PRED_ERROR,data.frame(PTT =
substr(i,11,16),ERROR_TYPE = "crwPredict"))
    ARGOS_CTCRW_PRED[[i]] <- NULL
    next
  }

  # returns internal TimeNum in numeric format to a POSIXct vector

ARGOS_CTCRW_PRED[[i]]$DATE_TIME=crawl::intToPOSIX(ARGOS_CTCRW_PRED[[i]]$TimeNum,
tz = "GMT")

  # save projected coordinates
  ARGOS_CTCRW_PRED[[i]]$LATITUDE = ARGOS_CTCRW_PRED[[i]]$mu.y
```

```
 ARGOS_CTCRW_PRED[[i]]$LONGITUDE = ARGOS_CTCRW_PRED[[i]]$mu.x

 # turn ARGOS_CTCRW_PRED[[i]] briefly into a SpatialPointsDataFrame to change
 # projection of CTCRW MLE predictions
 coordinates(ARGOS_CTCRW_PRED[[i]]) <- ~ mu.x + mu.y

 # project ARGOS_CTCRW_PRED[[i]] SpatialPointsDataFrame to appropriate local projection
 if(ARGOS_CTCRW_PRED[[i]]$REGION[1] %in% c("WAP","Weddell")){
  proj4string(ARGOS_CTCRW_PRED[[i]]) <- CRS(PROJ_WAP)
 }

 if(ARGOS_CTCRW_PRED[[i]]$REGION[1] == "Ross"){
  proj4string(ARGOS_CTCRW_PRED[[i]]) <- CRS(PROJ_ROSS)
 }

 ## Transform coordinates back to GCS_WGS84
 ARGOS_CTCRW_PRED[[i]] <- try(spTransform(ARGOS_CTCRW_PRED[[i]],
CRS("+init=epsg:4326")))

 if(class(ARGOS_CTCRW_PRED[[i]]) == "try-error"){
  ARGOS_CTCRW_PRED_ERROR = rbind(ARGOS_CTCRW_PRED_ERROR,data.frame(PTT =
substr(i,11,16),ERROR_TYPE = "spTransform"))
  ARGOS_CTCRW_PRED[[i]] <- NULL
  next
 }

 # turn ARGOS_CTCRW_PRED[[i]] back into a data frame
 ARGOS_CTCRW_PRED[[i]] <- as.data.frame(ARGOS_CTCRW_PRED[[i]],stringsAsFactors = F)

 # assign LATITUDE_GCS and LONGITUDE_GCS
 ARGOS_CTCRW_PRED[[i]]$LATITUDE_GCS = ARGOS_CTCRW_PRED[[i]]$mu.y
 ARGOS_CTCRW_PRED[[i]]$LONGITUDE_GCS = ARGOS_CTCRW_PRED[[i]]$mu.x

}

# Clean-up
remove(i)

#Add Species information to ARGOS_CTCRW_PRED_ERROR
ARGOS_CTCRW_PRED_ERROR <-
merge(ARGOS_CTCRW_PRED_ERROR,TAGS[,c("PTT","Species","SPP")],by = "PTT",all.x = T)


##### ARGOS_CTCRW #####
```

```
# Concatenate data.frames in ARGOS_CTCRW_PRED list into a single data.frame
ARGOS_CTCRW = do.call("rbind",ARGOS_CTCRW_PRED)

# Assign MIGRATION status based on manually scored
# MIGRATION timing landmarks (departure, turning, return, end)
ARGOS_CTCRW$MIGRATION = "PRE_MIG"

for(i in which(!is.na(MIGRATION$OUT_MIG))){
  ARGOS_CTCRW[ARGOS_CTCRW$PTT == MIGRATION$PTT[i]
          & ARGOS_CTCRW$DATE_TIME >= MIGRATION$OUT_MIG[i],"MIGRATION"] =
"OUT_MIG"

  if(!is.na(MIGRATION$TURN[i]))
  {ARGOS_CTCRW[ARGOS_CTCRW$PTT == MIGRATION$PTT[i]
          & ARGOS_CTCRW$DATE_TIME >= MIGRATION$TURN[i],"MIGRATION"] = "TURN_N"}

  if(!is.na(MIGRATION$FURTHEST_N[i]))
  {ARGOS_CTCRW[ARGOS_CTCRW$PTT == MIGRATION$PTT[i]
          & ARGOS_CTCRW$DATE_TIME >= MIGRATION$FURTHEST_N[i],"MIGRATION"] =
"TURN_S"}

  if(!is.na(MIGRATION$IN_MIG[i]))
  {ARGOS_CTCRW[ARGOS_CTCRW$PTT == MIGRATION$PTT[i]
          & ARGOS_CTCRW$DATE_TIME >= MIGRATION$IN_MIG[i],"MIGRATION"] = "IN_MIG"}

  if(!is.na(MIGRATION$END_MIG[i]))
  {ARGOS_CTCRW[ARGOS_CTCRW$PTT == MIGRATION$PTT[i]
          & ARGOS_CTCRW$DATE_TIME >= MIGRATION$END_MIG[i],"MIGRATION"] =
"POST_MIG"}
}

# Clean-up
remove(i)

# Create a date corresponding to the format of raster names (timestamps)
ARGOS_CTCRW$YEAR=as.numeric(as.character(lubridate::year(ARGOS_CTCRW$DATE_TIME)))
ARGOS_CTCRW$MONTH=sprintf("%02d",as.numeric(lubridate::month(ARGOS_CTCRW$DATE_T
IME)))
ARGOS_CTCRW$DAY=sprintf("%02d",as.numeric(as.character(lubridate::day(ARGOS_CTCRW$D
ATE_TIME))))
ARGOS_CTCRW$RAST_DATE=paste("X",ARGOS_CTCRW$YEAR,"_",ARGOS_CTCRW$MONTH,"_",
ARGOS_CTCRW$DAY,sep="")
```

```
#Calculate astronomical values based on location and time (sunrise, sunset, solar elevation, solar azimuth)
Sys.setenv(TZ="GMT")
ARGOS_CTCRW$SUNRISE=maptools::sunriset(as.matrix(cbind(ARGOS_CTCRW$LONGITUDE_GCS,ARGOS_CTCRW$LATITUDE_GCS)),
                                      as.POSIXct(ARGOS_CTCRW[,"DATE_TIME"],tz="GMT"),
                                      direction="sunrise", POSIXct.out=TRUE)$time
ARGOS_CTCRW$SUNSET=maptools::sunriset(as.matrix(cbind(ARGOS_CTCRW$LONGITUDE_GCS,ARGOS_CTCRW$LATITUDE_GCS)),
                                     as.POSIXct(ARGOS_CTCRW[,"DATE_TIME"],tz="GMT"),
                                     direction="sunset", POSIXct.out=TRUE)$time
ARGOS_CTCRW$SUN_ELEV=maptools::solarpos(as.matrix(cbind(ARGOS_CTCRW$LONGITUDE_GCS,ARGOS_CTCRW$LATITUDE_GCS)),
                                       as.POSIXct(ARGOS_CTCRW[,"DATE_TIME"],tz="GMT"),
                                       direction="sunset", POSIXct.out=TRUE)[,2]
ARGOS_CTCRW$SUN_AZI=maptools::solarpos(as.matrix(cbind(ARGOS_CTCRW$LONGITUDE_GCS,ARGOS_CTCRW$LATITUDE_GCS)),
                                      as.POSIXct(ARGOS_CTCRW[,"DATE_TIME"],tz="GMT"),
                                      direction="sunset", POSIXct.out=TRUE)[,1]


# Add time since last locType=="o" column to remove duty cycle periods from predictions
ARGOS_CTCRW[ARGOS_CTCRW$locType=="o","OBS_TIME"]=ARGOS_CTCRW[ARGOS_CTCRW$locType=="o","DATE_TIME"]
ARGOS_CTCRW[ARGOS_CTCRW$locType=="o","OBS_TIME_1"]=ARGOS_CTCRW[ARGOS_CTCRW$locType=="o","DATE_TIME"][c(2:nrow(ARGOS_CTCRW[ARGOS_CTCRW$locType=="o",]),nrow(ARGOS_CTCRW[ARGOS_CTCRW$locType=="o",]))]
ARGOS_CTCRW$OBS_TIME=zoo::na.locf(ARGOS_CTCRW$OBS_TIME)
ARGOS_CTCRW$OBS_TIME_1=zoo::na.locf(ARGOS_CTCRW$OBS_TIME_1)
ARGOS_CTCRW$OBS_TIME = crawl::intToPOSIX(ARGOS_CTCRW$OBS_TIME)
ARGOS_CTCRW$OBS_TIME_1 = crawl::intToPOSIX(ARGOS_CTCRW$OBS_TIME_1)
ARGOS_CTCRW$OBS_TIME_DIFF=as.numeric(difftime(ARGOS_CTCRW$OBS_TIME_1,ARGOS_CTCRW$OBS_TIME,units = "hours"))


# ARGOS_CTCRW[ARGOS_CTCRW$SPP=="KWC","BATHY"] <- raster::extract(BATHY_ROSS,
#                                                                ARGOS_CTCRW[ARGOS_CTCRW$SPP=="KWC",c("LONGITUDE_GCS_360",
#                                                                          "LATITUDE_GCS_360")])
# ARGOS_CTCRW[ARGOS_CTCRW$SPP!="KWC","BATHY"] <- raster::extract(BATHY,
#                                                                ARGOS_CTCRW[ARGOS_CTCRW$SPP!="KWC",c("LONGITUDE_GCS",
#                                                                          "LATITUDE_GCS")])
```

```
# Save a copy of full data.frame before removing locType = "o" rows
ARGOS_CTCRW_FULL <- ARGOS_CTCRW

#Remove locType == "o" rows to leave just the regular predicted locations
ARGOS_CTCRW = ARGOS_CTCRW[ARGOS_CTCRW$locType == "p",]

## Save as set of coordinates in 0:360 longitude for plotting accross the dateline
ARGOS_CTCRW$LONGITUDE_GCS_360 <- ifelse(ARGOS_CTCRW$LONGITUDE_GCS<0,
                                ARGOS_CTCRW$LONGITUDE_GCS+360,
                                ARGOS_CTCRW$LONGITUDE_GCS)
ARGOS_CTCRW$LATITUDE_GCS_360 <- ARGOS_CTCRW$LATITUDE_GCS

# Calculate the distance between sequential CTCRW predicted locations
# within each PTT in kilometers
for(i in unique(ARGOS_CTCRW$PTT))
{ ARGOS_CTCRW [ARGOS_CTCRW$PTT == i,"DIST_GEO"] = 0
ARGOS_CTCRW [ARGOS_CTCRW$PTT == i,"DIST_GEO"][2:nrow(ARGOS_CTCRW
[ARGOS_CTCRW$PTT == i,])] <-
  geosphere::distGeo(p1 = ARGOS_CTCRW [ARGOS_CTCRW$PTT ==
i,c("LONGITUDE_GCS","LATITUDE_GCS")][1:(nrow(ARGOS_CTCRW [ARGOS_CTCRW$PTT == i,])-
1),],
            p2 = ARGOS_CTCRW [ARGOS_CTCRW$PTT ==
i,c("LONGITUDE_GCS","LATITUDE_GCS")][2:nrow(ARGOS_CTCRW [ARGOS_CTCRW$PTT ==
i,]),])/1000
}

# Clean-up
remove(i)


# Calculate the predicted velocity (actually speed) in km/h
# based on a 0.5 hour interval between sequential predictions
ARGOS_CTCRW$VEL_GEO = ARGOS_CTCRW$DIST_GEO/0.5

# ARGOS_CTCRW$DIST_GEO_2H = NA
# for(i in unique(ARGOS_CTCRW$PTT))
# {
#   ARGOS_CTCRW[ARGOS_CTCRW$PTT == i,"INDEX_2H"] = rep(1:100000,each = 4,length.out =
nrow(ARGOS_CTCRW [ARGOS_CTCRW$PTT == i,]))
#
#   for(j in unique(ARGOS_CTCRW[ARGOS_CTCRW$PTT == i,"INDEX_2H"]))
#   {
```

```
#    ARGOS_CTCRW[ARGOS_CTCRW$PTT == i & ARGOS_CTCRW$INDEX_2H ==
j,"DIST_GEO_2H"] = sum(ARGOS_CTCRW[ARGOS_CTCRW$PTT == i &
ARGOS_CTCRW$INDEX_2H == j ,"DIST_GEO"])
#   }
# }
#
# ARGOS_CTCRW$VEL_GEO_2H = ARGOS_CTCRW$DIST_GEO_2H/2
#
# # Clean-up
# remove(i,j)



# Calculate the animal's CTCRW predicted DIRECTION of travel in 0:360 degrees
ARGOS_CTCRW$DIRECTION = -1 * (-90+atan2(ARGOS_CTCRW$nu.y,ARGOS_CTCRW$nu.x) *
180/pi)
ARGOS_CTCRW$DIRECTION = ifelse(ARGOS_CTCRW$DIRECTION<0,
                        360+ARGOS_CTCRW$DIRECTION,
                        ARGOS_CTCRW$DIRECTION)

#Calculate the change in direction between sequential CTCRW segments
for(i in unique(ARGOS_CTCRW$PTT))
{
  ARGOS_CTCRW[ARGOS_CTCRW$PTT == i,"DIRECTION_DIFF"] =
c(0,ARGOS_CTCRW[ARGOS_CTCRW$PTT ==
i,"DIRECTION"][1:(nrow(ARGOS_CTCRW[ARGOS_CTCRW$PTT == i,])-1)]-
                                          ARGOS_CTCRW[ARGOS_CTCRW$PTT ==
i,"DIRECTION"][2:nrow(ARGOS_CTCRW[ARGOS_CTCRW$PTT == i,])])
  # for(j in unique(ARGOS_CTCRW[ARGOS_CTCRW$PTT == i,"INDEX_2H"]))
  # {
  #   ARGOS_CTCRW[ARGOS_CTCRW$PTT == i & ARGOS_CTCRW$INDEX_2H ==
j,"DIRECTION_VAR_2H"] = var(ARGOS_CTCRW[ARGOS_CTCRW$PTT == i &
ARGOS_CTCRW$INDEX_2H == j ,"DIRECTION_DIFF"])
  # }
}

# Clean-up
remove(i)


# ARGOS_CTCRW$DIRECTION_VAR_2H = NA
# for(i in unique(ARGOS_CTCRW$PTT))
# {
#   for(j in unique(ARGOS_CTCRW[ARGOS_CTCRW$PTT == i,"INDEX_2H"]))
```

```
#   {
#     ARGOS_CTCRW[ARGOS_CTCRW$PTT == i & ARGOS_CTCRW$INDEX_2H ==
j,"DIRECTION_VAR_2H"] = var(ARGOS_CTCRW[ARGOS_CTCRW$PTT == i &
ARGOS_CTCRW$INDEX_2H == j ,"DIRECTION_DIFF"])
#   }
# }
#
# # Clean-up
# remove(i,j)

##### ARGOS_CTCRW_SIM #####

ARGOS_CTCRW_SIM = list()
ARGOS_CTCRW_SIM_ERROR = data.frame()

for(i in names(ARGOS_CTCRW_FIT)){

  ARGOS_CTCRW_SIM[[i]] <- try(crawl::crwSimulator(ARGOS_CTCRW_FIT[[i]], predTime =
PRED_TIME[[i]]))

  if(class(ARGOS_CTCRW_SIM[[i]]) == "try-error"){

    ARGOS_CTCRW_SIM_ERROR = rbind(ARGOS_CTCRW_SIM_ERROR,data.frame(PTT =
substr(i,11,16),
                                                   ERROR_TYPE = "crwSimulator"))
    ARGOS_CTCRW_SIM[[i]] <- NULL
    next
  }

}
# Clean-up
remove(i)

##### ARGOS_CTCRW_SIM_TRACKS #####

## Use crwPostIS to generate a random draw from the posterior and save in a list
ARGOS_CTCRW_SIM_TRACKS <- list()

for(i in names(ARGOS_CTCRW_SIM))
{# create a list within a list to house each track stored as a SpatialPointsDataFrame
  ARGOS_CTCRW_SIM_TRACKS[[i]] <- list()

  for(j in 1:20)
  {## Use crwPostIS to generate a random draw from the posterior of ARGOS_CTCRW_FIT[[i]]
```

```
    SIM_TRACKS_temp <- crawl::crwPostIS(ARGOS_CTCRW_SIM[[i]])

    #convert crwIS simulation objects to dataframes
    SIM_TRACKS_temp <- data.frame(Time = SIM_TRACKS_temp$Time,
                          SIM_TRACKS_temp$alpha.sim,
                          locType = SIM_TRACKS_temp$locType)
    SIM_TRACKS_temp$Time  = crawl::intToPOSIX(SIM_TRACKS_temp$Time)
    SIM_TRACKS_temp$Time_Elapsed =
as.numeric(difftime(SIM_TRACKS_temp$Time,SIM_TRACKS_temp$Time[1],units = "days"))

    #label and convert dataframes to SpatialPointsDataFrames
    coordinates(SIM_TRACKS_temp) <- c("mu.x","mu.y")

    # project SpatialPointsDataFrame to appropriate local projection
    if(ARGOS[ARGOS$PTT%in%substr(i,11,16), "REGION"][1] %in% c("WAP","Weddell"))
    {proj4string(SIM_TRACKS_temp) <- CRS(PROJ_WAP)}

    if(ARGOS[ARGOS$PTT%in%substr(i,11,16), "REGION"][1] == "Ross")
    {proj4string(SIM_TRACKS_temp) <- CRS(PROJ_ROSS)}

    # store track
    ARGOS_CTCRW_SIM_TRACKS[[i]][[paste("TRACK_",j,sep="")]] <- SIM_TRACKS_temp}

}

remove(SIM_TRACKS_temp)

# Clean-up
remove(i,j)


# Simulate up to 100 tracks from CTCRW tracks of "Bb","Mn","KWA","KWB","KWG" in Peninsula
region (for UD calculation)
for(i in
paste("CTCRW_FIT_",unique(ARGOS_CTCRW[ARGOS_CTCRW$SPP%in%c("Bb","Mn","KWA","K
WB","KWG")
                              & !ARGOS_CTCRW$PTT%in%ARGOS_CTCRW_SIM_UD_ERROR$PTT
                              &
ARGOS_CTCRW$PTT%in%substr(names(ARGOS_CTCRW_SIM),11,16)
                              &
ARGOS_CTCRW$REGION%in%c("WAP","Weddell"),"PTT"]),sep="")){
  for(j in 21:100){

    ## Use crwPostIS to generate a random draw from the posterior of ARGOS_CTCRW_FIT[[i]]
```

```
    SIM_TRACKS_temp <- crawl::crwPostIS(ARGOS_CTCRW_SIM[[i]])

    #convert crwIS simulation objects to dataframes
    SIM_TRACKS_temp <- data.frame(Time = SIM_TRACKS_temp$Time,
                                  SIM_TRACKS_temp$alpha.sim,
                                  locType = SIM_TRACKS_temp$locType)
    SIM_TRACKS_temp$Time  = crawl::intToPOSIX(SIM_TRACKS_temp$Time)
    SIM_TRACKS_temp$Time_Elapsed =
as.numeric(difftime(SIM_TRACKS_temp$Time,SIM_TRACKS_temp$Time[1],units = "days"))

    #label and convert dataframes to SpatialPointsDataFrames
    coordinates(SIM_TRACKS_temp) <- c("mu.x","mu.y")

    # project SpatialPointsDataFrame to appropriate local projection
    if(ARGOS[ARGOS$PTT%in%substr(i,11,16), "REGION"][1] %in% c("WAP","Weddell"))
    {proj4string(SIM_TRACKS_temp) <- CRS(PROJ_WAP)}

    if(ARGOS[ARGOS$PTT%in%substr(i,11,16), "REGION"][1] == "Ross")
    {proj4string(SIM_TRACKS_temp) <- CRS(PROJ_ROSS)}

    # store track
    ARGOS_CTCRW_SIM_TRACKS[[i]][[paste("TRACK_",j,sep="")]] <- SIM_TRACKS_temp}

}

remove(SIM_TRACKS_temp)

# Clean-up
remove(i,j)


# Save model fit as an .rds file in case of crashes
saveRDS(object = ARGOS_CTCRW_SIM_TRACKS,
        file = paste("/Users/trevor.joyce/Grad School/Research/",
                     "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                     "ARGOS_CTCRW_SIM_TRACKS.rds",sep=""))

# Reconstitute STPP_SIM_TRACKS from saved copy
ARGOS_CTCRW_SIM_TRACKS <- readRDS(file = paste("/Users/trevor.joyce/Grad
School/Research/",
                                   "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                                   "ARGOS_CTCRW_SIM_TRACKS.rds",sep=""))
```

```
for(i in unique(ARGOS_CTCRW[ARGOS_CTCRW$SPP%in%c("Bb","Mn","KWA","KWB","KWG")
                    & !ARGOS_CTCRW$PTT%in%ARGOS_CTCRW_SIM_UD_ERROR$PTT
                    & ARGOS_CTCRW$PTT%in%substr(names(ARGOS_CTCRW_SIM),11,16)
                    & ARGOS_CTCRW$REGION%in%c("WAP","Weddell"),"PTT"])){
  for(j in c(1:100)){

    #Pull each simulated track from ARGOS_CTCRW_SIM_TRACKS
    SIM_TRACKS_temp <- ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep =
"")]][[paste("TRACK_",j,sep="")]]

    #Convert simulated tracks into GCS_WGS84
    SIM_TRACKS_temp <- sp::spTransform(SIM_TRACKS_temp,
                                        CRS("+init=epsg:4326"))

    # Save unprojected (GCS_WGS84) coordinates
    SIM_TRACKS_temp$LONG <- SIM_TRACKS_temp$mu.x
    SIM_TRACKS_temp$LAT <- SIM_TRACKS_temp$mu.y

    # convert to data.frame to allow diff and na.locf functions (don't work with
SpatialPointsDataFrame)
    SIM_TRACKS_temp <- as.data.frame(SIM_TRACKS_temp)

    # Define TRACK for later multiple imputation implementation
    SIM_TRACKS_temp$TRACK <- paste("TRACK_",j,sep="")

    # Create variables to house LAND corrected values
    SIM_TRACKS_temp[,c("LAND","LONG_FIXED","LAT_FIXED")] <- NA

    # Select a subset of fields for merging back to ARGOS_CTCRW_SIM_TRACKS
    SIM_TRACKS_temp <-
SIM_TRACKS_temp[,c("TRACK","LONG","LAT","LAND","LONG_FIXED","LAT_FIXED")]

    # Convert ARGOS_CTCRW_SIM_TRACKS from SpatialPointsDataFrame to data.frame
    ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep = "")]][[paste("TRACK_",j,sep="")]]
<- as.data.frame(ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep =
"")]][[paste("TRACK_",j,sep="")]])

    # Append SIM_TRACKS_temp information the data.frames in ARGOS_CTCRW_SIM_TRACKS
    ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep = "")]][[paste("TRACK_",j,sep="")]]
<- cbind(ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep =
"")]][[paste("TRACK_",j,sep="")]],
                                                              SIM_TRACKS_temp)
```

```
 }

}

remove(SIM_TRACKS_temp,i,j)

# Save kernel intensity values to a single overall .rds file
# (avoids having to re-run calculations)
saveRDS(object = ARGOS_CTCRW_SIM_TRACKS,
      file = paste("/Users/trevor.joyce/Grad School/Research/",
                "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                "ARGOS_CTCRW_SIM_TRACKS.rds",sep=""))

# Save kernel intensity values to a single overall .rds file
# (avoids having to re-run calculations)
saveRDS(object =
ARGOS_CTCRW_SIM_TRACKS[names(ARGOS_CTCRW_SIM_TRACKS)%in%paste("CTCRW_FIT_",
unique(ARGOS_CTCRW[ARGOS_CTCRW$SPP%in%c("Bb","Mn","KWA","KWB","KWG")
                                                            &
!ARGOS_CTCRW$PTT%in%ARGOS_CTCRW_SIM_UD_ERROR$PTT
                                                            &
ARGOS_CTCRW$PTT%in%substr(names(ARGOS_CTCRW_SIM),11,16)
                                                            &
ARGOS_CTCRW$REGION%in%c("WAP","Weddell"),"PTT"]),sep = "")],
      file = paste("/Users/trevor.joyce/Grad School/Research/",
                "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                "ARGOS_CTCRW_SIM_TRACKS_SUB.rds",sep=""))

#
# # Loop through each CTCRW_FIT in ARGOS_CTCRW_SIM_TRACKS to run FIX_SIM_TRACKS
function
# # which LAND corrects simulated TRACKS
# for(i in unique(ARGOS_CTCRW[ARGOS_CTCRW$SPP%in%c("Bb","Mn","KWA","KWB")
#                 & !ARGOS_CTCRW$PTT%in%ARGOS_CTCRW_SIM_UD_ERROR$PTT
#                 & ARGOS_CTCRW$PTT%in%substr(names(ARGOS_CTCRW_SIM),11,16)
#                 & ARGOS_CTCRW$REGION%in%c("WAP","Weddell"),"PTT"])){
#
#   print(i)
#   PROCESSING_TIME_temp <- proc.time()
#
#   # run FIX_SIM_TRACKS function which on each simulated TRACK within i CTCRW_FIT point
#   # that falls within the bbox of COVAR
#   SIM_TRACKS_FIXED_temp <- try(parallel::mclapply(X = c(1:100),
```

```
#                               DATA = ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_", i,
sep = "")]],
#                               LAND_RAST = LAND_RAST_temp,
#                               TRANS_MATRIX = TRANS_MATRIX_temp,
#                               FUN = FIX_SIM_TRACKS,
#                               mc.cores = 2))
#  # Error-handling
#  if(class(SIM_TRACKS_FIXED_temp) == "try-error"){
#   print(SIM_TRACKS_FIXED_temp[1])
#   next
#  }
#
#  # Print information on PROCESSING_TIME associated with specific PTT and TRACK
#  print(proc.time() - PROCESSING_TIME_temp)
#
#  for (j in c(1:100)){
#   ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_", i, sep =
"")]][[paste("TRACK_",j,sep="")]][,c("LAND","LONG_FIXED","LAT_FIXED")] <-
SIM_TRACKS_FIXED_temp[[j]][,c("LAND","LONG_FIXED","LAT_FIXED")]
#  }
#
#  remove(SIM_TRACKS_FIXED_temp,PROCESSING_TIME_temp); gc()
#
#  # Save kernel intensity values to a single overall .rds file
#  # (avoids having to re-run calculations)
#  saveRDS(object = ARGOS_CTCRW_SIM_TRACKS,
#       file = paste("/Users/trevor.joyce/Grad School/Research/",
#              "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
#              "ARGOS_CTCRW_SIM_TRACKS_FIXED.rds",sep=""))
# }
#
#
# for(i in unique(ARGOS_CTCRW[ARGOS_CTCRW$SPP%in%c("Bb","Mn","KWA","KWB")
#                & !ARGOS_CTCRW$PTT%in%ARGOS_CTCRW_SIM_UD_ERROR$PTT
#                & ARGOS_CTCRW$PTT%in%substr(names(ARGOS_CTCRW_SIM),11,16)
#                & ARGOS_CTCRW$REGION%in%c("WAP","Weddell"),"PTT"])){
#  for(j in c(1:100)){
#
#   #Pull each simulated track from ARGOS_CTCRW_SIM_TRACKS
#   SIM_TRACKS_temp <- ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep =
"")]][[paste("TRACK_",j,sep="")]]
#
#   # calculate the time difference between observed Argos fixes
```

```
#    SIM_TRACKS_temp[SIM_TRACKS_temp$locType=="o","OBS_TIME_DIFF"] <-
c(as.numeric(diff(SIM_TRACKS_temp[SIM_TRACKS_temp$locType=="o","Time"],
#                                                   units = "secs")),0)
#    # copy down OBS_TIME_DIFF to locType=="p" rows
#    SIM_TRACKS_temp$OBS_TIME_DIFF <- zoo::na.locf(SIM_TRACKS_temp$OBS_TIME_DIFF)
#
#    # convert OBS_TIME_DIFF to units of hours
#    SIM_TRACKS_temp$OBS_TIME_DIFF <- SIM_TRACKS_temp$OBS_TIME_DIFF/(60*60)
#
#    # Define TRACK for later multiple imputation implementation
#    SIM_TRACKS_temp$TRACK <- paste("TRACK_",j,sep="")
#
#    # Define CRW_FIT for later multiple imputation implementation
#    SIM_TRACKS_temp$CTCRW_FIT <- paste("CTCRW_FIT_",i,sep = "")
#
#    # Define PTT
#    SIM_TRACKS_temp$PTT <- i
#
#    # Define SPP
#    SIM_TRACKS_temp$SPP <- TAGS[TAGS$PTT == i,"SPP"][1]
#
#    # Define a WEDDELL_SEA polygon
#    WEDDELL_SEA <- data.frame(LONG = c(-57.3,-56.4,-45.5,-14.0,-06.0,-28.0,-58.0,-65.0,-64.0,-
57.3),
#                    LAT =  c(-63.25,-63.0,-61.1,-63.5,-71.0,-78.0,-77.0,-73.0,-66.0,-63.25))
#
#    # Determine which points fall within WEDDELL_SEA polygon
#    SIM_TRACKS_temp$REGION <- sp::point.in.polygon(point.x = SIM_TRACKS_temp$LONG,
#                              point.y = SIM_TRACKS_temp$LAT,
#                              pol.x = WEDDELL_SEA$LONG,
#                              pol.y = WEDDELL_SEA$LAT)
#
#    # Convert binary 0 or 1 to REGION lable
#    STPP_SIM_TRACKS$REGION <- ifelse(STPP_SIM_TRACKS$REGION >= 1, "WEDD","WAP")
#
#
#    # Label predictions as falling before (PRE_MIG) or after (MIG) the OUT_MIG date in
MIGRATION table
#    SIM_TRACKS_temp$MIGRATION <- "PRE_MIG"
#
#    if(!is.na(MIGRATION[MIGRATION$PTT == i, "OUT_MIG"])){
#      SIM_TRACKS_temp[SIM_TRACKS_temp$Time >= MIGRATION[MIGRATION$PTT == i,
"OUT_MIG"],"MIGRATION"] <- "MIG"
#    }
```

```
#
#    #Select a subset of fields for merging back to ARGOS_CTCRW_SIM_TRACKS
#    SIM_TRACKS_temp <-
SIM_TRACKS_temp[,c("OBS_TIME_DIFF","TRACK","CTCRW_FIT","PTT","SPP","REGION","MIGRA
TION")]
#
#    # Append SIM_TRACKS_temp data fields to each ARGOS_CTCRW_SIM_TRACKS
SpatialPointsDataFrame
#    ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep =
"")]][[paste("TRACK_",j,sep="")]] <-
cbind(ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep =
"")]][[paste("TRACK_",j,sep="")]],
#                                                            SIM_TRACKS_temp)
#
#   }
#
# }
#
# remove(SIM_TRACKS_temp,i,j)
#
#

# Reconstitute STPP_SIM_TRACKS from saved copy
ARGOS_CTCRW_SIM_TRACKS_FIXED <- readRDS(file = paste("/Users/trevor.joyce/Grad
School/Research/",
                        "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                        "ARGOS_CTCRW_SIM_TRACKS_FIXED.rds",sep=""))


##### ARGOS_CTCRW_SIM_UD #####

ARGOS_CTCRW_SIM_UD <- list()
ARGOS_CTCRW_SIM_UD_ERROR = data.frame()

## Generate Simulated UD for PTT from the West Antarctic Peninsula (WAP) region

for(i in names(ARGOS_CTCRW_SIM_TRACKS_FIXED)){

  FIXED_TRACKS_temp <- do.call("rbind",ARGOS_CTCRW_SIM_TRACKS_FIXED[[i]])
  FIXED_TRACKS_temp <- FIXED_TRACKS_temp[FIXED_TRACKS_temp$locType ==
"p",c("LONG_FIXED","LAT_FIXED")]

  SIM_UD_temp<- try(raster::rasterize(x = FIXED_TRACKS_temp,
                        y = raster::raster(ext = raster::extent(-71,-51,-70,-60),
```

```
                                         nrow=500,ncol=450,
                                         crs = sp::CRS("+init=epsg:4326")),
                           fun = "count")[[1]])
  if(class(SIM_UD_temp) == "try-error"){

    ARGOS_CTCRW_SIM_UD_ERROR = rbind(ARGOS_CTCRW_SIM_UD_ERROR,data.frame(PTT =
substr(i,11,16),
                                                     ERROR_TYPE = "rasterize"))
    next
  }

  ARGOS_CTCRW_SIM_UD[[i]] <- SIM_UD_temp
}

# Clean-up
remove(i, SIM_UD_temp, FIXED_TRACKS_temp)

# ## Generate Simulated UD for PTT from the Ross Sea (Ross) region
#
# for(i in names(ARGOS_CTCRW_SIM)[which(substr(names(ARGOS_CTCRW_SIM),11,16) %in%
TAGS[TAGS$Region == "Ross","PTT"])])
# { SIM_UD_temp<- try(raster::rasterize(x =
spTransform(do.call("rbind",ARGOS_CTCRW_SIM_TRACKS[[i]]),
#                                       CRS("+init=epsg:4326")),
#                            y = raster::raster(ext = raster::extent(160,180,-79,-69),
#                                         nrow=500,ncol=450,
#                                         crs = CRS("+init=epsg:4326")),
#                            fun = "count")[[1]])
# if(class(SIM_UD_temp) == "try-error")
# {ARGOS_CTCRW_SIM_UD_ERROR = rbind(ARGOS_CTCRW_SIM_UD_ERROR,data.frame(PTT =
substr(i,11,16),
#                                                    ERROR_TYPE = "spTransform"))
# next}
#
# ARGOS_CTCRW_SIM_UD[[i]] <- SIM_UD_temp
# }
#
# # Clean-up
# remove(i, SIM_UD_temp)

#Aggregrate ARGOS_CTCRW_SIM_UD by species (SPP)
for(i in c("KWA", "KWB", "KWG", "Mn",  "Bb")){
```

```
  ARGOS_CTCRW_SIM_UD[[i]] =
raster::brick(ARGOS_CTCRW_SIM_UD[substr(names(ARGOS_CTCRW_SIM_UD),11,16)%in%TAG
S[TAGS$SPP == i
                                                        & TAGS$Region %in%
c("WAP","Weddell"),"PTT"]])
  ARGOS_CTCRW_SIM_UD[[i]] = sum(ARGOS_CTCRW_SIM_UD[[i]],na.rm = T)
  ARGOS_CTCRW_SIM_UD[[i]][ARGOS_CTCRW_SIM_UD[[i]]==0]<-NA
}

# Clean-up
remove(i)

# for(i in c("KWC")){
#   ARGOS_CTCRW_SIM_UD[[i]] =
raster::brick(ARGOS_CTCRW_SIM_UD[substr(names(ARGOS_CTCRW_SIM_UD),11,16)%in%TAG
S[TAGS$SPP == i
#                                                         & TAGS$Region %in%
c("Ross"),"PTT"]])
#   ARGOS_CTCRW_SIM_UD[[i]] = sum(ARGOS_CTCRW_SIM_UD[[i]],na.rm = T)
#   ARGOS_CTCRW_SIM_UD[[i]][ARGOS_CTCRW_SIM_UD[[i]]==0]<-NA
# }

# Clean-up
remove(i)



#Aggregrate ARGOS_CTCRW_SIM_UD by species (SPP)
for(i in c("Mn",  "Bb")){
  ARGOS_CTCRW_SIM_UD[[paste(i,2013,sep="_")]] =
raster::brick(ARGOS_CTCRW_SIM_UD[substr(names(ARGOS_CTCRW_SIM_UD),11,16)%in%uniq
ue(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == i
                                                                &
lubridate::year(STPP_SIM_TRACKS$Time) == 2013
                                                                &
lubridate::month(STPP_SIM_TRACKS$Time) <= 8,"PTT"])])
  ARGOS_CTCRW_SIM_UD[[paste(i,2013,sep="_")]] =
sum(ARGOS_CTCRW_SIM_UD[[paste(i,2013,sep="_")]],na.rm = T)
  #
ARGOS_CTCRW_SIM_UD[[paste(i,2013,sep="_")]][ARGOS_CTCRW_SIM_UD[[paste(i,2013,sep="
_")]]==0]<-NA

  raster::writeRaster(ARGOS_CTCRW_SIM_UD[[paste(i,2013,sep="_")]],
            filename=paste("/Users/trevor.joyce/Grad School/Research/",
```

```
                        "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                        "ARGOS_CTCRW_SIM_UD_",i,"_",2013,".tif",sep=""),
                format = "GTiff", overwrite=TRUE)
}




######## SIM_UD_QUANTILE_VALUE #######

# Function to calculate the UD cell values (i.e., counts of simulated points)
# containing a specified quantile of the simulated locations

SIM_UD_QUANTILE_VALUE <- function(SIM_UD,QUANTILE){

  # Transform UD cell values into a dataframe ordered from largest to smallest values
  # and calculate the cumulative sum of those values from largest to smallest values
  SIM_UD_DF= data.frame(VALUES = rev(sort(as.vector(SIM_UD))),
                CUM_SUM = cumsum(rev(sort(as.vector(SIM_UD)))))

  # Calculate the sum of all UD cell values
  SIM_UD_DF$TOTAL = sum(SIM_UD_DF$VALUES)

  # Output the cell value of the first cell where the cumulative sum
  # is greater than or equal to specified quantile of the total
  return(SIM_UD_DF[SIM_UD_DF$CUM_SUM >= SIM_UD_DF$TOTAL[1]*QUANTILE
,"VALUES"][1])
}


######  SIM_UD_Mn_2013_90 ######

# Calculate the UD cell value containing 90% of the simulated locations
SIM_UD_QUANTILE_VALUE(SIM_UD = ARGOS_CTCRW_SIM_UD[["Mn_2013"]],QUANTILE = 0.9)
# [1] 52

# Import 90th percentile UD contour generated in ArcGIS as a SpatialLinesDataFrame
SIM_UD_Mn_2013_90 <- rgdal::readOGR(dsn=paste("/Users/trevor.joyce/Grad
School/Research/",
                          "1_2016_Antarctic Whale Tracking/Point Process Model
Example/SIM_UD",sep=""),
                    layer="SIM_UD_Mn_2013_90")

# Convert from SpatialLinesDataFrame to SpatialPolygons object
```

```
SIM_UD_Mn_2013_90 <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(SIM_UD_Mn_2013_90))

# Add a dataframe with polygon IDs based on SpatialPolygons object names to create a
SpatialPolygonsDataFrame
SIM_UD_Mn_2013_90 <- sp::SpatialPolygonsDataFrame(SIM_UD_Mn_2013_90,data.frame(ID =
names(SIM_UD_Mn_2013_90)))
raster::projection(SIM_UD_Mn_2013_90) <- sp::CRS("+init=epsg:4326")

# Collapse overlapping polygons (one which should be holes)
SIM_UD_Mn_2013_90 <- rgeos::gUnaryUnion(SIM_UD_Mn_2013_90)

# Import the holes in 90th percentile UD polygons (hand selected and exported in ArcGIS) as a
SpatialLinesDataFrame
SIM_UD_Mn_2013_90_HOLES <- rgdal::readOGR(dsn=paste("/Users/trevor.joyce/Grad
School/Research/",
                                   "1_2016_Antarctic Whale Tracking/Point Process Model
Example/SIM_UD",sep=""),
                              layer="SIM_UD_Mn_2013_90_HOLES")

# Convert from SpatialLinesDataFrame to SpatialPolygons object
SIM_UD_Mn_2013_90_HOLES <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(SIM_UD_Mn_2013_90_HOL
ES))

# Add a dataframe with polygon IDs based on SpatialPolygons object names to create a
SpatialPolygonsDataFrame
SIM_UD_Mn_2013_90_HOLES <-
sp::SpatialPolygonsDataFrame(SIM_UD_Mn_2013_90_HOLES,data.frame(ID =
names(SIM_UD_Mn_2013_90_HOLES)))
raster::projection(SIM_UD_Mn_2013_90_HOLES) <- sp::CRS("+init=epsg:4326")

# Collapse overlapping polygons
SIM_UD_Mn_2013_90_HOLES <- rgeos::gUnaryUnion(SIM_UD_Mn_2013_90_HOLES)

# Take the geographic difference (non-overlapping areas) to create holes in 90th percentile UD
contour
SIM_UD_Mn_2013_90 <-
rgeos::gSymdifference(SIM_UD_Mn_2013_90,SIM_UD_Mn_2013_90_HOLES)


######  SIM_UD_Mn_2013_50 ######

# Calculate the UD cell value containing 50% of the simulated locations
```

```
SIM_UD_QUANTILE_VALUE(SIM_UD = ARGOS_CTCRW_SIM_UD[["Mn_2013"]],QUANTILE = 0.5)
# [1] 412

# Import 50th percentile UD contour generated in ArcGIS as a SpatialLinesDataFrame
SIM_UD_Mn_2013_50 <- rgdal::readOGR(dsn=paste("/Users/trevor.joyce/Grad
School/Research/",
                                    "1_2016_Antarctic Whale Tracking/Point Process Model
Example/SIM_UD",sep=""),
                            layer="SIM_UD_Mn_2013_50")

# Convert from SpatialLinesDataFrame to SpatialPolygons object
SIM_UD_Mn_2013_50 <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(SIM_UD_Mn_2013_50))

# Add a dataframe with polygon IDs based on SpatialPolygons object names to create a
SpatialPolygonsDataFrame
SIM_UD_Mn_2013_50 <- sp::SpatialPolygonsDataFrame(SIM_UD_Mn_2013_50,data.frame(ID =
names(SIM_UD_Mn_2013_50)))
raster::projection(SIM_UD_Mn_2013_50) <- sp::CRS("+init=epsg:4326")

# Collapse overlapping polygons (one which should be holes)
SIM_UD_Mn_2013_50 <- rgeos::gUnaryUnion(SIM_UD_Mn_2013_50)

# Import the holes in 50th percentile UD polygons (hand selected and exported in ArcGIS) as a
SpatialLinesDataFrame
SIM_UD_Mn_2013_50_HOLES <- rgdal::readOGR(dsn=paste("/Users/trevor.joyce/Grad
School/Research/",
                                         "1_2016_Antarctic Whale Tracking/Point Process Model
Example/SIM_UD",sep=""),
                                 layer="SIM_UD_Mn_2013_50_HOLES")

# Convert from SpatialLinesDataFrame to SpatialPolygons object
SIM_UD_Mn_2013_50_HOLES <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(SIM_UD_Mn_2013_50_HOL
ES))

# Add a dataframe with polygon IDs based on SpatialPolygons object names to create a
SpatialPolygonsDataFrame
SIM_UD_Mn_2013_50_HOLES <-
sp::SpatialPolygonsDataFrame(SIM_UD_Mn_2013_50_HOLES,data.frame(ID =
names(SIM_UD_Mn_2013_50_HOLES)))
raster::projection(SIM_UD_Mn_2013_50_HOLES) <- sp::CRS("+init=epsg:4326")

# Collapse overlapping polygons
```

```
SIM_UD_Mn_2013_50_HOLES <- rgeos::gUnaryUnion(SIM_UD_Mn_2013_50_HOLES)

# Take the geographic difference (non-overlapping areas) to create holes in 50th percentile UD
contour
SIM_UD_Mn_2013_50 <-
rgeos::gSymdifference(SIM_UD_Mn_2013_50,SIM_UD_Mn_2013_50_HOLES)


######  SIM_UD_Bb_2013_90 ######

# Calculate the UD cell value containing 90% of the simulated locations
SIM_UD_QUANTILE_VALUE(SIM_UD = ARGOS_CTCRW_SIM_UD[["Bb_2013"]],QUANTILE = 0.9)
# [1] 39

# Import 90th percentile UD contour generated in ArcGIS as a SpatialLinesDataFrame
SIM_UD_Bb_2013_90 <- rgdal::readOGR(dsn=paste("/Users/trevor.joyce/Grad
School/Research/",
                                   "1_2016_Antarctic Whale Tracking/Point Process Model
Example/SIM_UD",sep=""),
                             layer="SIM_UD_Bb_2013_90")

# Convert from SpatialLinesDataFrame to SpatialPolygons object
SIM_UD_Bb_2013_90 <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(SIM_UD_Bb_2013_90))

# Add a dataframe with polygon IDs based on SpatialPolygons object names to create a
SpatialPolygonsDataFrame
SIM_UD_Bb_2013_90 <- sp::SpatialPolygonsDataFrame(SIM_UD_Bb_2013_90,data.frame(ID =
names(SIM_UD_Bb_2013_90)))
raster::projection(SIM_UD_Bb_2013_90) <- sp::CRS("+init=epsg:4326")

# Collapse overlapping polygons (one which should be holes)
SIM_UD_Bb_2013_90 <- rgeos::gUnaryUnion(SIM_UD_Bb_2013_90)

# Import the holes in 90th percentile UD polygons (hand selected and exported in ArcGIS) as a
SpatialLinesDataFrame
SIM_UD_Bb_2013_90_HOLES <- rgdal::readOGR(dsn=paste("/Users/trevor.joyce/Grad
School/Research/",
                                      "1_2016_Antarctic Whale Tracking/Point Process Model
Example/SIM_UD",sep=""),
                                layer="SIM_UD_Bb_2013_90_HOLES")

# Convert from SpatialLinesDataFrame to SpatialPolygons object
```

```
SIM_UD_Bb_2013_90_HOLES <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(SIM_UD_Bb_2013_90_HOL
ES))

# Add a dataframe with polygon IDs based on SpatialPolygons object names to create a
SpatialPolygonsDataFrame
SIM_UD_Bb_2013_90_HOLES <-
sp::SpatialPolygonsDataFrame(SIM_UD_Bb_2013_90_HOLES,data.frame(ID =
names(SIM_UD_Bb_2013_90_HOLES)))
raster::projection(SIM_UD_Bb_2013_90_HOLES) <- sp::CRS("+init=epsg:4326")

# Collapse overlapping polygons
SIM_UD_Bb_2013_90_HOLES <- rgeos::gUnaryUnion(SIM_UD_Bb_2013_90_HOLES)

# Take the geographic difference (non-overlapping areas) to create holes in 90th percentile UD
contour
SIM_UD_Bb_2013_90 <-
rgeos::gSymdifference(SIM_UD_Bb_2013_90,SIM_UD_Bb_2013_90_HOLES)


######  SIM_UD_Bb_2013_50 ######

# Calculate the UD cell value containing 50% of the simulated locations
SIM_UD_QUANTILE_VALUE(SIM_UD = ARGOS_CTCRW_SIM_UD[["Bb_2013"]],QUANTILE = 0.5)
# [1] 740

# Import 50th percentile UD contour generated in ArcGIS as a SpatialLinesDataFrame
SIM_UD_Bb_2013_50 <- rgdal::readOGR(dsn=paste("/Users/trevor.joyce/Grad
School/Research/",
                                   "1_2016_Antarctic Whale Tracking/Point Process Model
Example/SIM_UD",sep=""),
                            layer="SIM_UD_Bb_2013_50")

# Convert from SpatialLinesDataFrame to SpatialPolygons object
SIM_UD_Bb_2013_50 <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(SIM_UD_Bb_2013_50))

# Add a dataframe with polygon IDs based on SpatialPolygons object names to create a
SpatialPolygonsDataFrame
SIM_UD_Bb_2013_50 <- sp::SpatialPolygonsDataFrame(SIM_UD_Bb_2013_50,data.frame(ID =
names(SIM_UD_Bb_2013_50)))
raster::projection(SIM_UD_Bb_2013_50) <- sp::CRS("+init=epsg:4326")

# Collapse overlapping polygons (one which should be holes)
```

```
SIM_UD_Bb_2013_50 <- rgeos::gUnaryUnion(SIM_UD_Bb_2013_50)

# Import the holes in 50th percentile UD polygons (hand selected and exported in ArcGIS) as a
SpatialLinesDataFrame
SIM_UD_Bb_2013_50_HOLES <- rgdal::readOGR(dsn=paste("/Users/trevor.joyce/Grad
School/Research/",
                                  "1_2016_Antarctic Whale Tracking/Point Process Model
Example/SIM_UD",sep=""),
                            layer="SIM_UD_Bb_2013_50_HOLES")

# Convert from SpatialLinesDataFrame to SpatialPolygons object
SIM_UD_Bb_2013_50_HOLES <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(SIM_UD_Bb_2013_50_HOL
ES))

# Add a dataframe with polygon IDs based on SpatialPolygons object names to create a
SpatialPolygonsDataFrame
SIM_UD_Bb_2013_50_HOLES <-
sp::SpatialPolygonsDataFrame(SIM_UD_Bb_2013_50_HOLES,data.frame(ID =
names(SIM_UD_Bb_2013_50_HOLES)))
raster::projection(SIM_UD_Bb_2013_50_HOLES) <- sp::CRS("+init=epsg:4326")

# Collapse overlapping polygons
SIM_UD_Bb_2013_50_HOLES <- rgeos::gUnaryUnion(SIM_UD_Bb_2013_50_HOLES)

# Take the geographic difference (non-overlapping areas) to create holes in 50th percentile UD
contour
SIM_UD_Bb_2013_50 <-
rgeos::gSymdifference(SIM_UD_Bb_2013_50,SIM_UD_Bb_2013_50_HOLES)

sp::plot(rgeos::gSymdifference(SIM_UD_Bb_2013_50,SIM_UD_Mn_2013_50),col = "green")


###### ICE_MODIS_LAND_MASK_RES_2000m ####

# Raster LAND_MASK at approximately 2000m resolution
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)
ICE_MODIS_LAND_MASK_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad
School/Research/",
                                  "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                                  "ICE_MODIS_LAND_MASK_RES_2000m.tif",sep=""))

###### WAP_MASK_RES_2000m ####
```

```
# Raster LAND_MASK that also excludes the WEDDELL_SEA at approximately 2000m resolution
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

WAP_MASK_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad School/Research/",
                        "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                        "WAP_MASK_RES_2000m.tif",sep=""))


###### ICE_MODIS_CONC_COMP_2013_RES_2000m  #############

# Annual composite layers of ICE_MODIS concentration at approximately 2000m resolution
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

ICE_MODIS_CONC_COMP_2013_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad
School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                              "ICE_MODIS_CONC_COMP_2013_RES_2000m.tif",sep=""))

###### ICE_MODIS_CONC_COMP_2016_RES_2000m  #############

# Annual composite layers of ICE_MODIS concentration index at approximately 2000m
resolution
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

ICE_MODIS_CONC_COMP_2016_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad
School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                              "ICE_MODIS_CONC_COMP_2016_RES_2000m.tif",sep=""))

#### MIN_DIST_DIR_MEAN_RES_2000m #####

# Reconstitute MIN_DIST_DIR_MEAN_RES_2000m from saved raster
MIN_DIST_DIR_MEAN_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad
School/Research/",
                           "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                           "MIN_DIST_DIR_MEAN_RES_2000m.tif",sep=""))


#### MIN_DIST_DIR_MED_RES_2000m #####

# Reconstitute MIN_DIST_DIR_MED_RES_2000m from saved raster
MIN_DIST_DIR_MED_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad
School/Research/",
                           "1_2016_Antarctic Whale Tracking/Data/Coastline/",
```

```
                                  "MIN_DIST_DIR_MED_RES_2000m.tif",sep=""))


###### COAST_CLASS_RES_2000m #######

# Reconstitute COAST_CLASS_RES_2000m from saved raster
COAST_CLASS_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad School/Research/",
                                  "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                                  "COAST_CLASS_RES_2000m.tif",sep=""))

###### COAST_CLASS_RES_2000m #######

# Reconstitute COAST_CLASS_GEN_RES_2000m from saved raster
COAST_CLASS_GEN_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad
School/Research/",
                                    "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                                    "COAST_CLASS_GEN_RES_2000m.tif",sep=""))


###### DIST_ICE_MODIS_RES_2000m #########

# Distances to contours (15% and 30%) of annual composite layers of ICE_MODIS
# concentration at approximately 2000m resolution
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

# Reconstitute DIST_ICE_MODIS_RES_2000m from individual rasters

DIST_ICE_MODIS_RES_2000m<- raster::stack()

for(i in list.files(paste("/Users/trevor.joyce/Grad School/Research/",
                  "1_2016_Antarctic Whale Tracking/Data/Ice
Data/DIST_ICE_MODIS_RES_2000m/",sep=""))){


DIST_ICE_MODIS_RES_2000m_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevor.joyce
/Grad School/Research/",
                                                "1_2016_Antarctic Whale Tracking/Data/Ice
Data/DIST_ICE_MODIS_RES_2000m/",i,sep="")))

  names(DIST_ICE_MODIS_RES_2000m_temp) <- substr(i,26,33)

  DIST_ICE_MODIS_RES_2000m <-
raster::stack(DIST_ICE_MODIS_RES_2000m,DIST_ICE_MODIS_RES_2000m_temp)
```

```
}

remove(DIST_ICE_MODIS_RES_2000m_temp,i)


####### BATHY_IBCSO_RES_2000m #########

# Bathymetric depths from International Bathymetric Chart of the Southern Ocean
# reprojected at an approximately 2000m resolution
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

# Reconstitute BATHY_IBCSO_RES_2000m from saved raster
BATHY_IBCSO_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad School/Research/",
                        "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                        "BATHY_IBCSO_RES_2000m.tif",sep=""))

####### SLOPE_IBCSO_RES_2000m #########

# Bathymetric slopes from International Bathymetric Chart of the Southern Ocean
# reprojected at an approximately 2000m resolution
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

# Reconstitute SLOPE_IBCSO_RES_2000m from saved raster
SLOPE_IBCSO_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad School/Research/",
                        "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                        "SLOPE_IBCSO_RES_2000m.tif",sep=""))

####### PROFILE_IBCSO_RES_2000m #########

# Bathymetric profile curvature from International Bathymetric Chart of the Southern Ocean
# reprojected at an approximately 2000m resolution
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

# Reconstitute PROFILE_IBCSO_RES_2000m from saved raster
PROFILE_IBCSO_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad
School/Research/",
                          "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                          "PROFILE_IBCSO_RES_2000m.tif",sep=""))

####### DIST_CONTINENTAL_SHELF_RES_2000m #########

# Distance to the edge of the Continental Shelf (defined at 450m bathymetric depth contour)
# based on the International Bathymetric Chart of the Southern Ocean.
# This version includes internal structure within outer shelf boundaries (e.g., shelf deeps).
```

```
# This has been reprojected at 2000m resolution.
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.6.R)

# Reconstitute DIST_CONTINENTAL_SHELF_RES_2000m from saved raster
DIST_CONTINENTAL_SHELF_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad
School/Research/",
                                    "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                                    "DIST_CONTINENTAL_SHELF_RES_2000m.tif",sep=""))

###### DIST_CONTINENTAL_SHELF_OUTER_RES_2000m #########

# Distance to the edge of the Continental Shelf (defined at 450m bathymetric depth contour)
# based on the International Bathymetric Chart of the Southern Ocean.
# This version includes only outer shelf margin and valleys that cut into shelf.
# This has been reprojected at 2000m resolution.
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.6.R)

# Reconstitute DIST_CONTINENTAL_SHELF_OUTER_RES_2000m from saved raster
DIST_CONTINENTAL_SHELF_OUTER_RES_2000m =
raster::raster(paste("/Users/trevor.joyce/Grad School/Research/",
                                       "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",

"DIST_CONTINENTAL_SHELF_OUTER_RES_2000m.tif",sep=""))

###### DIST_LAND_RES_2000m #########

# Distance to a COAST outline file derived from the British Antarctic Survey Geodata Portal.
# This has been reprojected at 2000m resolution.
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

# Reconstitute DIST_LAND_RES_2000m from saved raster
DIST_LAND_RES_2000m = raster::raster(paste("/Users/trevor.joyce/Grad School/Research/",
                             "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                             "DIST_LAND_RES_2000m.tif",sep=""))

###### CHL_IDW_INTERP_RES_2000m #########

# Seasonal composite layers of MODIS chlorophyll concentration
# reprojected at approximately 2000m resolution.
# (see calculation details in Minke_Humpback_STPP_Covariate_Processing_v2.3.4.R)

# Reconstitute CHL_IDW_INTERP_RES_2000m from individual saved rasters
CHL_IDW_INTERP_RES_2000m<- raster::stack()
```

```
for(i in list.files(paste("/Users/trevor.joyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Data/Chl
Data/CHL_IDW_INTERP_RES_2000m/",sep=""))){


CHL_IDW_INTERP_RES_2000m_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevor.joyc
e/Grad School/Research/",
                                                    "1_2016_Antarctic Whale Tracking/Data/Chl
Data/CHL_IDW_INTERP_RES_2000m/",i,sep="")))

  names(CHL_IDW_INTERP_RES_2000m_temp) <- substr(i,26,30)

  CHL_IDW_INTERP_RES_2000m <-
raster::stack(CHL_IDW_INTERP_RES_2000m,CHL_IDW_INTERP_RES_2000m_temp)

}

remove(CHL_IDW_INTERP_RES_2000m_temp,i)


###### STPP_COVAR ######

STPP_COVAR <- raster::stack(DIST_LAND_RES_2000m)
names(STPP_COVAR) <- c("DIST_LAND")

STPP_COVAR[["DIST_LAND"]] <- DIST_LAND_RES_2000m

STPP_COVAR[["DIST_SHELF"]] <- DIST_CONTINENTAL_SHELF_RES_2000m

STPP_COVAR[["DIST_SHELF_OUTER"]] <- DIST_CONTINENTAL_SHELF_OUTER_RES_2000m

STPP_COVAR[["BATHY"]] <- BATHY_IBCSO_RES_2000m

STPP_COVAR[["SLOPE"]] <- SLOPE_IBCSO_RES_2000m

STPP_COVAR[["PROFILE"]] <- PROFILE_IBCSO_RES_2000m

STPP_COVAR[["ICE_CONC_2013"]] <- ICE_MODIS_CONC_COMP_2013_RES_2000m

STPP_COVAR[["ICE_CONC_2016"]] <- ICE_MODIS_CONC_COMP_2016_RES_2000m

STPP_COVAR[["DIST_ICE_2013_30"]] <- DIST_ICE_MODIS_RES_2000m[["X2013_30"]]

STPP_COVAR[["DIST_ICE_2016_30"]] <- DIST_ICE_MODIS_RES_2000m[["X2016_30"]]
```

```
STPP_COVAR[["DIST_ICE_2013_15"]] <- DIST_ICE_MODIS_RES_2000m[["X2013_15"]]

STPP_COVAR[["DIST_ICE_2016_15"]] <- DIST_ICE_MODIS_RES_2000m[["X2016_15"]]

STPP_COVAR[["CHL_2013"]] <- CHL_IDW_INTERP_RES_2000m[["X2013"]]

STPP_COVAR[["CHL_2016"]] <- CHL_IDW_INTERP_RES_2000m[["X2016"]]

STPP_COVAR[["COAST_CLASS"]] <- COAST_CLASS_RES_2000m

STPP_COVAR[["COAST_CLASS_GEN"]] <- COAST_CLASS_GEN_RES_2000m

STPP_COVAR[["ENCLOSURE_MEAN"]] <- 1 - MIN_DIST_DIR_MEAN_RES_2000m

STPP_COVAR[["ENCLOSURE_MED"]] <- 1 - MIN_DIST_DIR_MED_RES_2000m

STPP_COVAR[["OCEAN"]] <-  ICE_MODIS_LAND_MASK_RES_2000m
STPP_COVAR[["OCEAN"]][!is.na(STPP_COVAR[["OCEAN"]])]<-1
STPP_COVAR[["OCEAN"]][is.na(STPP_COVAR[["OCEAN"]])]<-0

STPP_COVAR[["WAP"]] <-  WAP_MASK_RES_2000m

STPP_COVAR[["INT"]] = STPP_COVAR[["OCEAN"]]
STPP_COVAR[["INT"]][] = 1

STPP_COVAR[["LONG"]] <- STPP_COVAR[["INT"]]
STPP_COVAR[["LONG"]][] <- sp::coordinates(STPP_COVAR)[,1]

STPP_COVAR[["LAT"]] <- STPP_COVAR[["INT"]]
STPP_COVAR[["LAT"]][] <- sp::coordinates(STPP_COVAR)[,2]

STPP_COVAR[["DIST_N"]] <- STPP_COVAR[["INT"]]
STPP_COVAR[["DIST_N"]][] <- ifelse(sp::coordinates(STPP_COVAR)[,2]>0,1,-1) *
geosphere::distGeo(p1 = data.frame(X = sp::coordinates(STPP_COVAR)[,1],
                                                                  Y = 0),
                                                        p2 = data.frame(X =
sp::coordinates(STPP_COVAR)[,1],
                                                                  Y = sp::coordinates(STPP_COVAR)[,2]))

STPP_COVAR[["DIST_E"]] <- STPP_COVAR[["INT"]]
STPP_COVAR[["DIST_E"]][] <- ifelse(sp::coordinates(STPP_COVAR)[,1]>0,1,-1) *
geosphere::distGeo(p1 = data.frame(X = 0,
                                                                  Y = sp::coordinates(STPP_COVAR)[,2]),
```

```
                                          p2 = data.frame(X =
sp::coordinates(STPP_COVAR)[,1],
                                                Y = sp::coordinates(STPP_COVAR)[,2]))

##### FIX_SIM_TRACKS ######

#Function to LAND correct simulated track
FIX_SIM_TRACKS <- function(X, DATA, LAND_RAST, TRANS_MATRIX){

  #Take a subset of DATA including only a single TRACK
  DATA <- DATA[DATA$TRACK == paste("TRACK_",X,sep = ""),]

  #Create a column that indicates whether each point in simulated TRACK intersects LAND
  DATA$LAND <- raster::extract(x = LAND_RAST,
                       y = DATA[,c("LONG","LAT")])

  # check if the simulated track does indeed cross LAND (raster::extract()
  # will return a 1 if the track crosses land raster cells), and
  # if it doesn't cross-land directly return original coordinates
  # (avoids crawl::fix_path error when there are no segments to fix)
  if(!1%in%DATA$LAND){

    # If it doesn't cross land just return the original points
    DATA <- DATA[,c("LAND","LONG","LAT")]
    colnames(DATA) <- c("LAND","LONG_FIXED","LAT_FIXED")
    return(DATA)

  }

  # if path does cross LAND -> run the crawl::fix_path algorithm and then merge the results back
to DATA
  if(1%in%DATA$LAND){

    # run the crawl::fix_path algorithm to find the shortest path around
    # LAND_RAST cells using the TRANS_MATRIX calculated above
    SIM_PATH_FIXED <- crawl::fix_path(xy = as.matrix(DATA[,c("LONG","LAT")]),
                          time = DATA[,c("Time")],
                          res_raster = LAND_RAST,
                          trans = TRANS_MATRIX)

    colnames(SIM_PATH_FIXED) <- c("LONG_FIXED","LAT_FIXED", "Time")

    # Occasionally crawl::fix_path will drop one or more records because of the following error
    # "Path ends in restricted area, last  1  observations removed"
```

```
    # Implement merge based on "Time" to return a data.frame of the same length as the input
DATA
    DATA <- merge(x = DATA[,c("Time","LAND")],
                  y = SIM_PATH_FIXED,
                  by = "Time", all.x = T)

    return(DATA[,c("LAND","LONG_FIXED","LAT_FIXED")])
  }


}




###### STPP_SIM_TRACKS ######

# # Create list that will house data.frames of simulated tracks
# # processed for use in STPP_KERNEL_DF and STPP_COUNT
#
# STPP_SIM_TRACKS = data.frame()
#
# for(i in unique(ARGOS_CTCRW[ARGOS_CTCRW$SPP%in%c("Bb","Mn")
#                       & !ARGOS_CTCRW$PTT%in%ARGOS_CTCRW_SIM_UD_ERROR$PTT
#                       & ARGOS_CTCRW$PTT%in%substr(names(ARGOS_CTCRW_SIM),11,16)
#                       & ARGOS_CTCRW$REGION%in%c("WAP","Weddell"),"PTT"])){
#   for(j in 1:10){
#
#     #Pull each simulated track from ARGOS_CTCRW_SIM_TRACKS
#     STPP_SIM_TRACKS_temp <- ARGOS_CTCRW_SIM_TRACKS[[paste("CTCRW_FIT_",i,sep =
""))]][[paste("TRACK_",j,sep="")]]
#
#     #Convert simulated tracks into GCS_WGS84
#     STPP_SIM_TRACKS_temp <- sp::spTransform(STPP_SIM_TRACKS_temp,
#                                     CRS("+init=epsg:4326"))
#
#     # convert to data.frame to allow diff and na.locf functions (don't work with
SpatialPointsDataFrame)
#     STPP_SIM_TRACKS_temp <- as.data.frame(STPP_SIM_TRACKS_temp)
#
#     # calculate the time difference between observed Argos fixes
#     STPP_SIM_TRACKS_temp[STPP_SIM_TRACKS_temp$locType=="o","OBS_TIME_DIFF"] <-
c(as.numeric(diff(STPP_SIM_TRACKS_temp[STPP_SIM_TRACKS_temp$locType=="o","Time"],
#                                                         units = "secs")),0)
```

```
#    # copy down OBS_TIME_DIFF to locType=="p" rows
#    STPP_SIM_TRACKS_temp$OBS_TIME_DIFF <-
zoo::na.locf(STPP_SIM_TRACKS_temp$OBS_TIME_DIFF)
#
#    # convert OBS_TIME_DIFF to units of hours
#    STPP_SIM_TRACKS_temp$OBS_TIME_DIFF <-
STPP_SIM_TRACKS_temp$OBS_TIME_DIFF/(60*60)
#
#    # Define TRACK for later multiple imputation implementation
#    STPP_SIM_TRACKS_temp$TRACK <- paste("TRACK_",j,sep="")
#
#    # Define CRW_FIT for later multiple imputation implementation
#    STPP_SIM_TRACKS_temp$CTCRW_FIT <- paste("CTCRW_FIT_",i,sep = "")
#
#    # Define PTT
#    STPP_SIM_TRACKS_temp$PTT <- i
#
#    # Define SPP
#    STPP_SIM_TRACKS_temp$SPP <- TAGS[TAGS$PTT == i,"SPP"][1]
#
#    # Label predictions as falling before (PRE_MIG) or after (MIG) the OUT_MIG date in
MIGRATION table
#    STPP_SIM_TRACKS_temp$MIGRATION <- "PRE_MIG"
#
#    if(!is.na(MIGRATION[MIGRATION$PTT == i, "OUT_MIG"])){
#      STPP_SIM_TRACKS_temp[STPP_SIM_TRACKS_temp$Time >=
MIGRATION[MIGRATION$PTT == i, "OUT_MIG"],"MIGRATION"] <- "MIG"
#    }
#
#    # select only the predictions (locType=="p")
#    STPP_SIM_TRACKS_temp <-
STPP_SIM_TRACKS_temp[STPP_SIM_TRACKS_temp$locType=="p",]
#
#    # select predictions at a 2-hour interval
#    # (i.e., every fourth prediction when original predictions are at 0.5 hour intervals)
#    STPP_SIM_TRACKS_temp <-
STPP_SIM_TRACKS_temp[seq(1,nrow(STPP_SIM_TRACKS_temp), by = 4),]
#
#    # Write projected track (STPP_SIM_TRACKS_temp) to STPP_SIM_TRACKS
#    STPP_SIM_TRACKS <- rbind(STPP_SIM_TRACKS, STPP_SIM_TRACKS_temp)
#
#   }
#
# }
```

```
#
# remove(STPP_SIM_TRACKS_temp,i,j)
#
# colnames(STPP_SIM_TRACKS)[colnames(STPP_SIM_TRACKS)%in%c("mu.x","mu.y")] <-
c("LONG","LAT")
#
#
# WEDDELL_SEA <- data.frame(LONG = c(-57.3,-56.4,-45.5,-14.0,-06.0,-28.0,-58.0,-65.0,-64.0,-
57.3),
#                   LAT =  c(-63.25,-63.0,-61.1,-63.5,-71.0,-78.0,-77.0,-73.0,-66.0,-63.25))
#
# STPP_SIM_TRACKS$REGION <- sp::point.in.polygon(point.x = STPP_SIM_TRACKS$LONG,
#                                         point.y = STPP_SIM_TRACKS$LAT,
#                                         pol.x = WEDDELL_SEA$LONG,
#                                         pol.y = WEDDELL_SEA$LAT)
#
#
# STPP_SIM_TRACKS$REGION <- ifelse(STPP_SIM_TRACKS$REGION >= 1, "WEDD","WAP")
#
#
# # Save kernel intensity values to a single overall .rds file
# # (avoids having to re-run calculations)
# saveRDS(object = STPP_SIM_TRACKS,
#       file = paste("/Users/trevor.joyce/Grad School/Research/",
#               "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
#               "STPP_SIM_TRACKS.rds",sep=""))
#
# # Reconstitute STPP_SIM_TRACKS from saved copy
# STPP_SIM_TRACKS <- readRDS(file = paste("/Users/trevor.joyce/Grad School/Research/",
#                           "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
#                           "STPP_SIM_TRACKS.rds",sep=""))
#
# # Extract a subset of TRACKS that will be used in STPP analysis
# STPP_SIM_TRACKS <- STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP %in% c("Mn","Bb")
#                       & lubridate::year(STPP_SIM_TRACKS$Time) %in% c(2013,2016)
#                       & lubridate::month(STPP_SIM_TRACKS$Time) <= 8
#                       & STPP_SIM_TRACKS$LONG > -78
#                       & STPP_SIM_TRACKS$LONG < -52
#                       & STPP_SIM_TRACKS$LAT > -71
#                       & STPP_SIM_TRACKS$LAT < -61,]
#
# #Create transition matrix
# TRANS_MATRIX_temp <- gdistance::transition(STPP_COVAR[["OCEAN"]],
#                             transitionFunction=prod, directions = 16)
```

```
#
# #Create LAND raster that is the opposite of OCEAN (LAND = 1, OCEAN = 0)
# LAND_RAST_temp <- abs(STPP_COVAR[["OCEAN"]]-1)
#
# # Create variables to house LAND corrected values
# STPP_SIM_TRACKS[,c("LAND","LONG_FIXED","LAT_FIXED")] <- NA
#
# # Loop through each CTCRW_FIT in STPP_SIM_TRACKS to run FIX_SIM_TRACKS function
# # which LAND corrects simulated TRACKS
# for(i in unique(STPP_SIM_TRACKS$CTCRW_FIT)){
#
#   print(i)
#   PROCESSING_TIME_temp <- proc.time()
#
#   # run FIX_SIM_TRACKS function which on each simulated TRACK within i CTCRW_FIT point
#   # that falls within the bbox of STPP_COVAR
#   STPP_SIM_TRACKS_FIXED_temp <- try(parallel::mclapply(X = 1:10,
#                                     DATA = STPP_SIM_TRACKS[STPP_SIM_TRACKS$CTCRW_FIT ==
i,],
#                                     LAND_RAST = LAND_RAST_temp,
#                                     TRANS_MATRIX = TRANS_MATRIX_temp,
#                                     FUN = FIX_SIM_TRACKS,
#                                     mc.cores = 4))
#   # Error-handling
#   if(class(STPP_SIM_TRACKS_FIXED_temp) == "try-error"){
#     print(STPP_SIM_TRACKS_FIXED_temp[1])
#     next
#   }
#
#   # Print information on PROCESSING_TIME associated with specific PTT and TRACK
#   print(proc.time() - PROCESSING_TIME_temp)
#
#   for (j in 1:10){
#     STPP_SIM_TRACKS[STPP_SIM_TRACKS$CTCRW_FIT == i
#              & STPP_SIM_TRACKS$TRACK == paste("TRACK_",j,sep =
""),c("LAND","LONG_FIXED","LAT_FIXED")] <- STPP_SIM_TRACKS_FIXED_temp[[j]]
#   }
#
#   remove(STPP_SIM_TRACKS_FIXED_temp,PROCESSING_TIME_temp); gc()
#
# }
#
# remove(TRANS_MATRIX_temp,LAND_RAST_temp,i,j); gc()
#
```

```
#
# # Change class of calculated variables
# STPP_SIM_TRACKS$LONG_FIXED <- as.numeric(STPP_SIM_TRACKS$LONG_FIXED)
# STPP_SIM_TRACKS$LAT_FIXED <- as.numeric(STPP_SIM_TRACKS$LAT_FIXED)
# STPP_SIM_TRACKS$LAND <- as.numeric(STPP_SIM_TRACKS$LAND)
#
# # Save kernel intensity values to a single overall .rds file
# # (avoids having to re-run calculations)
# saveRDS(object = STPP_SIM_TRACKS,
#       file = paste("/Users/trevor.joyce/Grad School/Research/",
#                 "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
#                 "STPP_SIM_TRACKS_FIXED.rds",sep=""))
#

# Reconstitute STPP_SIM_TRACKS from saved copy
STPP_SIM_TRACKS <- readRDS(file = paste("/Users/trevor.joyce/Grad School/Research/",
                            "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                            "STPP_SIM_TRACKS_FIXED.rds",sep=""))



# Save original coordinates
STPP_SIM_TRACKS$LONG_UNFIXED <- STPP_SIM_TRACKS$LONG
STPP_SIM_TRACKS$LAT_UNFIXED <- STPP_SIM_TRACKS$LAT

# Replace LONG and LAT with FIXED (land-corrected) coordinates for further analyses
STPP_SIM_TRACKS$LONG <- STPP_SIM_TRACKS$LONG_FIXED
STPP_SIM_TRACKS$LAT <- STPP_SIM_TRACKS$LAT_FIXED

# Remove any records where LONG and LAT (land-corrected) are NA
STPP_SIM_TRACKS <- STPP_SIM_TRACKS[!is.na(STPP_SIM_TRACKS$LONG),]

###### STPP_PHI ######

# Data.frame to house velocities (phi) constraining the movement kernel (in km/h)
STPP_PHI <- data.frame(SPP = c("Bb","Mn"), stringsAsFactors = F)

# Assign velocity as mean of measured speeds based on CTCRW predictions
STPP_PHI[STPP_PHI$SPP%in%c("Bb"),"MEAN_VEL"] <-
mean(ARGOS_CTCRW[ARGOS_CTCRW$SPP%in%c("Bb")
                                    & ARGOS_CTCRW$YEAR%in%c(2013,2016)
                                    & ARGOS_CTCRW$MIGRATION%in%c("PRE_MIG",
"POST_MIG")
```

```
                                & ARGOS_CTCRW$OBS_TIME_DIFF < 24,"VEL_GEO"], na.rm
=T)

STPP_PHI[STPP_PHI$SPP%in%c("Mn"),"MEAN_VEL"] <-
mean(ARGOS_CTCRW[ARGOS_CTCRW$SPP%in%c("Mn")
                                & ARGOS_CTCRW$YEAR%in%c(2013,2016)
                                & ARGOS_CTCRW$MIGRATION%in%c("PRE_MIG",
"POST_MIG")
                                & ARGOS_CTCRW$OBS_TIME_DIFF < 24,"VEL_GEO"], na.rm
=T)


# Assign velocity as maximum of 15km/h based on conservative maximum sustained speed
from Ford et al. 2005, Noad and Cato 2007
STPP_PHI[STPP_PHI$SPP%in%c("Bb"),"MAX_VEL"] <- 15

STPP_PHI[STPP_PHI$SPP%in%c("Mn"),"MAX_VEL"] <- 15



######  STPP_KERNEL_DF_CALC ######

#Function to calculate the redistribution STPP_KERNEL for each point in a simulated track
STPP_KERNEL_DF_CALC <- function(X, DATA, RAST, PHI){

 #pull the track coordinates at time [X-1]
 POINT = DATA[X-1,c("LONG","LAT")]

 # if point [i,j,X] does not fall within a duty cycle period (OBS_TIME_DIFF < 24)
 # calculate the kernel intensity availability surface (ln_k)
 if(DATA[X-1,c("OBS_TIME_DIFF")] < 24){

  #time separating previous [X-1] and current [X] location fixes (in sec)
  DELTA_T = as.numeric(difftime(DATA[X,"Time"],
                   DATA[X-1,"Time"],
                   units = "hours"))

  #squared distance in km of each coordinate in COVAR grid to SIM_TRACKS coordinate [X-1]
  DIST = geosphere::distGeo(p1 = POINT,
                 p2 = as(RAST, "SpatialPoints"))/1000
  DIST = DIST^2

  #numerical approximation of integral in spatial point process intensity function (eq 4.48)
  #"similar in shape to a bivariate normal density centered on µ" (Hooten et al. 2017)
```

```
    #describes availability based on movement constraint
    #xxx = delta*exp(-d2/(2*phi*delta)) - (d2/(2*phi))*gamma_inc(0, d2/(2*phi*delta))
    KERNEL = DELTA_T * exp(-DIST / (2 * PHI * DELTA_T)) -
      (DIST /(2 * PHI))*gsl::gamma_inc(0, DIST / (2 * PHI * DELTA_T))

    #change any division by 0 errors (NaN, for very small gamma_inc values) into 0
    KERNEL[is.nan(KERNEL)] = 0

    # temporarily convert KERNEL to a raster with the topology of the COVAR raster stacl
    KERNEL_RAST <- RAST
    KERNEL_RAST[] <- KERNEL

    # Extract CELL and location information from raster and then convert to a data.frame
    KERNEL_DF <- as.data.frame(as(KERNEL_RAST,"SpatialPointsDataFrame"),
                        stringsAsFactors = F)
    colnames(KERNEL_DF)[colnames(KERNEL_DF)%in%c("x","y")] <- c("LONG","LAT")
    KERNEL_DF$CELL <- raster::cellFromXY(object = KERNEL_RAST,
                              xy = KERNEL_DF[,c("LONG","LAT")])

    # Add additional identifier information to data.frame records for later subsetting
    KERNEL_DF$PTT <- as.data.frame(DATA)[X,"PTT"]
    KERNEL_DF$SPP <- as.data.frame(DATA)[X,"SPP"]
    KERNEL_DF$CTCRW_FIT <- as.data.frame(DATA)[X,"CTCRW_FIT"]
    KERNEL_DF$TRACK <- as.data.frame(DATA)[X,"TRACK"]
    KERNEL_DF$TIME <- as.data.frame(DATA)[X,"Time"]
    KERNEL_DF$LONG_SIM <- as.data.frame(DATA)[X,"LONG"]
    KERNEL_DF$LAT_SIM <- as.data.frame(DATA)[X,"LAT"]
    KERNEL_DF$DELTA_T <- DELTA_T
    KERNEL_DF$PHI <- PHI

    # Select only the cells with an intesity value >0.00001
    # to save in KERNEL_DF_Bb data.frame (cells with values <0.00001 will be treated as
essentially 0)
    KERNEL_DF <- KERNEL_DF[KERNEL_DF$INT> 1e-5, ]

    # return KERNEL_DF as a data.frame

    return(KERNEL_DF)

  }

}

######  STPP_KERNEL_DF ######
```

```
#
# STPP_KERNEL_DF <- list()
#
# # Loop through each Bb CTCRW model fit
# for(i in unique(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Bb","PTT"])){
#
#   # Create a list within STPP_KERNEL_DF list to house data.frames
#   # of kernel intensity values associated with each simulated TRACK
#   STPP_KERNEL_DF[[paste("CTCRW_FIT_",i,sep = "")]] <- list()
#
#   # Loop n = 10 simulated tracks drawn from CTCRW model fit
#   for(j in paste("TRACK_",1:10,sep="")){
#
#    # Error handling: for cases where there is no PRE_MIG data to calculate a kernel surface
#    if(nrow(STPP_SIM_TRACKS[STPP_SIM_TRACKS$PTT == i
#                       & STPP_SIM_TRACKS$TRACK == j
#                       & STPP_SIM_TRACKS$MIGRATION == "PRE_MIG",]) == 0){
#
#     # print error message
#     print("ERROR: no PRE_MIG data to calculate a kernel surface")
#
#     # clear list object created above
#     STPP_KERNEL_DF[[paste("CTCRW_FIT_",i,sep = "")]] <- NULL
#
#     # move to the next track
#     next
#    }
#
#   # Print information on PTT and TRACK to track progress
#   print(paste("Bb", i, j))
#   PROCESSING_TIME_temp <- proc.time()
#
#   # Run STPP_KERNEL_DF_CALC to calculate the  kernel intensity values
#   # associated with each point in simulated TRACK.
#   # mclapply is used to parallelize this calculation across multiple cores
#   STPP_KERNEL_DF_temp <- parallel::mclapply(X = seq(2,
#                                   nrow(STPP_SIM_TRACKS[STPP_SIM_TRACKS$PTT == i
#                                                 & STPP_SIM_TRACKS$TRACK == j
#                                                 & STPP_SIM_TRACKS$MIGRATION == "PRE_MIG",])),
#                             DATA = STPP_SIM_TRACKS[STPP_SIM_TRACKS$PTT == i
#                                               & STPP_SIM_TRACKS$TRACK == j
#                                               & STPP_SIM_TRACKS$MIGRATION == "PRE_MIG",],
#                             RAST = STPP_COVAR$INT,
#                             PHI = STPP_PHI[STPP_PHI$SPP == "Bb","MAX_VEL"],
```

```
#                                   FUN = STPP_KERNEL_DF_CALC,
#                                  mc.cores = 6)
#
#   # Print information on PROCESSING_TIME associated with specific PTT and TRACK
#   print(proc.time() - PROCESSING_TIME_temp)
#
#   # Concatenate list of data.frames containing the kernel intensity values
#   # associated with each point in simulated TRACK
#   STPP_KERNEL_DF_temp <- do.call("rbind",STPP_KERNEL_DF_temp)
#
#   # # Save kernel intensity values associated with each simulated TRACK to a separate .csv
#   # # (in case loop crashes or system times out)
#   # write.csv(STPP_KERNEL_DF_temp,
#   #         paste("/Users/trevor.joyce/Grad School/Research/",
#   #               "1_2016_Antarctic Whale Tracking/Data/Output Data/",
#   #              "STPP_KERNEL_DF_",i,"_TRACK_",j,".csv",sep=""))
#
#   # Add resulting kernel intensity values to STPP_KERNEL_DF list
#   STPP_KERNEL_DF[[ STPP_KERNEL_DF_temp$CTCRW_FIT[1] ]][[
STPP_KERNEL_DF_temp$TRACK[1] ]] <- STPP_KERNEL_DF_temp
#
#   # # Clean up
#   # remove(STPP_KERNEL_DF_temp, PROCESSING_TIME_temp); gc()
#
#  }
#
# }
#
#
#
# # Loop through each Mn CTCRW model fit
# for(i in unique(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Mn","PTT"])){
#
#  # Create a list within STPP_KERNEL_DF list to house data.frames
#  # of kernel intensity values associated with each simulated TRACK
#  STPP_KERNEL_DF[[paste("CTCRW_FIT_",i,sep = "")]] <- list()
#
#  # Loop n = 10 simulated tracks drawn from CTCRW model fit
#  for(j in paste("TRACK_",1:10,sep="")){
#
#   # Error handling: for cases where there is no PRE_MIG data to calculate a kernel surface
#   if(nrow(STPP_SIM_TRACKS[STPP_SIM_TRACKS$PTT == i
#                      & STPP_SIM_TRACKS$TRACK == j
#                      & STPP_SIM_TRACKS$MIGRATION == "PRE_MIG",]) == 0){
```

```
#
#     # print error message
#     print("ERROR: no PRE_MIG data to calculate a kernel surface")
#
#     # clear list object created above
#     STPP_KERNEL_DF[[paste("CTCRW_FIT_",i,sep = "")]] <- NULL
#
#     # move to the next track
#     next
#   }
#
#   # Print information on PTT and TRACK to track progress
#   print(paste("Mn", i, j))
#   PROCESSING_TIME_temp <- proc.time()
#
#   # Run STPP_KERNEL_DF_CALC to calculate the  kernel intensity values
#   # associated with each point in simulated TRACK.
#   # mclapply is used to parallelize this calculation across multiple cores
#   STPP_KERNEL_DF_temp <- parallel::mclapply(X = seq(2,
#                                   nrow(STPP_SIM_TRACKS[STPP_SIM_TRACKS$PTT == i
#                                              & STPP_SIM_TRACKS$TRACK == j
#                                              & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG",])),
#                              DATA = STPP_SIM_TRACKS[STPP_SIM_TRACKS$PTT == i
#                                          & STPP_SIM_TRACKS$TRACK == j
#                                          & STPP_SIM_TRACKS$MIGRATION == "PRE_MIG",],
#                              RAST = STPP_COVAR$INT,
#                              PHI = STPP_PHI[STPP_PHI$SPP == "Mn","MAX_VEL"],
#                              FUN = STPP_KERNEL_DF_CALC,
#                              mc.cores = 6)
#
#   # Print information on PROCESSING_TIME associated with specific PTT and TRACK
#   print(proc.time() - PROCESSING_TIME_temp)
#
#   # Concatenate list of data.frames containing the kernel intensity values
#   # associated with each point in simulated TRACK
#   STPP_KERNEL_DF_temp <- do.call("rbind",STPP_KERNEL_DF_temp)
#
#   # # Save kernel intensity values associated with each simulated TRACK to a separate .csv
#   # # (in case loop crashes or system times out)
#   # write.csv(STPP_KERNEL_DF_temp,
#   #        paste("/Users/trevor.joyce/Grad School/Research/",
#   #             "1_2016_Antarctic Whale Tracking/Data/Output Data/",
#   #             "STPP_KERNEL_DF_",i,"_TRACK_",j,".csv",sep=""))
```

```
#
#    # Add resulting kernel intensity values to STPP_KERNEL_DF list
#    STPP_KERNEL_DF[[ STPP_KERNEL_DF_temp$CTCRW_FIT[1] ]][[
STPP_KERNEL_DF_temp$TRACK[1] ]] <- STPP_KERNEL_DF_temp
#
#    # Clean up
#    remove(STPP_KERNEL_DF_temp, PROCESSING_TIME_temp); gc()
#
#  }
#
# }
#
# # Save kernel intensity values to a single overall .rds file
# # (avoids having to re-run calculations)
# saveRDS(object = STPP_KERNEL_DF,
#        file = paste("/Users/trevor.joyce/Grad School/Research/",
#                     "1_2016_Antarctic Whale Tracking/Data/Output Data/",
#                     "STPP_KERNEL_DF.rds",sep=""))

# Reconstitute STPP_KERNEL_DF from saved copy
STPP_KERNEL_DF <- readRDS(file = paste("/Users/trevor.joyce/Grad School/Research/",
                          "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                          "STPP_KERNEL_DF.rds",sep=""))
# remove(STPP_KERNEL_DF)

# Implement a post-hoc restriction of STPP_KERNEL_DF to values greater than 0.01
(comparable to previous model run)
for(i in names(STPP_KERNEL_DF)){
 for(j in names(STPP_KERNEL_DF[[i]])){
  STPP_KERNEL_DF[[i]][[j]] <- STPP_KERNEL_DF[[i]][[j]][STPP_KERNEL_DF[[i]][[j]]$INT>1e-2,]
 }
}
remove(i,j)


#### STPP_KERNEL_CALC #####

# Function to sum intensity kernel values by raster cell numbers
# and then generate a raster of intensity kernel values
# for a simulated TRACK within an individual CTCRW_FIT

STPP_KERNEL_CALC <- function(X,DATA,RAST){

 # Calculate sums of INT column by CELL number within each TRACK
```

```
  GROUPED_DATA <- as.data.frame(dplyr::mutate(dplyr::group_by(DATA[[X]],CELL),
                                KERNEL_INT_SUM = sum(INT)),
                      stringsAsFactors = F)

  # Remove duplicated CELL numbers
  GROUPED_DATA <- GROUPED_DATA[!duplicated(GROUPED_DATA$CELL), ]

  # Set all values in template raster to NA
  RAST[] <- NA

  # Set values corresponding to raster CELL numbers
  # to summed of kernel intensity values (KERNEL_INT_SUM)
  RAST[GROUPED_DATA$CELL] <- GROUPED_DATA$KERNEL_INT_SUM

  # Return a raster object
  return(RAST)

}



##### STPP_KERNEL #####

# Sum intensity kernel values by raster cell numbers
# for each timesstep in STPP_KERNEL_DF at the level
# of each CTCRW_FIT and simulated TRACK.

# Generate a list of rasters that can then be summed accross
# individuals to produce the overall availability surfaces
# for each species and imputation.

# Create a placeholder list that will house kernel intensity availability surfaces
STPP_KERNEL <-list()

# Loop through each CTCRW model fit in STPP_KERNEL_DF
for(i in names(STPP_KERNEL_DF)){

  # Create a list within STPP_KERNEL_DF list to house data.frames
  # of kernel intensity values associated with each simulated TRACK
  STPP_KERNEL[[i]] <- list()

  # Print information on PTT and TRACK to track progress
  print(paste(i))
  PROCESSING_TIME_temp <- proc.time()
```

```
  # Run STPP_KERNEL_CALC function over the objects within each list STPP_KERNEL_DF[[i]].
  # Returns a list of raster objects with values corresponding to the summed kernel intensity
values
  # across the simulate locations within each TRACK within each individual CTCRW_FIT
  STPP_KERNEL[[i]] <-  parallel::mclapply(X = names(STPP_KERNEL_DF[[i]]),
                                  FUN = STPP_KERNEL_CALC,
                                  DATA = STPP_KERNEL_DF[[i]],
                                  RAST = STPP_COVAR$INT,
                                  mc.cores = 4)

  names(STPP_KERNEL[[i]]) <- names(STPP_KERNEL_DF[[i]])

  # Print information on PROCESSING_TIME associated with specific PTT and TRACK
  print(proc.time() - PROCESSING_TIME_temp)

}


# Save a copy for being able to reconstitute directly without re-running algorithm
saveRDS(object = STPP_KERNEL,
    file = paste("/Users/trevor.joyce/Grad School/Research/",
          "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
          "STPP_KERNEL.rds",sep=""))


STPP_KERNEL <- readRDS(file = paste("/Users/trevor.joyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                    "STPP_KERNEL.rds",sep=""))


##### STPP_COVAR$STPP_LN_KERNEL_Mn #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Mn

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Mn
STPP_KERNEL[["Mn"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Mn" that will be populated
```

```
  # by the jth TRACK from each "Mn" individual (i)
  STPP_KERNEL[["Mn"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL
  for(i in
names(STPP_KERNEL)[names(STPP_KERNEL)%in%unique(STPP_SIM_TRACKS[STPP_SIM_TRACK
S$SPP == "Mn","CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL raster for the ith Mn individual
    STPP_KERNEL[["Mn"]][[j]] <- c(STPP_KERNEL[["Mn"]][[j]],
                             STPP_KERNEL[[i]][[j]])

  }

  # Transform each TRACK list within Mn into a RasterStack
  STPP_KERNEL[["Mn"]][[j]] <- raster::stack(STPP_KERNEL[["Mn"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL[["Mn"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL[["Mn"]] <- NULL
remove(i,j)


##### STPP_COVAR$STPP_LN_KERNEL_Mn_2013 #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Mn and YEAR 2013

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Mn
STPP_KERNEL[["Mn"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){
```

```
  # Create a list object for "Mn" that will be populated
  # by the jth TRACK from each "Mn" individual (i)
  STPP_KERNEL[["Mn"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL
  for(i in
names(STPP_KERNEL)[names(STPP_KERNEL)%in%unique(STPP_SIM_TRACKS[STPP_SIM_TRACK
S$SPP == "Mn"
                                              & lubridate::year(STPP_SIM_TRACKS$Time) == 2013
                                              & lubridate::month(STPP_SIM_TRACKS$Time) <= 8
,"CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL raster for the ith Mn individual
    STPP_KERNEL[["Mn"]][[j]] <- c(STPP_KERNEL[["Mn"]][[j]],
                       STPP_KERNEL[[i]][[j]])

  }

  # Transform each TRACK list within Mn into a RasterStack
  STPP_KERNEL[["Mn"]][[j]] <- raster::stack(STPP_KERNEL[["Mn"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_2013_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL[["Mn"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_2013_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_2013_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL[["Mn"]] <- NULL
remove(i,j)


##### STPP_COVAR$STPP_LN_KERNEL_Mn_2016 #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Mn and YEAR 2016

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Mn
STPP_KERNEL[["Mn"]] <- list()
```

```
# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Mn" that will be populated
  # by the jth TRACK from each "Mn" individual (i)
  STPP_KERNEL[["Mn"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL
  for(i in
names(STPP_KERNEL)[names(STPP_KERNEL)%in%unique(STPP_SIM_TRACKS[STPP_SIM_TRACK
S$SPP == "Mn"
                                            & lubridate::year(STPP_SIM_TRACKS$Time) == 2016
                                            & lubridate::month(STPP_SIM_TRACKS$Time) <= 8
,"CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL raster for the ith Mn individual
    STPP_KERNEL[["Mn"]][[j]] <- c(STPP_KERNEL[["Mn"]][[j]],
                    STPP_KERNEL[[i]][[j]])

  }

  # Transform each TRACK list within Mn into a RasterStack
  STPP_KERNEL[["Mn"]][[j]] <- raster::stack(STPP_KERNEL[["Mn"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_2016_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL[["Mn"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_2016_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_Mn_2016_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL[["Mn"]] <- NULL
remove(i,j)



##### STPP_COVAR$STPP_LN_KERNEL_Bb #####

# Create a list that will be populated with TRACK lists of rasters
```

```
# for each imputation within SPP Bb
STPP_KERNEL[["Bb"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Bb" that will be populated
  # by the jth TRACK from each "Bb" individual (i)
  STPP_KERNEL[["Bb"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL
  for(i in
names(STPP_KERNEL)[names(STPP_KERNEL)%in%unique(STPP_SIM_TRACKS[STPP_SIM_TRACK
S$SPP == "Bb","CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL raster for the ith Bb individual
    STPP_KERNEL[["Bb"]][[j]] <- c(STPP_KERNEL[["Bb"]][[j]],
                                  STPP_KERNEL[[i]][[j]])

  }

  # Transform each TRACK list within Bb into a RasterStack
  STPP_KERNEL[["Bb"]][[j]] <- raster::stack(STPP_KERNEL[["Bb"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL[["Bb"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_",j,sep = "")]])] <- NA
}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL[["Bb"]] <- NULL
remove(i,j)



##### STPP_COVAR$STPP_LN_KERNEL_Bb_2013 #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Bb and YEAR 2013
```

```
# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Bb
STPP_KERNEL[["Bb"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Bb" that will be populated
  # by the jth TRACK from each "Bb" individual (i)
  STPP_KERNEL[["Bb"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL
  for(i in
names(STPP_KERNEL)[names(STPP_KERNEL)%in%unique(STPP_SIM_TRACKS[STPP_SIM_TRACK
S$SPP == "Bb"
                                                  & lubridate::year(STPP_SIM_TRACKS$Time) == 2013
                                                  & lubridate::month(STPP_SIM_TRACKS$Time) <= 8
,"CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL raster for the ith Bb individual
    STPP_KERNEL[["Bb"]][[j]] <- c(STPP_KERNEL[["Bb"]][[j]],
                    STPP_KERNEL[[i]][[j]])

  }

  # Transform each TRACK list within Bb into a RasterStack
  STPP_KERNEL[["Bb"]][[j]] <- raster::stack(STPP_KERNEL[["Bb"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_2013_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL[["Bb"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_2013_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_2013_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL[["Bb"]] <- NULL
remove(i,j)


##### STPP_COVAR$STPP_LN_KERNEL_Bb_2016 #####
```

```
## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Bb and YEAR 2016

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Bb
STPP_KERNEL[["Bb"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Bb" that will be populated
  # by the jth TRACK from each "Bb" individual (i)
  STPP_KERNEL[["Bb"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL
  for(i in
names(STPP_KERNEL)[names(STPP_KERNEL)%in%unique(STPP_SIM_TRACKS[STPP_SIM_TRACK
S$SPP == "Bb"
                                          & lubridate::year(STPP_SIM_TRACKS$Time) == 2016
                                          & lubridate::month(STPP_SIM_TRACKS$Time) <= 8
,"CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL raster for the ith Bb individual
    STPP_KERNEL[["Bb"]][[j]] <- c(STPP_KERNEL[["Bb"]][[j]],
                          STPP_KERNEL[[i]][[j]])

  }

  # Transform each TRACK list within Bb into a RasterStack
  STPP_KERNEL[["Bb"]][[j]] <- raster::stack(STPP_KERNEL[["Bb"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_2016_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL[["Bb"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_2016_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_Bb_2016_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL[["Bb"]] <- NULL
```

```
remove(i,j)


#### STPP_KERNEL_WAP_CALC #####

# Function to sum intensity kernel values by raster cell numbers
# and then generate a raster of intensity kernel values
# for a simulated TRACK within an individual CTCRW_FIT

STPP_KERNEL_WAP_CALC <- function(X,DATA,SIM_TRACK,RAST){


  WEDDELL_SEA <- data.frame(LONG = c(-57.3,-56.4,-45.5,-14.0,-06.0,-28.0,-58.0,-65.0,-64.0,-
57.3),
                  LAT =  c(-63.25,-63.0,-61.1,-63.5,-71.0,-78.0,-77.0,-73.0,-66.0,-63.25))

  WAP_DATA <- DATA[[X]]
  WAP_DATA$REGION <- sp::point.in.polygon(point.x = WAP_DATA$LONG_SIM,
                           point.y = WAP_DATA$LAT_SIM,
                           pol.x = WEDDELL_SEA$LONG,
                           pol.y = WEDDELL_SEA$LAT)
  WAP_DATA <- WAP_DATA[WAP_DATA$REGION < 1,]

  # Calculate sums of INT column by CELL number within each TRACK
  GROUPED_DATA <- as.data.frame(dplyr::mutate(dplyr::group_by(WAP_DATA,CELL),
                              KERNEL_INT_SUM = sum(INT)),
                      stringsAsFactors = F)

  # Remove duplicated CELL numbers
  GROUPED_DATA <- GROUPED_DATA[!duplicated(GROUPED_DATA$CELL), ]

  # Set all values in template raster to NA
  RAST[] <- NA

  # Set values corresponding to raster CELL numbers
  # to summed of kernel intensity values (KERNEL_INT_SUM)
  RAST[GROUPED_DATA$CELL] <- GROUPED_DATA$KERNEL_INT_SUM

  # Return a raster object
  return(RAST)

}
```

```
# #### STPP_KERNEL_WAP #####

# Sum intensity kernel values by raster cell numbers
# for each timesstep in STPP_KERNEL_DF at the level
# of each CTCRW_FIT and simulated TRACK.

# Generate a list of rasters that can then be summed accross
# individuals to produce the overall availability surfaces
# for each species and imputation.

# Create a placeholder list that will house kernel intensity availability surfaces
STPP_KERNEL_WAP <-list()

# Loop through each CTCRW model fit in STPP_KERNEL_WAP_DF
for(i in names(STPP_KERNEL_DF)){

  # Create a list within STPP_KERNEL_WAP_DF list to house data.frames
  # of kernel intensity values associated with each simulated TRACK
  STPP_KERNEL_WAP[[i]] <- list()

  # Print information on PTT and TRACK to track progress
  print(paste(i))
  PROCESSING_TIME_temp <- proc.time()

  # Run STPP_KERNEL_WAP_CALC function over the objects within each list
STPP_KERNEL_WAP_DF[[i]].
  # Returns a list of raster objects with values corresponding to the summed kernel intensity
values
  # across the simulate locations within each TRACK within each individual CTCRW_FIT
  STPP_KERNEL_WAP[[i]] <-  parallel::mclapply(X = names(STPP_KERNEL_DF[[i]]),
                                   FUN = STPP_KERNEL_WAP_CALC,
                                   DATA = STPP_KERNEL_DF[[i]],
                                   RAST = STPP_COVAR$INT,
                                   mc.cores = 4)

  names(STPP_KERNEL_WAP[[i]]) <- names(STPP_KERNEL_DF[[i]])

  # Print information on PROCESSING_TIME associated with specific PTT and TRACK
  print(proc.time() - PROCESSING_TIME_temp)

}

# Clean-up
```

```
remove(i)



# Save a copy for being able to reconstitute directly without re-running algorithm
saveRDS(object = STPP_KERNEL_WAP,
    file = paste("/Users/trevor.joyce/Grad School/Research/",
          "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
          "STPP_KERNEL_WAP.rds",sep=""))


STPP_KERNEL_WAP <- readRDS(file = paste("/Users/trevor.joyce/Grad School/Research/",
                  "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                  "STPP_KERNEL_WAP.rds",sep=""))

##### STPP_COVAR$STPP_LN_KERNEL_WAP_Mn #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Mn

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Mn
STPP_KERNEL_WAP[["Mn"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Mn" that will be populated
  # by the jth TRACK from each "Mn" individual (i)
  STPP_KERNEL_WAP[["Mn"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL_WAP
  for(i in
names(STPP_KERNEL_WAP)[names(STPP_KERNEL_WAP)%in%unique(STPP_SIM_TRACKS[STPP_
SIM_TRACKS$SPP == "Mn","CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL_WAP raster for the ith Mn individual
    STPP_KERNEL_WAP[["Mn"]][[j]] <- c(STPP_KERNEL_WAP[["Mn"]][[j]],
                    STPP_KERNEL_WAP[[i]][[j]])

  }

  # Transform each TRACK list within Mn into a RasterStack
  STPP_KERNEL_WAP[["Mn"]][[j]] <- raster::stack(STPP_KERNEL_WAP[["Mn"]][[j]])
```

```
  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL_WAP[["Mn"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL_WAP[["Mn"]] <- NULL
remove(i,j)


##### STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013 #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Mn and YEAR 2013

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Mn
STPP_KERNEL_WAP[["Mn"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Mn" that will be populated
  # by the jth TRACK from each "Mn" individual (i)
  STPP_KERNEL_WAP[["Mn"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL_WAP
  for(i in
names(STPP_KERNEL_WAP)[names(STPP_KERNEL_WAP)%in%unique(STPP_SIM_TRACKS[STPP_
SIM_TRACKS$SPP == "Mn"
                                          & lubridate::year(STPP_SIM_TRACKS$Time) ==
2013
                                          & lubridate::month(STPP_SIM_TRACKS$Time) <=
8 ,"CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL_WAP raster for the ith Mn individual
    STPP_KERNEL_WAP[["Mn"]][[j]] <- c(STPP_KERNEL_WAP[["Mn"]][[j]],
                    STPP_KERNEL_WAP[[i]][[j]])
```

```
  }

  # Transform each TRACK list within Mn into a RasterStack
  STPP_KERNEL_WAP[["Mn"]][[j]] <- raster::stack(STPP_KERNEL_WAP[["Mn"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_2013_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL_WAP[["Mn"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_2013_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_2013_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL_WAP[["Mn"]] <- NULL
remove(i,j)

##### STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016 #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Mn and YEAR 2016

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Mn
STPP_KERNEL_WAP[["Mn"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Mn" that will be populated
  # by the jth TRACK from each "Mn" individual (i)
  STPP_KERNEL_WAP[["Mn"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL_WAP
  for(i in
names(STPP_KERNEL_WAP)[names(STPP_KERNEL_WAP)%in%unique(STPP_SIM_TRACKS[STPP_
SIM_TRACKS$SPP == "Mn"
                                              & lubridate::year(STPP_SIM_TRACKS$Time) ==
2016
                                              & lubridate::month(STPP_SIM_TRACKS$Time) <=
8 ,"CTCRW_FIT"])]){
```

```
    # populate the jth TRACK with the STPP_KERNEL_WAP raster for the ith Mn individual
    STPP_KERNEL_WAP[["Mn"]][[j]] <- c(STPP_KERNEL_WAP[["Mn"]][[j]],
                                  STPP_KERNEL_WAP[[i]][[j]])

  }

  # Transform each TRACK list within Mn into a RasterStack
  STPP_KERNEL_WAP[["Mn"]][[j]] <- raster::stack(STPP_KERNEL_WAP[["Mn"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_2016_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL_WAP[["Mn"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_2016_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Mn_2016_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL_WAP[["Mn"]] <- NULL
remove(i,j)


##### STPP_COVAR$STPP_LN_KERNEL_WAP_Bb #####

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Bb
STPP_KERNEL_WAP[["Bb"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Bb" that will be populated
  # by the jth TRACK from each "Bb" individual (i)
  STPP_KERNEL_WAP[["Bb"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL_WAP
  for(i in
names(STPP_KERNEL_WAP)[names(STPP_KERNEL_WAP)%in%unique(STPP_SIM_TRACKS[STPP_
SIM_TRACKS$SPP == "Bb","CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL_WAP raster for the ith Bb individual
```

```
    STPP_KERNEL_WAP[["Bb"]][[j]] <- c(STPP_KERNEL_WAP[["Bb"]][[j]],
                        STPP_KERNEL_WAP[[i]][[j]])

  }

  # Transform each TRACK list within Bb into a RasterStack
  STPP_KERNEL_WAP[["Bb"]][[j]] <- raster::stack(STPP_KERNEL_WAP[["Bb"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL_WAP[["Bb"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_",j,sep = "")]])] <- NA


}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL_WAP[["Bb"]] <- NULL
remove(i,j)



##### STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013 #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Bb and YEAR 2013

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Bb
STPP_KERNEL_WAP[["Bb"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){

  # Create a list object for "Bb" that will be populated
  # by the jth TRACK from each "Bb" individual (i)
  STPP_KERNEL_WAP[["Bb"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL_WAP
```

```
 for(i in
names(STPP_KERNEL_WAP)[names(STPP_KERNEL_WAP)%in%unique(STPP_SIM_TRACKS[STPP_
SIM_TRACKS$SPP == "Bb"
                                                  & lubridate::year(STPP_SIM_TRACKS$Time) ==
2013
                                                  & lubridate::month(STPP_SIM_TRACKS$Time) <=
8 ,"CTCRW_FIT"])]){

   # populate the jth TRACK with the STPP_KERNEL_WAP raster for the ith Bb individual
   STPP_KERNEL_WAP[["Bb"]][[j]] <- c(STPP_KERNEL_WAP[["Bb"]][[j]],
                          STPP_KERNEL_WAP[[i]][[j]])

 }

 # Transform each TRACK list within Bb into a RasterStack
 STPP_KERNEL_WAP[["Bb"]][[j]] <- raster::stack(STPP_KERNEL_WAP[["Bb"]][[j]])

 # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
 STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_2013_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL_WAP[["Bb"]][[j]], sum, na.rm = T))


 # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
 STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_2013_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_2013_",j,sep = "")]])] <- NA

}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL_WAP[["Bb"]] <- NULL
remove(i,j)

##### STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016 #####

## Reorganize raster layers to calculate an STPP_COVAR layer
## for each imputation (TRACK) across all inividuals within SPP Bb and YEAR 2016

# Create a list that will be populated with TRACK lists of rasters
# for each imputation within SPP Bb
STPP_KERNEL_WAP[["Bb"]] <- list()

# Loop through each simulated TRACK
for(j in paste("TRACK_",1:10,sep = "")){
```

```
  # Create a list object for "Bb" that will be populated
  # by the jth TRACK from each "Bb" individual (i)
  STPP_KERNEL_WAP[["Bb"]][[j]] <- list()

  # Loop through each CTCRW model fit in STPP_KERNEL_WAP
  for(i in
names(STPP_KERNEL_WAP)[names(STPP_KERNEL_WAP)%in%unique(STPP_SIM_TRACKS[STPP_
SIM_TRACKS$SPP == "Bb"
                                                      & lubridate::year(STPP_SIM_TRACKS$Time) ==
2016
                                                      & lubridate::month(STPP_SIM_TRACKS$Time) <=
8 ,"CTCRW_FIT"])]){

    # populate the jth TRACK with the STPP_KERNEL_WAP raster for the ith Bb individual
    STPP_KERNEL_WAP[["Bb"]][[j]] <- c(STPP_KERNEL_WAP[["Bb"]][[j]],
                                 STPP_KERNEL_WAP[[i]][[j]])

  }

  # Transform each TRACK list within Bb into a RasterStack
  STPP_KERNEL_WAP[["Bb"]][[j]] <- raster::stack(STPP_KERNEL_WAP[["Bb"]][[j]])

  # Use calc to sum these raster layers to achieve a single STPP_COVAR layer
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_2016_",j,sep = "")]] <-
log(raster::calc(STPP_KERNEL_WAP[["Bb"]][[j]], sum, na.rm = T))

  # Eliminate any -Inf (i.e., log(0)) values (messes up calculation of SUPPORT)
  STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_2016_",j,sep =
"")]][is.infinite(STPP_COVAR[[paste("STPP_LN_KERNEL_WAP_Bb_2016_",j,sep = "")]])] <- NA
}

# Clean-up: free the disk space used to calculate STPP_COVAR layer
STPP_KERNEL_WAP[["Bb"]] <- NULL
remove(i,j)


##### STPP_COVAR$SUPPORT_Mn #####

STPP_COVAR$SUPPORT_Mn <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_1)),
               raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_2)),
               raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_3)),
               raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_4)),
               raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_5)),
```

```
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_6)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_7)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_8)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_9)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_TRACK_10)),na.rm =
T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_Mn[STPP_COVAR$SUPPORT_Mn > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_Mn <- STPP_COVAR$SUPPORT_Mn * STPP_COVAR$OCEAN

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_Mn[STPP_COVAR$SUPPORT_Mn==0] <- NA


##### STPP_COVAR$SUPPORT_Mn_2013 #####

STPP_COVAR$SUPPORT_Mn_2013 <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_1)),
                      raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_2)),
                      raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_3)),
                      raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_4)),
                      raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_5)),
                      raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_6)),
                      raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_7)),
                      raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_8)),
                      raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2013_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_Mn_2013[STPP_COVAR$SUPPORT_Mn_2013 > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_Mn_2013 <- STPP_COVAR$SUPPORT_Mn_2013 *
STPP_COVAR$OCEAN

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_Mn_2013[STPP_COVAR$SUPPORT_Mn_2013==0] <- NA

##### STPP_COVAR$SUPPORT_Mn_2016 #####
```

```
STPP_COVAR$SUPPORT_Mn_2016 <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_1)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_2)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_3)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_4)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_5)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_6)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_7)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_8)),
                    raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Mn_2016_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_Mn_2016[STPP_COVAR$SUPPORT_Mn_2016 > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_Mn_2016 <- STPP_COVAR$SUPPORT_Mn_2016 *
STPP_COVAR$OCEAN

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_Mn_2016[STPP_COVAR$SUPPORT_Mn_2016==0] <- NA


##### STPP_COVAR$SUPPORT_Bb #####

STPP_COVAR$SUPPORT_Bb <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_1)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_2)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_3)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_4)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_5)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_6)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_7)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_8)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_9)),
                 raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_TRACK_10)),na.rm =
T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_Bb[STPP_COVAR$SUPPORT_Bb > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_Bb <- STPP_COVAR$SUPPORT_Bb * STPP_COVAR$OCEAN
```

```
# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_Bb[STPP_COVAR$SUPPORT_Bb==0] <- NA


##### STPP_COVAR$SUPPORT_Bb_2013 #####

STPP_COVAR$SUPPORT_Bb_2013 <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_1)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_2)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_3)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_4)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_5)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_6)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_7)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_8)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2013_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_Bb_2013[STPP_COVAR$SUPPORT_Bb_2013 > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_Bb_2013 <- STPP_COVAR$SUPPORT_Bb_2013 * STPP_COVAR$OCEAN

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_Bb_2013[STPP_COVAR$SUPPORT_Bb_2013==0] <- NA


##### STPP_COVAR$SUPPORT_Bb_2016 #####

STPP_COVAR$SUPPORT_Bb_2016 <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_1)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_2)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_3)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_4)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_5)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_6)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_7)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_8)),
                  raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_Bb_2016_TRACK_10)),na.rm = T)
```

```
# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_Bb_2016[STPP_COVAR$SUPPORT_Bb_2016 > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_Bb_2016 <- STPP_COVAR$SUPPORT_Bb_2016 * STPP_COVAR$OCEAN

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_Bb_2016[STPP_COVAR$SUPPORT_Bb_2016==0] <- NA



##### STPP_COVAR$SUPPORT_WAP_Mn #####

STPP_COVAR$SUPPORT_WAP_Mn <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_1)),
                        raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_2)),
                        raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_3)),
                        raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_4)),
                        raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_5)),
                        raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_6)),
                        raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_7)),
                        raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_8)),
                        raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_WAP_Mn[STPP_COVAR$SUPPORT_WAP_Mn > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_WAP_Mn <- STPP_COVAR$SUPPORT_WAP_Mn *
STPP_COVAR$OCEAN * STPP_COVAR$WAP

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_WAP_Mn[STPP_COVAR$SUPPORT_WAP_Mn==0] <- NA

##### STPP_COVAR$SUPPORT_WAP_Mn_2013 #####

STPP_COVAR$SUPPORT_WAP_Mn_2013 <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_1)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_2)),
```

```
raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_3)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_4)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_5)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_6)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_7)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_8)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2013_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_WAP_Mn_2013[STPP_COVAR$SUPPORT_WAP_Mn_2013 > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_WAP_Mn_2013 <- STPP_COVAR$SUPPORT_WAP_Mn_2013 *
STPP_COVAR$OCEAN * STPP_COVAR$WAP

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_WAP_Mn_2013[STPP_COVAR$SUPPORT_WAP_Mn_2013==0] <- NA


##### STPP_COVAR$SUPPORT_WAP_Mn_2016 #####

STPP_COVAR$SUPPORT_WAP_Mn_2016 <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_1)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_2)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_3)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_4)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_5)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_6)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_7)),
```

```
raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_8)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Mn_2016_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_WAP_Mn_2016[STPP_COVAR$SUPPORT_WAP_Mn_2016 > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_WAP_Mn_2016 <- STPP_COVAR$SUPPORT_WAP_Mn_2016 *
STPP_COVAR$OCEAN * STPP_COVAR$WAP

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_WAP_Mn_2016[STPP_COVAR$SUPPORT_WAP_Mn_2016==0] <- NA

##### STPP_COVAR$SUPPORT_WAP_Bb #####

STPP_COVAR$SUPPORT_WAP_Bb <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_1)),
                raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_2)),
                raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_3)),
                raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_4)),
                raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_5)),
                raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_6)),
                raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_7)),
                raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_8)),
                raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_WAP_Bb[STPP_COVAR$SUPPORT_WAP_Bb > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_WAP_Bb <- STPP_COVAR$SUPPORT_WAP_Bb * STPP_COVAR$OCEAN
* STPP_COVAR$WAP

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_WAP_Bb[STPP_COVAR$SUPPORT_WAP_Bb==0] <- NA


##### STPP_COVAR$SUPPORT_WAP_Bb_2013 #####
```

```
STPP_COVAR$SUPPORT_WAP_Bb_2013 <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_1)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_2)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_3)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_4)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_5)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_6)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_7)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_8)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2013_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_WAP_Bb_2013[STPP_COVAR$SUPPORT_WAP_Bb_2013 > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_WAP_Bb_2013 <- STPP_COVAR$SUPPORT_WAP_Bb_2013 *
STPP_COVAR$OCEAN * STPP_COVAR$WAP

# exclude areas off the continental shelf (different behavioral pattern, i.e. migration)
STPP_COVAR$SUPPORT_WAP_Bb_2013[STPP_COVAR$BATHY <= -2000] <- NA

# exclude areas S of Marguerite Bay (more polar climate, year round offshore sea ice)
STPP_COVAR$SUPPORT_WAP_Bb_2013[STPP_COVAR$LAT <= -69] <- NA

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_WAP_Bb_2013[STPP_COVAR$SUPPORT_WAP_Bb_2013==0] <- NA


##### STPP_COVAR$SUPPORT_WAP_Bb_2016 #####

STPP_COVAR$SUPPORT_WAP_Bb_2016 <-
sum(raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_1)),
```

```
raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_2)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_3)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_4)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_5)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_6)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_7)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_8)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_9)),

raster::Which(!is.na(STPP_COVAR$STPP_LN_KERNEL_WAP_Bb_2016_TRACK_10)),na.rm = T)

# set all kernel intensity surface values to 1
STPP_COVAR$SUPPORT_WAP_Bb_2016[STPP_COVAR$SUPPORT_WAP_Bb_2016 > 0] <- 1

# exclude raster grid cells where kernel intensity surface intersects land
STPP_COVAR$SUPPORT_WAP_Bb_2016 <- STPP_COVAR$SUPPORT_WAP_Bb_2016 *
STPP_COVAR$OCEAN * STPP_COVAR$WAP

# exclude all areas where kernel intensity surface = 0
STPP_COVAR$SUPPORT_WAP_Bb_2016[STPP_COVAR$SUPPORT_WAP_Bb_2016==0] <- NA


######## STPP_COVAR$STPP_LN_KERNEL_ORTHO_Mn ######

## Calculate residuals of kernel regressed on covariates (ln_k_ortho)

# Convert STPP_COVAR into a data.frame for the calculation orthogonal kernel intensity surface
for each TRACK
STPP_COVAR_DF_temp <- as.data.frame(as(STPP_COVAR,"SpatialPointsDataFrame"))

# Convert COAST_CLASS ordinal variables from numeric to factor
STPP_COVAR_DF_temp$COAST_CLASS <- as.factor(STPP_COVAR_DF_temp$COAST_CLASS)
STPP_COVAR_DF_temp$COAST_CLASS_GEN <-
as.factor(STPP_COVAR_DF_temp$COAST_CLASS_GEN)
```

```
for(i in paste("TRACK_",1:10,sep="")){

  # Take a copy of STPP_COVAR_DF_temp that will be used to calculate orthogonal kernel
intensity surface
  # for each specific imputation (TRACK)
  STPP_COVAR_DF_TRACK_temp <- STPP_COVAR_DF_temp

  # Create a variable STPP_LN_KERNEL_Mn representing the kernel intensity surface for
TRACK[i] (a single imputation from CTCRW model)
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn <-
STPP_COVAR_DF_TRACK_temp[,paste("STPP_LN_KERNEL_Mn_",i,sep = "")]

  # Create a variable that filters for any cells where the response variable or any covariate is NA
  STPP_COVAR_DF_TRACK_temp$NON_NA <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY +
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2016 +
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2016_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2016 + STPP_COVAR_DF_TRACK_temp$SUPPORT_Mn

  # Create a variable that filters for any cells where the response variable or any covariate is NA
(WAP region only)
  STPP_COVAR_DF_TRACK_temp$NON_NA_WAP <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY +
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2016 +
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2016_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2016 +
STPP_COVAR_DF_TRACK_temp$SUPPORT_WAP_Mn

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Mn <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),"STPP_LN_KE
RNEL_ORTHO_Mn"] <- residuals(lm(STPP_LN_KERNEL_Mn ~ DIST_SHELF + BATHY +
ENCLOSURE_MED +
                                                  SLOPE + ICE_CONC_2016 +
DIST_ICE_2016_15 + CHL_2016,
```

```
                                              data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),]))

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Mn <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),"STPP_L
N_KERNEL_ORTHO_WAP_Mn"] <- residuals(lm(STPP_LN_KERNEL_Mn ~ DIST_SHELF + BATHY +
ENCLOSURE_MED +
                                                  SLOPE + ICE_CONC_2016 +
DIST_ICE_2016_15 + CHL_2016,
                                                 data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),]))


  # Save the resulting orthogonal kernel intensity surface to a variable in STPP_COVAR
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Mn_",i,sep = "")]] <- STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Mn_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Mn

  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_",i,sep = "")]] <- STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Mn

}

# Clean-up
# remove(STPP_COVAR_DF_temp)
remove(STPP_COVAR_DF_TRACK_temp,i)



######## STPP_COVAR$STPP_LN_KERNEL_ORTHO_Mn_2013 ######

## Calculate residuals of kernel regressed on covariates (ln_k_ortho)

# # Convert STPP_COVAR into a data.frame for the calculation orthogonal kernel intensity
surface for each TRACK
# STPP_COVAR_DF_temp <- as.data.frame(as(STPP_COVAR,"SpatialPointsDataFrame"))

for(i in paste("TRACK_",1:10,sep="")){
```

```
  # Take a copy of STPP_COVAR_DF_temp that will be used to calculate orthogonal kernel
intensity surface
  # for each specific imputation (TRACK)
  STPP_COVAR_DF_TRACK_temp <- STPP_COVAR_DF_temp

  # Create a variable STPP_LN_KERNEL_Mn_2013 representing the kernel intensity surface for
TRACK[i] (a single imputation from CTCRW model)
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn_2013 <-
STPP_COVAR_DF_TRACK_temp[,paste("STPP_LN_KERNEL_Mn_2013_",i,sep = "")]

  # Create a variable that filters for any cells where the response variable or any covariate is NA
  STPP_COVAR_DF_TRACK_temp$NON_NA <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn_2013 +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY +
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2013 +
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2013_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2013 +
STPP_COVAR_DF_TRACK_temp$SUPPORT_Mn_2013

  # Create a variable that filters for any cells where the response variable or any covariate is NA
(WAP region only)
  STPP_COVAR_DF_TRACK_temp$NON_NA_WAP <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn_2013 +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY +
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2013 +
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2013_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2013 +
STPP_COVAR_DF_TRACK_temp$SUPPORT_WAP_Mn_2013

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Mn_2013 <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),"STPP_LN_KE
RNEL_ORTHO_Mn_2013"] <- residuals(lm(STPP_LN_KERNEL_Mn_2013 ~ DIST_SHELF + BATHY +
ENCLOSURE_MED +
                                                                      SLOPE + ICE_CONC_2013 +
DIST_ICE_2013_15 + CHL_2013,
```

```
                                                            data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),]))

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Mn_2013 <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),"STPP_L
N_KERNEL_ORTHO_WAP_Mn_2013"] <- residuals(lm(STPP_LN_KERNEL_Mn_2013 ~
DIST_SHELF + BATHY + ENCLOSURE_MED +
                                                                  SLOPE + ICE_CONC_2013 +
DIST_ICE_2013_15 + CHL_2013,
                                                            data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),]))


  # Save the resulting orthogonal kernel intensity surface to a variable in STPP_COVAR
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Mn_2013_",i,sep = "")]] <- STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Mn_2013_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Mn_2013

  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_2013_",i,sep = "")]] <-
STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_2013_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Mn_2013

}

# Clean-up
# remove(STPP_COVAR_DF_temp)
remove(STPP_COVAR_DF_TRACK_temp,i)



######## STPP_COVAR$STPP_LN_KERNEL_ORTHO_Mn_2016 ######

## Calculate residuals of kernel regressed on covariates (ln_k_ortho)

# # Convert STPP_COVAR into a data.frame for the calculation orthogonal kernel intensity
surface for each TRACK
# STPP_COVAR_DF_temp <- as.data.frame(as(STPP_COVAR,"SpatialPointsDataFrame"))

for(i in paste("TRACK_",1:10,sep="")){
```

```
  # Take a copy of STPP_COVAR_DF_temp that will be used to calculate orthogonal kernel intensity surface
  # for each specific imputation (TRACK)
  STPP_COVAR_DF_TRACK_temp <- STPP_COVAR_DF_temp

  # Create a variable STPP_LN_KERNEL_Mn_2016 representing the kernel intensity surface for TRACK[i] (a single imputation from CTCRW model)
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn_2016 <- 
STPP_COVAR_DF_TRACK_temp[,paste("STPP_LN_KERNEL_Mn_2016_",i,sep = "")]

  # Create a variable that filters for any cells where the response variable or any covariate is NA
  STPP_COVAR_DF_TRACK_temp$NON_NA <- 
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn_2016 +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY + 
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2016 + 
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2016_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2016 + 
STPP_COVAR_DF_TRACK_temp$SUPPORT_Mn_2016

  # Create a variable that filters for any cells where the response variable or any covariate is NA (WAP region only)
  STPP_COVAR_DF_TRACK_temp$NON_NA_WAP <- 
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Mn_2016 +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY + 
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2016 + 
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2016_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2016 + 
STPP_COVAR_DF_TRACK_temp$SUPPORT_WAP_Mn_2016

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Mn_2016 <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates
  
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),"STPP_LN_KERNEL_ORTHO_Mn_2016"] <- residuals(lm(STPP_LN_KERNEL_Mn_2016 ~ DIST_SHELF + BATHY + ENCLOSURE_MED +
                                                                                SLOPE + ICE_CONC_2016 + 
DIST_ICE_2016_15 + CHL_2016,
```

```
                                                                                                            data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),]))

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Mn_2016 <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),"STPP_L
N_KERNEL_ORTHO_WAP_Mn_2016"] <- residuals(lm(STPP_LN_KERNEL_Mn_2016 ~
DIST_SHELF + BATHY + ENCLOSURE_MED +
                                                                                                                    SLOPE + ICE_CONC_2016 +
DIST_ICE_2016_15 + CHL_2016,
                                                                                                              data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),]))


  # Save the resulting orthogonal kernel intensity surface to a variable in STPP_COVAR
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Mn_2016_",i,sep = "")]] <- STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Mn_2016_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Mn_2016

  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_2016_",i,sep = "")]] <-
STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_2016_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Mn_2016

}

# Clean-up
# remove(STPP_COVAR_DF_temp)
remove(STPP_COVAR_DF_TRACK_temp,i)




######## STPP_COVAR$STPP_LN_KERNEL_ORTHO_Bb ######

## Calculate residuals of kernel regressed on covariates (ln_k_ortho)

# # Convert STPP_COVAR into a data.frame for the calculation orthogonal kernel intensity
surface for each TRACK
# STPP_COVAR_DF_temp <- as.data.frame(as(STPP_COVAR,"SpatialPointsDataFrame"))
```

```
for(i in paste("TRACK_",1:10,sep="")){

  # Take a copy of STPP_COVAR_DF_temp that will be used to calculate orthogonal kernel
intensity surface
  # for each specific imputation (TRACK)
  STPP_COVAR_DF_TRACK_temp <- STPP_COVAR_DF_temp

  # Create a variable STPP_LN_KERNEL_Bb representing the kernel intensity surface for TRACK[i]
(a single imputation from CTCRW model)
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb <-
STPP_COVAR_DF_TRACK_temp[,paste("STPP_LN_KERNEL_Bb_",i,sep = "")]

  # Create a variable that filters for any cells where the response variable or any covariate is NA
  STPP_COVAR_DF_TRACK_temp$NON_NA <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY +
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2016 +
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2016_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2016 + STPP_COVAR_DF_TRACK_temp$SUPPORT_Bb

  # Create a variable that filters for any cells where the response variable or any covariate is NA
(WAP region only)
  STPP_COVAR_DF_TRACK_temp$NON_NA_WAP <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY +
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2016 +
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2016_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2016 +
STPP_COVAR_DF_TRACK_temp$SUPPORT_WAP_Bb

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Bb <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),"STPP_LN_KE
RNEL_ORTHO_Bb"] <- residuals(lm(STPP_LN_KERNEL_Bb ~ DIST_SHELF + BATHY +
ENCLOSURE_MED +
```

```
                                                            SLOPE + ICE_CONC_2016 +
DIST_ICE_2016_15 + CHL_2016,
                                                           data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),]))

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Bb <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),"STPP_L
N_KERNEL_ORTHO_WAP_Bb"] <- residuals(lm(STPP_LN_KERNEL_Bb ~ DIST_SHELF + BATHY +
ENCLOSURE_MED +
                                                                SLOPE + ICE_CONC_2016 +
DIST_ICE_2016_15 + CHL_2016,
                                                               data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),]))


  # Save the resulting orthogonal kernel intensity surface to a variable in STPP_COVAR
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Bb_",i,sep = "")]] <- STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Bb_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Bb

  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_",i,sep = "")]] <- STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Bb

}

# Clean-up
# remove(STPP_COVAR_DF_temp)
remove(STPP_COVAR_DF_TRACK_temp,i)


######## STPP_COVAR$STPP_LN_KERNEL_ORTHO_Bb_2013 ######

## Calculate residuals of kernel regressed on covariates (ln_k_ortho)

# # Convert STPP_COVAR into a data.frame for the calculation orthogonal kernel intensity
surface for each TRACK
# STPP_COVAR_DF_temp <- as.data.frame(as(STPP_COVAR,"SpatialPointsDataFrame"))

for(i in paste("TRACK_",1:10,sep="")){
```

```
  # Take a copy of STPP_COVAR_DF_temp that will be used to calculate orthogonal kernel intensity surface
  # for each specific imputation (TRACK)
  STPP_COVAR_DF_TRACK_temp <- STPP_COVAR_DF_temp

  # Create a variable STPP_LN_KERNEL_Bb_2013 representing the kernel intensity surface for TRACK[i] (a single imputation from CTCRW model)
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb_2013 <- 
STPP_COVAR_DF_TRACK_temp[,paste("STPP_LN_KERNEL_Bb_2013_",i,sep = "")]

  # Create a variable that filters for any cells where the response variable or any covariate is NA
  STPP_COVAR_DF_TRACK_temp$NON_NA <- 
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb_2013 +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY + 
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2013 + 
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2013_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2013 + 
STPP_COVAR_DF_TRACK_temp$SUPPORT_Bb_2013

  # Create a variable that filters for any cells where the response variable or any covariate is NA (WAP region only)
  STPP_COVAR_DF_TRACK_temp$NON_NA_WAP <- 
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb_2013 +
    STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY + 
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
    STPP_COVAR_DF_TRACK_temp$SLOPE +
    STPP_COVAR_DF_TRACK_temp$ICE_CONC_2013 + 
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2013_15 +
    STPP_COVAR_DF_TRACK_temp$CHL_2013 + 
STPP_COVAR_DF_TRACK_temp$SUPPORT_WAP_Bb_2013

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Bb_2013 <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates
  
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),"STPP_LN_KERNEL_ORTHO_Bb_2013"] <- residuals(lm(STPP_LN_KERNEL_Bb_2013 ~ DIST_SHELF + BATHY + ENCLOSURE_MED +
                                                                        SLOPE + ICE_CONC_2013 + 
DIST_ICE_2013_15 + CHL_2013,
```

```
                                                                data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),]))

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Bb_2013 <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),"STPP_L
N_KERNEL_ORTHO_WAP_Bb_2013"] <- residuals(lm(STPP_LN_KERNEL_Bb_2013 ~ DIST_SHELF
+ BATHY + ENCLOSURE_MED +
                                                                      SLOPE + ICE_CONC_2013 +
DIST_ICE_2013_15 + CHL_2013,
                                                                   data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),]))


  # Save the resulting orthogonal kernel intensity surface to a variable in STPP_COVAR
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Bb_2013_",i,sep = "")]] <- STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Bb_2013_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Bb_2013

  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_2013_",i,sep = "")]] <-
STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_2013_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Bb_2013

}

# Clean-up
# remove(STPP_COVAR_DF_temp)
remove(STPP_COVAR_DF_TRACK_temp,i)



######## STPP_COVAR$STPP_LN_KERNEL_ORTHO_Bb_2016 ######

## Calculate residuals of kernel regressed on covariates (ln_k_ortho)

# # Convert STPP_COVAR into a data.frame for the calculation orthogonal kernel intensity
surface for each TRACK
# STPP_COVAR_DF_temp <- as.data.frame(as(STPP_COVAR,"SpatialPointsDataFrame"))

for(i in paste("TRACK_",1:10,sep="")){
```

```
 # Take a copy of STPP_COVAR_DF_temp that will be used to calculate orthogonal kernel
intensity surface
 # for each specific imputation (TRACK)
 STPP_COVAR_DF_TRACK_temp <- STPP_COVAR_DF_temp

 # Create a variable STPP_LN_KERNEL_Bb_2016 representing the kernel intensity surface for
TRACK[i] (a single imputation from CTCRW model)
 STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb_2016 <-
STPP_COVAR_DF_TRACK_temp[,paste("STPP_LN_KERNEL_Bb_2016_",i,sep = "")]

 # Create a variable that filters for any cells where the response variable or any covariate is NA
 STPP_COVAR_DF_TRACK_temp$NON_NA <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb_2016 +
   STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY +
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
   STPP_COVAR_DF_TRACK_temp$SLOPE +
   STPP_COVAR_DF_TRACK_temp$ICE_CONC_2016 +
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2016_15 +
   STPP_COVAR_DF_TRACK_temp$CHL_2016 +
STPP_COVAR_DF_TRACK_temp$SUPPORT_Bb_2016

 # Create a variable that filters for any cells where the response variable or any covariate is NA
(WAP region only)
 STPP_COVAR_DF_TRACK_temp$NON_NA_WAP <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_Bb_2016 +
   STPP_COVAR_DF_TRACK_temp$DIST_SHELF + STPP_COVAR_DF_TRACK_temp$BATHY +
STPP_COVAR_DF_TRACK_temp$ENCLOSURE_MED +
   STPP_COVAR_DF_TRACK_temp$SLOPE +
   STPP_COVAR_DF_TRACK_temp$ICE_CONC_2016 +
STPP_COVAR_DF_TRACK_temp$DIST_ICE_2016_15 +
   STPP_COVAR_DF_TRACK_temp$CHL_2016 +
STPP_COVAR_DF_TRACK_temp$SUPPORT_WAP_Bb_2016

 # Create empty variable to house residuals of kernel regressed on covariates
 STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Bb_2016 <- NA

 # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),"STPP_LN_KE
RNEL_ORTHO_Bb_2016"] <- residuals(lm(STPP_LN_KERNEL_Bb_2016 ~ DIST_SHELF + BATHY +
ENCLOSURE_MED +
                                                        SLOPE + ICE_CONC_2016 +
DIST_ICE_2016_15 + CHL_2016,
```

```
                                                          data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA),]))

  # Create empty variable to house residuals of kernel regressed on covariates
  STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Bb_2016 <- NA

  # Calculate the residuals of a regression of the kernel intensity surface against all covariates

STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),"STPP_L
N_KERNEL_ORTHO_WAP_Bb_2016"] <- residuals(lm(STPP_LN_KERNEL_Bb_2016 ~ DIST_SHELF
+ BATHY + ENCLOSURE_MED +
                                                               SLOPE + ICE_CONC_2016 +
DIST_ICE_2016_15 + CHL_2016,
                                                            data =
STPP_COVAR_DF_TRACK_temp[!is.na(STPP_COVAR_DF_TRACK_temp$NON_NA_WAP),]))


  # Save the resulting orthogonal kernel intensity surface to a variable in STPP_COVAR
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Bb_2016_",i,sep = "")]] <- STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_Bb_2016_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_Bb_2016

  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_2016_",i,sep = "")]] <-
STPP_COVAR$INT
  STPP_COVAR[[paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_2016_",i,sep = "")]][] <-
STPP_COVAR_DF_TRACK_temp$STPP_LN_KERNEL_ORTHO_WAP_Bb_2016

}

# Clean-up
# remove(STPP_COVAR_DF_temp)
remove(STPP_COVAR_DF_TRACK_temp,i)


##### STPP_COVAR$COUNT_Mn #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_Mn_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Mn"
                                  & STPP_SIM_TRACKS$TRACK == i
                                  & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                  & STPP_SIM_TRACKS$MIGRATION == "PRE_MIG"
,c("LONG","LAT")],
                            STPP_COVAR$INT, fun="count")
```

```
  STPP_COVAR[[paste("COUNT_Mn_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_Mn_",i,sep="")]]@data@values),
                                  0,
                                  STPP_COVAR[[paste("COUNT_Mn_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)




##### STPP_COVAR$COUNT_Mn_2013 #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_Mn_2013_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Mn"
                                                    & STPP_SIM_TRACKS$TRACK == i
                                                    & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                                    & lubridate::year(STPP_SIM_TRACKS$Time) ==
2013
                                                    & lubridate::month(STPP_SIM_TRACKS$Time)
<= 8
                                                    & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                                          STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_Mn_2013_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_Mn_2013_",i,sep="")]]@data@values),
                                    0,

STPP_COVAR[[paste("COUNT_Mn_2013_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)

##### STPP_COVAR$COUNT_Mn_2016 #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_Mn_2016_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Mn"
                                                    & STPP_SIM_TRACKS$TRACK == i
```

```
                                                    & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                                    & lubridate::year(STPP_SIM_TRACKS$Time) ==
2016
                                                    & lubridate::month(STPP_SIM_TRACKS$Time)
<= 8
                                                    & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                                          STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_Mn_2016_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_Mn_2016_",i,sep="")]]@data@values),
                                     0,

STPP_COVAR[[paste("COUNT_Mn_2016_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)

##### STPP_COVAR$COUNT_Bb #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_Bb_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Bb"
                                                 & STPP_SIM_TRACKS$TRACK == i
                                                 & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                                 & STPP_SIM_TRACKS$MIGRATION == "PRE_MIG"
,c("LONG","LAT")],
                                       STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_Bb_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_Bb_",i,sep="")]]@data@values),
                                  0,
                                  STPP_COVAR[[paste("COUNT_Bb_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)


##### STPP_COVAR$COUNT_Bb_2013 #####
for(i in paste("TRACK_",1:10,sep="")){
```

```
  STPP_COVAR[[paste("COUNT_Bb_2013_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Bb"
                                  & STPP_SIM_TRACKS$TRACK == i
                                  & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                  & lubridate::year(STPP_SIM_TRACKS$Time) ==
2013
                                  & lubridate::month(STPP_SIM_TRACKS$Time)
<= 8
                                  & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                            STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_Bb_2013_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_Bb_2013_",i,sep="")]]@data@values),
                        0,

STPP_COVAR[[paste("COUNT_Bb_2013_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)

##### STPP_COVAR$COUNT_Bb_2016 #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_Bb_2016_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Bb"
                                  & STPP_SIM_TRACKS$TRACK == i
                                  & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                  & lubridate::year(STPP_SIM_TRACKS$Time) ==
2016
                                  & lubridate::month(STPP_SIM_TRACKS$Time)
<= 8
                                  & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                            STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_Bb_2016_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_Bb_2016_",i,sep="")]]@data@values),
                        0,

STPP_COVAR[[paste("COUNT_Bb_2016_",i,sep="")]]@data@values)
}
```

```
# Clean-up
remove(i)


##### STPP_COVAR$COUNT_WAP_Mn #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_WAP_Mn_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Mn"
                                             & STPP_SIM_TRACKS$TRACK == i
                                             & STPP_SIM_TRACKS$REGION == "WAP"
                                             & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                             & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                                     STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_WAP_Mn_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_WAP_Mn_",i,sep="")]]@data@values),
                              0,

STPP_COVAR[[paste("COUNT_WAP_Mn_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)

##### STPP_COVAR$COUNT_WAP_Mn_2013 #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_WAP_Mn_2013_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Mn"
                                              & STPP_SIM_TRACKS$TRACK == i
                                              & STPP_SIM_TRACKS$REGION == "WAP"
                                              & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                              & lubridate::year(STPP_SIM_TRACKS$Time)
== 2013
                                              & lubridate::month(STPP_SIM_TRACKS$Time)
<= 8
                                              & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                                      STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_WAP_Mn_2013_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_WAP_Mn_2013_",i,sep="")]]@data@values),
```

```
                                          0,
STPP_COVAR[[paste("COUNT_WAP_Mn_2013_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)

##### STPP_COVAR$COUNT_WAP_Mn_2016 #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_WAP_Mn_2016_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Mn"
                                        & STPP_SIM_TRACKS$TRACK == i
                                        & STPP_SIM_TRACKS$REGION == "WAP"
                                        & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                        & lubridate::year(STPP_SIM_TRACKS$Time)
== 2016
                                        & lubridate::month(STPP_SIM_TRACKS$Time)
<= 8
                                        & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                                 STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_WAP_Mn_2016_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_WAP_Mn_2016_",i,sep="")]]@data@values),
                                          0,
STPP_COVAR[[paste("COUNT_WAP_Mn_2016_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)


##### STPP_COVAR$COUNT_WAP_Bb #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_WAP_Bb_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Bb"
                                      & STPP_SIM_TRACKS$TRACK == i
                                      & STPP_SIM_TRACKS$REGION == "WAP"
                                      & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
```

```
                                                                 & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                                                     STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_WAP_Bb_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_WAP_Bb_",i,sep="")]]@data@values),
                                              0,

STPP_COVAR[[paste("COUNT_WAP_Bb_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)


##### STPP_COVAR$COUNT_WAP_Bb_2013 #####
for(i in paste("TRACK_",1:10,sep="")){

  STPP_COVAR[[paste("COUNT_WAP_Bb_2013_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Bb"
                                                                    & STPP_SIM_TRACKS$TRACK == i
                                                                    & STPP_SIM_TRACKS$REGION == "WAP"
                                                                    & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                                                    & lubridate::year(STPP_SIM_TRACKS$Time)
== 2013
                                                                    & lubridate::month(STPP_SIM_TRACKS$Time)
<= 8
                                                                    & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                                                        STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_WAP_Bb_2013_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_WAP_Bb_2013_",i,sep="")]]@data@values),
                                                 0,

STPP_COVAR[[paste("COUNT_WAP_Bb_2013_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)

##### STPP_COVAR$COUNT_WAP_Bb_2016 #####
for(i in paste("TRACK_",1:10,sep="")){
```

```
  STPP_COVAR[[paste("COUNT_WAP_Bb_2016_",i,sep="")]] <-
raster::rasterize(STPP_SIM_TRACKS[STPP_SIM_TRACKS$SPP == "Bb"
                                          & STPP_SIM_TRACKS$TRACK == i
                                          & STPP_SIM_TRACKS$REGION == "WAP"
                                          & STPP_SIM_TRACKS$OBS_TIME_DIFF < 24
                                          & lubridate::year(STPP_SIM_TRACKS$Time)
== 2016
                                          & lubridate::month(STPP_SIM_TRACKS$Time)
<= 8
                                          & STPP_SIM_TRACKS$MIGRATION ==
"PRE_MIG" ,c("LONG","LAT")],
                                   STPP_COVAR$INT, fun="count")

  STPP_COVAR[[paste("COUNT_WAP_Bb_2016_",i,sep="")]][] <-
ifelse(is.na(STPP_COVAR[[paste("COUNT_WAP_Bb_2016_",i,sep="")]]@data@values),
                               0,

STPP_COVAR[[paste("COUNT_WAP_Bb_2016_",i,sep="")]]@data@values)
}

# Clean-up
remove(i)




####### STPP_COVAR_DF #####

# Convert raster grid of covariates and response variable to dataframe (1 row per grid cell)

STPP_COVAR_DF <- as.data.frame(as(STPP_COVAR,"SpatialPointsDataFrame"), stringsAsFactors
= F)

STPP_COVAR_DF <- rbind(data.frame(STPP_COVAR_DF,TRACK = "TRACK_1", stringsAsFactors =
F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_2", stringsAsFactors = F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_3", stringsAsFactors = F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_4", stringsAsFactors = F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_5", stringsAsFactors = F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_6", stringsAsFactors = F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_7", stringsAsFactors = F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_8", stringsAsFactors = F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_9", stringsAsFactors = F),
          data.frame(STPP_COVAR_DF,TRACK = "TRACK_10", stringsAsFactors = F))
```

```
for(i in unique(STPP_COVAR_DF$TRACK)){
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"COUNT_Bb"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("COUNT_Bb_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_Bb"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_Bb_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_WAP_Bb"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_WAP_Bb_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_Bb"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_Bb_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_WAP_Bb"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"COUNT_Bb_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("COUNT_Bb_2013_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_Bb_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_Bb_2013_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_WAP_Bb_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_WAP_Bb_2013_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_Bb_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_Bb_2013_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_WAP_Bb_2013"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_2013_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"COUNT_Bb_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("COUNT_Bb_2016_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_Bb_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_Bb_2016_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_WAP_Bb_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_WAP_Bb_2016_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_Bb_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_Bb_2016_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_WAP_Bb_2016"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_WAP_Bb_2016_", i, sep="")]
}
```

```
for(i in unique(STPP_COVAR_DF$TRACK)){
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"COUNT_Mn"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("COUNT_Mn_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_Mn"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_Mn_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_WAP_Mn"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_WAP_Mn_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_Mn"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_Mn_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_WAP_Mn"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_",
i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_Mn"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_Mn_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_WAP_Mn"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_",
i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"COUNT_Mn_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("COUNT_Mn_2013_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_Mn_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_Mn_2013_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_WAP_Mn_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_WAP_Mn_2013_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_Mn_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_Mn_2013_",
i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_WAP_Mn_2013"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_2013_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"COUNT_Mn_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("COUNT_Mn_2016_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_Mn_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_Mn_2016_", i,
sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_WAP_Mn_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_WAP_Mn_2016_", i,
sep="")]
```

```
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_Mn_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_Mn_2016_",
i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_WAP_Mn_2016"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_WAP_Mn_2016_", i, sep="")]
}

for(i in unique(STPP_COVAR_DF$TRACK)){
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_CLASS_Bb"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_CLASS_Bb_",
i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_CLASS_WAP_Bb"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_WAP_Bb_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_CLASS_Bb_2013"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_Bb_2013_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,"STPP_LN_KERNEL_ORTHO_CLASS_WAP_Bb_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_WAP_Bb_2013_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_CLASS_Bb_2016"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_Bb_2016_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,"STPP_LN_KERNEL_ORTHO_CLASS_WAP_Bb_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_WAP_Bb_2016_", i, sep="")]
}



for(i in unique(STPP_COVAR_DF$TRACK)){
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_CLASS_Mn"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,paste("STPP_LN_KERNEL_ORTHO_CLASS_Mn_",
i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_CLASS_WAP_Mn"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_WAP_Mn_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_CLASS_Mn_2013"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_Mn_2013_", i, sep="")]
```

```
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,"STPP_LN_KERNEL_ORTHO_CLASS_WAP_Mn_2013"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_WAP_Mn_2013_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK == i,"STPP_LN_KERNEL_ORTHO_CLASS_Mn_2016"]
<- STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_Mn_2016_", i, sep="")]
  STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,"STPP_LN_KERNEL_ORTHO_CLASS_WAP_Mn_2016"] <-
STPP_COVAR_DF[STPP_COVAR_DF$TRACK ==
i,paste("STPP_LN_KERNEL_ORTHO_CLASS_WAP_Mn_2016_", i, sep="")]
}



# Clean-up
remove(i)


# Convert COAST_CLASS ordinal variables from numeric to factor
STPP_COVAR_DF$COAST_CLASS <- as.factor(STPP_COVAR_DF$COAST_CLASS)
STPP_COVAR_DF$COAST_CLASS_GEN <- as.factor(STPP_COVAR_DF$COAST_CLASS_GEN)

# After consolidating per TRACK information into columns,
# remove columns specific to each TRACK to reduce data.frame memory size
STPP_COVAR_DF <- STPP_COVAR_DF[,colnames(STPP_COVAR_DF)[-1*grep("TRACK_",
colnames(STPP_COVAR_DF))]]


# Save a copy for being able to reconstitute directly without re-running algorithm
saveRDS(object = STPP_COVAR_DF,
     file = paste("/Users/trevor.joyce/Grad School/Research/",
             "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
             "STPP_COVAR_DF.rds",sep=""))


STPP_COVAR_DF <- readRDS(file = paste("/Users/trevor.joyce/Grad School/Research/",
                         "1_2016_Antarctic Whale Tracking/Point Process Model Example/",
                         "STPP_COVAR_DF.rds",sep=""))

###### STPP_FIT ######

STPP_FIT <- list()
```

```
##### STPP_FIT$ORTHO_WAP_Bb_2013_SMOOTH ######

#### Fit a Poisson generalized additive model (GAM) to the
#### counts of telemetry locations within each cell (yl)
#### with a log link function, kernel availability surface, and
#### spatially correlated random effect.

STPP_FIT$ORTHO_WAP_Bb_2013_SMOOTH <- mgcv::bam(COUNT_Bb_2013 ~
s(STPP_LN_KERNEL_ORTHO_WAP_Bb_2013)
                              + s(DIST_SHELF, bs = "ts", k = 5)
                              + s(BATHY, bs = "ts", k = 5)
                              + s(ENCLOSURE_MED, bs = "ts", k = 5)
                              + s(SLOPE, bs = "ts", k = 5)
                              + s(ICE_CONC_2013, bs = "ts", k = 5)
                              + s(DIST_ICE_2013_15, bs = "ts", k = 5)
                              + s(CHL_2013, bs = "ts", k = 5)
                              + s(DIST_E,DIST_N),
                              family = "poisson",
                              weights=rep(1/10,

nrow(STPP_COVAR_DF[!is.na(STPP_COVAR_DF$SUPPORT_WAP_Bb_2013),])),

data=STPP_COVAR_DF[!is.na(STPP_COVAR_DF$SUPPORT_WAP_Bb_2013),],
                              select = T)

# Print model summary information
mgcv::summary(STPP_FIT$ORTHO_WAP_Bb_2013_SMOOTH)


########  STPP_COVAR$STPP_FIT_ORTHO_WAP_Bb_2013_SMOOTH #####

# Calculate the log KDE effect surface
STPP_COVAR$STPP_FIT_ORTHO_WAP_Bb_2013_SMOOTH = STPP_COVAR$INT
STPP_COVAR$STPP_FIT_ORTHO_WAP_Bb_2013_SMOOTH[] =
rowSums(mgcv::predict.bam(object = STPP_FIT$ORTHO_WAP_Bb_2013_SMOOTH,
                                             newdata = STPP_COVAR_DF,
                                             type = "terms",
                                             terms =
c("s(DIST_SHELF)","s(BATHY)","s(ENCLOSURE_MED)",

"s(SLOPE)","s(ICE_CONC_2013)","s(DIST_ICE_2013_15)",
                                                    "s(CHL_2013)")))
##### STPP_FIT$ORTHO_WAP_Mn_2013_SMOOTH ######
```

```
#### Fit a Poisson generalized additive model (GAM) to the
#### counts of telemetry locations within each cell (yl)
#### with a log link function, kernel availability surface, and
#### spatially correlated random effect.

STPP_FIT$ORTHO_WAP_Mn_2013_SMOOTH <- mgcv::bam(COUNT_Mn_2013 ~
s(STPP_LN_KERNEL_ORTHO_WAP_Mn_2013)
                              + s(DIST_SHELF, bs = "ts", k = 5)
                              + s(BATHY, bs = "ts", k = 5)
                              + s(ENCLOSURE_MED, bs = "ts", k = 5)
                              + s(SLOPE, bs = "ts", k = 5)
                              + s(ICE_CONC_2013, bs = "ts", k = 5)
                              + s(DIST_ICE_2013_15, bs = "ts", k = 5)
                              + s(CHL_2013, bs = "ts", k = 5)
                              + s(DIST_E,DIST_N),
                              family = "poisson",
                              weights=rep(1/10,

nrow(STPP_COVAR_DF[!is.na(STPP_COVAR_DF$SUPPORT_WAP_Mn_2013),])),

data=STPP_COVAR_DF[!is.na(STPP_COVAR_DF$SUPPORT_WAP_Mn_2013),],
                              select = T)

# Print model summary information
summary(STPP_FIT$ORTHO_WAP_Mn_2013_SMOOTH)


########  STPP_COVAR$STPP_FIT_ORTHO_WAP_Mn_2013_SMOOTH #####

# Calculate the log KDE effect surface
STPP_COVAR$STPP_FIT_ORTHO_WAP_Mn_2013_SMOOTH = STPP_COVAR$INT
STPP_COVAR$STPP_FIT_ORTHO_WAP_Mn_2013_SMOOTH[] =
rowSums(mgcv::predict.bam(object = STPP_FIT$ORTHO_WAP_Mn_2013_SMOOTH,
                                    newdata = STPP_COVAR_DF,
                                    type = "terms",
                                    terms =
c("s(DIST_SHELF)","s(BATHY)","s(ENCLOSURE_MED)",

"s(SLOPE)","s(ICE_CONC_2013)","s(DIST_ICE_2013_15)",
                                          "s(CHL_2013)")))
```

```
##### Script to Prepare Covariates for Analyses Presented in:

##### Sympatry and resource partitioning between the largest krill consumers around the
Antarctic Peninsula
##### Ari S. Friedlaender1*, Trevor Joyce2, John W. Durban3, David W. Johnston4, Andrew J.
Read4,
##### Douglas P. Nowacek4#, Jeremy A. Goldbogen5, and Nick Gales6

##### Script developed by Trevor Joyce
##### Version 3.0 (May 6, 2021)


####### COAST_MED #########

# #Import a COAST outline file derived from the British Antarctic Survey Geodata Portal -
Antarctic Digital Database server output
# #dowloaded as a shapefile from
(http://add.antarctica.ac.uk/repository/entry/show?entryid=f477219b-9121-44d6-afa6-
d8552762dc45)
# #that has been projected from its native Antarctic Polar Stereographic (ESPG: 3031) projected
coordinate system
# #into GCS_WGS84 (EPSG: 4326) in ESRI ArcGIS v10.3

COAST_MED=rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad
School/Research/1_2016_Antarctic Whale
Tracking/Data/Coastline/Coastline_medium_res_polygon",sep=""),
                  layer="Coastline_medium_res_polygon_GCS_WGS84")


####### COAST_WAP #########

# #Import a COAST outline file derived from the British Antarctic Survey Geodata Portal -
Antarctic Digital Database server output
# #dowloaded as a shapefile from
(http://add.antarctica.ac.uk/repository/entry/show?entryid=f477219b-9121-44d6-afa6-
d8552762dc45)
# #that has been projected from its native Antarctic Polar Stereographic (ESPG: 3031) projected
coordinate system
# #into GCS_WGS84 (EPSG: 4326) in ESRI ArcGIS v10.3

COAST_WAP=rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad
School/Research/1_2016_Antarctic Whale Tracking/",
                   "Data/Coastline/Coastline_high_res_polygon",sep=""),
             layer="Coastline_high_res_polygon_GCS_WGS84")
```

```
# Crop polygons to the region of interest
COAST_WAP <- raster::crop(COAST_WAP,raster::extent(c(xmin = -78,
                                      xmax = -52,
                                      ymin = -71,
                                      ymax = -61)))

# Unify polygons to remove boundaries between Iceshelves and Land
COAST_WAP = rgeos::gUnaryUnion(COAST_WAP)


####### COAST_BUFFER_500m #########

#Import high resolution COAST layer to which a geodesic buffer has been added in ArcGIS 10.4
COAST_BUFFER_500m=rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad
School/Research/1_2016_Antarctic Whale Tracking/",
                           "Data/Coastline",sep=""),
                    layer="COAST_GCS_WGS4_Clipped_500m_Buffer")

COAST_BUFFER_500m = rgeos::gUnaryUnion(COAST_BUFFER_500m)


###### ICE_MODIS ####

ICE_MODIS <- raster::raster(paste("~/Grad School/Research/1_2016_Antarctic Whale
Tracking/Data/Ice Data/MODIS/MODIS_Aqua/",
                          "MODIS_Aqua_TrueColor_","2013_01_07",".tif",sep=""))


###### ICE_MODIS_LAND_MASK ####

# ICE_MODIS_LAND_MASK =  raster::rasterize(COAST_WAP,
#                                 ICE_MODIS)
#
# ICE_MODIS_LAND_MASK[is.na(ICE_MODIS_LAND_MASK)] <- -500
# ICE_MODIS_LAND_MASK[ICE_MODIS_LAND_MASK>0] <- NA
# ICE_MODIS_LAND_MASK[ICE_MODIS_LAND_MASK<=0] <- 1
#
# raster::writeRaster(ICE_MODIS_LAND_MASK,
#                 filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                           "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                           "ICE_MODIS_LAND_MASK.tif",sep=""),
#                 format = "GTiff", overwrite=TRUE)
```

```
ICE_MODIS_LAND_MASK = raster::raster(paste("/Users/trevorjoyce/Grad School/Research/",
                            "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                            "ICE_MODIS_LAND_MASK.tif",sep=""))


###### ICE_MODIS_LAND_MASK_BUFFER_500m ####

#This will be used to exclude erroneous near shore values in ICE_MODIS_CONC_COMP

# ICE_MODIS_LAND_MASK_BUFFER_500m =  raster::rasterize(COAST_BUFFER_500m,
#                                ICE_MODIS)
#
# ICE_MODIS_LAND_MASK_BUFFER_500m[is.na(ICE_MODIS_LAND_MASK_BUFFER_500m)] <- -
500
# ICE_MODIS_LAND_MASK_BUFFER_500m[ICE_MODIS_LAND_MASK_BUFFER_500m>0] <- NA
# ICE_MODIS_LAND_MASK_BUFFER_500m[ICE_MODIS_LAND_MASK_BUFFER_500m<=0] <- 1
#
# raster::writeRaster(ICE_MODIS_LAND_MASK_BUFFER_500m,
#             filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                       "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                       "ICE_MODIS_LAND_MASK_BUFFER_500m.tif",sep=""),
#             format = "GTiff", overwrite=TRUE)

ICE_MODIS_LAND_MASK_BUFFER_500m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                                    "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                                    "ICE_MODIS_LAND_MASK_BUFFER_500m.tif",sep=""))


###### ICE_MODIS_LAND_MASK_RES_2000m ####

#This will be as the basis for CTMC analysis

# ICE_MODIS_LAND_MASK_RES_2000m <- raster::projectRaster(from =
ICE_MODIS_LAND_MASK,
#                                  res = c(0.040,0.020),crs = CRS("+init=epsg:4326"),
#                                  method = "bilinear")
#
# ICE_MODIS_LAND_MASK_RES_2000m[!is.na(ICE_MODIS_LAND_MASK_RES_2000m)] <- 1
#
# ICE_MODIS_LAND_MASK_RES_2000m <-
raster::crop(ICE_MODIS_LAND_MASK_RES_2000m,raster::extent(ICE_MODIS_LAND_MASK))
#
```

```
# raster::writeRaster(ICE_MODIS_LAND_MASK_RES_2000m,
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                         "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                         "ICE_MODIS_LAND_MASK_RES_2000m.tif",sep=""),
#               format = "GTiff", overwrite=TRUE)

ICE_MODIS_LAND_MASK_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                                    "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                                    "ICE_MODIS_LAND_MASK_RES_2000m.tif",sep=""))


###### WAP_MASK_RES_2000m ####

# #This will be used to exclude the WEDDELL_SEA
# WEDDELL_SEA <- data.frame(LONG = c(-57.3,-56.4,-45.5,-14.0,-06.0,-28.0,-58.0,-65.0,-64.0,-
57.3),
#                   LAT =  c(-63.25,-63.0,-61.1,-63.5,-71.0,-78.0,-77.0,-73.0,-66.0,-63.25))
# WEDDELL_SEA <-
sp::SpatialPolygonsDataFrame(sp::SpatialPolygons(list(sp::Polygons(list(sp::Polygon(WEDDELL_
SEA)), ID = 1))),
#                               data = data.frame(ID = "WEDD"), match.ID = F)
# raster::projection(WEDDELL_SEA) <- sp::CRS("+init=epsg:4326")
#
# WAP_MASK_RES_2000m =  raster::rasterize(WEDDELL_SEA,
#                             ICE_MODIS_LAND_MASK_RES_2000m)
#
#
#
# WAP_MASK_RES_2000m[is.na(WAP_MASK_RES_2000m)] <- -1
# WAP_MASK_RES_2000m[WAP_MASK_RES_2000m>0] <- NA
# WAP_MASK_RES_2000m[!is.na(WAP_MASK_RES_2000m)] <- 1
#
# WAP_MASK_RES_2000m <- WAP_MASK_RES_2000m * ICE_MODIS_LAND_MASK_RES_2000m
#
# raster::writeRaster(WAP_MASK_RES_2000m,
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                         "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                         "WAP_MASK_RES_2000m.tif",sep=""),
#               format = "GTiff", overwrite=TRUE)

WAP_MASK_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad School/Research/",
                                    "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                                    "WAP_MASK_RES_2000m.tif",sep=""))
```

```
###### ICE_MODIS_CONC_MASK ####

# ICE_MODIS_CONC_MASK = list()
#
# for(i in unique(substr(list.files(paste("/Users/trevorjoyce/Grad School/Research/",
#                                 "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                                 "MODIS_WAP_Ice_Concentration/Aqua",sep="")),1,39))){
#
#   ICE_MODIS_CONC_MASK[[paste("X",substr(i,30,39),"_A",sep="")]] =
rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad School/Research/",
#                                                               "1_2016_Antarctic Whale
Tracking/Data/Ice Data/",
#
"MODIS_WAP_Ice_Concentration/Aqua",sep=""),
#                                                        layer=i)
#   ICE_MODIS_CONC_MASK[[paste("X",substr(i,30,39),"_A",sep="")]] =
sp::spChFIDs(ICE_MODIS_CONC_MASK[[paste("X",substr(i,30,39),"_A",sep="")]],
#                                                    paste(paste("X",substr(i,30,39),"_A",sep=""),
#
rownames(as.data.frame(ICE_MODIS_CONC_MASK[[paste("X",substr(i,30,39),"_A",sep="")]])),s
ep="_"))
#   ICE_MODIS_CONC_MASK[[paste("X",substr(i,30,39),"_A",sep="")]]$ID =
paste("X",substr(i,30,39),"_A",sep="")
#
#   raster::projection(ICE_MODIS_CONC_MASK[[paste("X",substr(i,30,39),"_A",sep="")]]) <-
sp::CRS("+init=epsg:4326")
# }
#
#
# # Import shapefiles that define cloud free areas from Terra imagery
# for(i in unique(substr(list.files(paste("/Users/trevorjoyce/Grad School/Research/",
#                                 "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                                 "MODIS_WAP_Ice_Concentration/Terra",sep="")),1,40))){
#
#   ICE_MODIS_CONC_MASK[[paste("X",substr(i,31,40),"_T",sep="")]] =
rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad School/Research/",
#                                                               "1_2016_Antarctic Whale
Tracking/Data/Ice Data/",
#
"MODIS_WAP_Ice_Concentration/Terra",sep=""),
#                                                        layer=i)
```

```
#   ICE_MODIS_CONC_MASK[[paste("X",substr(i,31,40),"_T",sep="")]] =
sp::spChFIDs(ICE_MODIS_CONC_MASK[[paste("X",substr(i,31,40),"_T",sep="")]],
#                                              paste(paste("X",substr(i,31,40),"_T",sep=""),
#
rownames(as.data.frame(ICE_MODIS_CONC_MASK[[paste("X",substr(i,31,40),"_T",sep="")]])),s
ep="_"))
#   ICE_MODIS_CONC_MASK[[paste("X",substr(i,31,40),"_T",sep="")]]$ID =
paste("X",substr(i,31,40),"_T",sep="")
#
#   raster::projection(ICE_MODIS_CONC_MASK[[paste("X",substr(i,31,40),"_T",sep="")]]) <-
sp::CRS("+init=epsg:4326")
# }
#
# ICE_MODIS_CONC_MASK = do.call("rbind",ICE_MODIS_CONC_MASK)
#
# ICE_MODIS_CONC_MASK$RASTER_DATES = substr(ICE_MODIS_CONC_MASK$ID,1,11)
# ICE_MODIS_CONC_MASK$YEAR=as.numeric(substr(ICE_MODIS_CONC_MASK$ID,2,5))
# ICE_MODIS_CONC_MASK$MONTH=as.numeric(substr(ICE_MODIS_CONC_MASK$ID,7,8))
# ICE_MODIS_CONC_MASK$DAY=as.numeric(substr(ICE_MODIS_CONC_MASK$ID,10,11))
#
# saveRDS(object = ICE_MODIS_CONC_MASK,
#        file = paste("/Users/trevorjoyce/Grad School/Research/",
#                  "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                  "ICE_MODIS_CONC_MASK.rds",sep=""))

ICE_MODIS_CONC_MASK <- readRDS(file = paste("/Users/trevorjoyce/Grad School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                              "ICE_MODIS_CONC_MASK.rds",sep=""))


###### ICE_MODIS_CONC ####

# ICE_MODIS_CONC <- raster::stack()
#
# for(i in unique(ICE_MODIS_CONC_MASK$ID)){
#
#   ICE_MODIS_CONC_MASK_temp <-
raster::rasterize(x=ICE_MODIS_CONC_MASK[ICE_MODIS_CONC_MASK$ID==i,],
#                                y=ICE_MODIS_LAND_MASK_500m)
#   ICE_MODIS_CONC_MASK_temp[ICE_MODIS_CONC_MASK_temp>0] = 1
#
#   if(substr(i,13,13) == "A")
#   {ICE_MODIS_CONC_temp <- raster::raster(paste("~/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/Ice Data/MODIS/MODIS_Aqua/",
```

```
#                                     "MODIS_Aqua_TrueColor_",substr(i,2,11),".tif",sep=""))}
#
#   if(substr(i,13,13) == "T")
#   {ICE_MODIS_CONC_temp <- raster::raster(paste("~/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/Ice Data/MODIS/MODIS_Terra/",
#                                     "MODIS_Terra_TrueColor_",substr(i,2,11),".tif",sep=""))}
#
#   ICE_MODIS_CONC_temp = ICE_MODIS_CONC_temp * ICE_MODIS_CONC_MASK_temp *
ICE_MODIS_LAND_MASK_500m
#
#   names(ICE_MODIS_CONC_temp) <- i
#
#   ICE_MODIS_CONC <- raster::stack(ICE_MODIS_CONC,ICE_MODIS_CONC_temp)
# }
#
# remove(ICE_MODIS_CONC_temp, ICE_MODIS_CONC_MASK_temp)
#
# # Save to individual raster files in common geotiff format (more secure and transferable than
.rds)
#
# raster::writeRaster(ICE_MODIS_CONC,
#                 filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                            "1_2016_Antarctic Whale Tracking/Data/Ice Data/ICE_MODIS_CONC/",
#                            "ICE_MODIS_CONC.tif",sep=""),
#                 format = "GTiff",bylayer = T,suffix = "names")

# Reconstitute ICE_MODIS_CONC from individual rasters

ICE_MODIS_CONC<- raster::stack()

for(i in list.files(paste("/Users/trevorjoyce/Grad School/Research/",
                  "1_2016_Antarctic Whale Tracking/Data/Ice Data/ICE_MODIS_CONC/",sep=""))){

  ICE_MODIS_CONC_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Grad
School/Research/",
                                        "1_2016_Antarctic Whale Tracking/Data/Ice
Data/ICE_MODIS_CONC/",i,sep="")))

  names(ICE_MODIS_CONC_temp) <- substr(i,16,28)

  ICE_MODIS_CONC <- raster::stack(ICE_MODIS_CONC,ICE_MODIS_CONC_temp)

}
```

```
remove(ICE_MODIS_CONC_temp)

###### ICE_MODIS_CONC_COMP #############

# ICE_MODIS_CONC_COMP = ICE_MODIS_CONC
# ICE_MODIS_CONC_COMP = (ICE_MODIS_CONC_COMP-30)/(255-30)
# ICE_MODIS_CONC_COMP[ICE_MODIS_CONC_COMP<0] = 0

# Save to individual raster files in geotiff format (more secure and transferable than .rds)

# raster::writeRaster(ICE_MODIS_CONC_COMP,
#                 filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                                "1_2016_Antarctic Whale Tracking/Data/Ice
Data/ICE_MODIS_CONC_COMP/",
#                                "ICE_MODIS_CONC_COMP.tif",sep=""),
#                 format = "GTiff",bylayer = T,suffix = "names")

# Reconstitute ICE_MODIS_CONC from individual rasters

ICE_MODIS_CONC_COMP<- raster::stack()

for(i in 1:64){


ICE_MODIS_CONC_COMP_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Gra
d School/Research/",
                                            "1_2016_Antarctic Whale Tracking/Data/Ice
Data/ICE_MODIS_CONC_COMP/",
                                            "ICE_MODIS_CONC_COMP_layer.",i,".tif",sep="")))

  names(ICE_MODIS_CONC_COMP_temp) <- names(ICE_MODIS_CONC)[i]

  ICE_MODIS_CONC_COMP <-
raster::stack(ICE_MODIS_CONC_COMP,ICE_MODIS_CONC_COMP_temp)

}

remove(ICE_MODIS_CONC_COMP_temp)


###### ICE_MODIS_CONC_COMP_ALL #############

# ICE_MODIS_CONC_COMP_ALL = calc(x = ICE_MODIS_CONC_COMP,fun = mean, na.rm = T)
#
```

```
#
# raster::writeRaster(ICE_MODIS_CONC_COMP_ALL,
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                         "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                         "ICE_MODIS_CONC_COMP_ALL.tif",sep=""),
#               format = "GTiff")
#

ICE_MODIS_CONC_COMP_ALL = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                              "ICE_MODIS_CONC_COMP_ALL.tif",sep=""))


###### ICE_MODIS_CONC_COMP_2013 #############

# #Calculate mean ICE_MODIS_CONC sea ice concentration index from the cloud free areas
# ICE_MODIS_CONC_COMP_2013 = calc(x =
ICE_MODIS_CONC_COMP[[which(substr(names(ICE_MODIS_CONC_COMP),2,5)%in%"2013")]],f
un = mean, na.rm = T)
#
# #Replace any NA values in specific year with mean ICE_MODIS_CONC sea ice concentration
index values
# #across all years of sampling (mostly fills in open water areas off WAP)
# ICE_MODIS_CONC_COMP_2013[is.na(as.vector(ICE_MODIS_CONC_COMP_2013))]<-
ICE_MODIS_CONC_COMP_ALL[is.na(as.vector(ICE_MODIS_CONC_COMP_2013))]
#
# raster::writeRaster(ICE_MODIS_CONC_COMP_2013,
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                         "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                         "ICE_MODIS_CONC_COMP_2013.tif",sep=""),
#               format = "GTiff",overwrite=TRUE)


ICE_MODIS_CONC_COMP_2013 = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                              "ICE_MODIS_CONC_COMP_2013.tif",sep=""))

###### ICE_MODIS_CONC_COMP_2016 #############

# #Calculate mean ICE_MODIS_CONC sea ice concentration index from the cloud free areas
# ICE_MODIS_CONC_COMP_2016 = calc(x =
ICE_MODIS_CONC_COMP[[which(substr(names(ICE_MODIS_CONC_COMP),2,5)%in%"2016")]],
```

```
#                       fun = mean, na.rm = T)
#
# #Replace any NA values in specific year with mean ICE_MODIS_CONC sea ice concentration
index values
# #across all years of sampling (mostly fills in open water areas off WAP)
# ICE_MODIS_CONC_COMP_2016[is.na(as.vector(ICE_MODIS_CONC_COMP_2016))]<-
ICE_MODIS_CONC_COMP_ALL[is.na(as.vector(ICE_MODIS_CONC_COMP_2016))]
#
#
# raster::writeRaster(ICE_MODIS_CONC_COMP_2016,
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                         "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                         "ICE_MODIS_CONC_COMP_2016.tif",sep=""),
#               format = "GTiff")
#

ICE_MODIS_CONC_COMP_2016 = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                              "ICE_MODIS_CONC_COMP_2016.tif",sep=""))

###### ICE_MODIS_CONC_COMP_2013_RES_2000m  #############


# ICE_MODIS_CONC_COMP_2013_RES_2000m <- raster::resample(x =
ICE_MODIS_CONC_COMP_2013,
#                                      y = ICE_MODIS_LAND_MASK_RES_2000m,
#                                      method = "bilinear")
#
# raster::writeRaster(ICE_MODIS_CONC_COMP_2013_RES_2000m,
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                         "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                         "ICE_MODIS_CONC_COMP_2013_RES_2000m.tif",sep=""),
#               format = "GTiff", overwrite=TRUE)
#

ICE_MODIS_CONC_COMP_2013_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                                   "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                                   "ICE_MODIS_CONC_COMP_2013_RES_2000m.tif",sep=""))

###### ICE_MODIS_CONC_COMP_2016_RES_2000m  #############
```

```
# ICE_MODIS_CONC_COMP_2016_RES_2000m <- raster::resample(x =
ICE_MODIS_CONC_COMP_2016,
#                                      y = ICE_MODIS_LAND_MASK_RES_2000m,
#                                      method = "bilinear")
#
# raster::writeRaster(ICE_MODIS_CONC_COMP_2016_RES_2000m,
#              filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                        "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
#                        "ICE_MODIS_CONC_COMP_2016_RES_2000m.tif",sep=""),
#              format = "GTiff", overwrite=TRUE)
#

ICE_MODIS_CONC_COMP_2016_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                                     "1_2016_Antarctic Whale Tracking/Data/Ice Data/",
                                     "ICE_MODIS_CONC_COMP_2016_RES_2000m.tif",sep=""))



###### DIST_ICE_MODIS_POLAR  #############

# Calculate 30% and 15% contour lines from ICE_MODIS_CONC_COMP_2013_RES_2000m
# and ICE_MODIS_CONC_COMP_2016_RES_2000m
#
# arcpy.gp.ContourList_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Ice/ICE_MODIS_CONC_COMP_2013_RES_2000m.tif",
#                "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ice_modis_conc_comp_2013_res_2000m_contour_30.shp",
#                "0.3")
#
# arcpy.gp.ContourList_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Ice/ICE_MODIS_CONC_COMP_2013_RES_2000m.tif",
#                "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ice_modis_conc_comp_2013_res_2000m_contour_15.shp",
#                "0.15")
#
#
# arcpy.gp.ContourList_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Ice/ICE_MODIS_CONC_COMP_2016_RES_2000m.tif",
#                "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ice_modis_conc_comp_2016_res_2000m_contour_30.shp",
```

```
#                 "0.3")
#
#
# arcpy.gp.ContourList_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Ice/ICE_MODIS_CONC_COMP_2016_RES_2000m.tif",
#                 "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ice_modis_conc_comp_2016_res_2000m_contour_15.shp",
#                 "0.15")

# Calculate path distance from 30% and 15% contour lines using fine resolution
# Antarctic Polar Stereographic grid (ICE_MODIS_LAND_MASK_POLAR projected in m)
# Note: important to extent to match ICE_MODIS_LAND_MASK_POLAR under
# Environments > Processing Extent > Extent > "Same as layer
ICE_MODIS_LAND_MASK_POLAR.tif"
#
# arcpy.gp.PathDistance_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Ice/ice_modis_conc_comp_2013_res_2000m_contour_30.shp",
#                 "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS
Data/Ice/ICE_MODIS_CONC_COMP_2013_RES_2000m_30_PathDist_POLAR.tif",
#                 "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ICE_MODIS_LAND_MASK_POLAR.tif",
#                 "", "", "BINARY 1 45", "", "BINARY 1 -30 30", "", "", "", "", "", "")
#
# arcpy.gp.PathDistance_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Ice/ice_modis_conc_comp_2013_res_2000m_contour_15.shp",
#                 "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS
Data/Ice/ICE_MODIS_CONC_COMP_2013_RES_2000m_15_PathDist_POLAR.tif",
#                 "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ICE_MODIS_LAND_MASK_POLAR.tif",
#                 "", "", "BINARY 1 45", "", "BINARY 1 -30 30", "", "", "", "", "", "")
#
# arcpy.gp.PathDistance_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Ice/ice_modis_conc_comp_2016_res_2000m_contour_30.shp",
#                 "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS
Data/Ice/ICE_MODIS_CONC_COMP_2016_RES_2000m_30_PathDist_POLAR.tif",
#                 "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ICE_MODIS_LAND_MASK_POLAR.tif",
```

```
#                   "", "", "BINARY 1 45", "", "BINARY 1 -30 30", "", "", "", "", "", "")
#
# arcpy.gp.PathDistance_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Ice/ice_modis_conc_comp_2016_res_2000m_contour_15.shp",
#                   "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS
Data/Ice/ICE_MODIS_CONC_COMP_2016_RES_2000m_15_PathDist_POLAR.tif",
#                   "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ICE_MODIS_LAND_MASK_POLAR.tif",
#                   "", "", "BINARY 1 45", "", "BINARY 1 -30 30", "", "", "", "", "", "")


DIST_ICE_MODIS_POLAR <- raster::stack()

for(i in c(2013,2016)){
 for(j in c(15,30)){


DIST_ICE_MODIS_POLAR_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Grad
School/Research/",
                                         "1_2016_Antarctic Whale Tracking/Data/Ice Data/",

"ICE_MODIS_CONC_COMP_",i,"_RES_2000m_",j,"_PathDist_POLAR.tif",sep="")))
   names(DIST_ICE_MODIS_POLAR_temp) <- paste("X",i,"_",j,sep="")

   DIST_ICE_MODIS_POLAR <- raster::stack(DIST_ICE_MODIS_POLAR,
                            DIST_ICE_MODIS_POLAR_temp)
 }

}

remove(DIST_ICE_MODIS_POLAR_temp)

raster::projection(DIST_ICE_MODIS_POLAR) <- sp::CRS("+init=epsg:3031")

# ###### DIST_ICE_MODIS_RES_2000m #########
#
# DIST_ICE_MODIS_RES_2000m <- raster::projectRaster(from = DIST_ICE_MODIS_POLAR,
#                             to = ICE_MODIS_LAND_MASK_RES_2000m,
#                             crs = CRS("+init=epsg:4326"),
#                             method = "bilinear")
#
# # Create a temporary layer to set ICE_MODIS_CONC_COMP values greater
```

```
# # than threshold (e.g, 0.15, 0.30) to negative
# ICE_MODIS_CONC_COMP_2013_RES_2000m_temp <-
ICE_MODIS_CONC_COMP_2013_RES_2000m
#
ICE_MODIS_CONC_COMP_2013_RES_2000m_temp[ICE_MODIS_CONC_COMP_2013_RES_2000
m_temp >= 0.15] <- -1
#
ICE_MODIS_CONC_COMP_2013_RES_2000m_temp[ICE_MODIS_CONC_COMP_2013_RES_2000
m_temp > -1] <- 1
#
# # Multiply DISTANCE raster by this raster so that distance
# # within ice are negative and outside ice are positive
# DIST_ICE_MODIS_RES_2000m[["X2013_15"]] <- DIST_ICE_MODIS_RES_2000m[["X2013_15"]]
*
#   ICE_MODIS_CONC_COMP_2013_RES_2000m_temp
# remove(ICE_MODIS_CONC_COMP_2013_RES_2000m_temp)
#
#
# # Create a temporary layer to set ICE_MODIS_CONC_COMP values greater
# # than threshold (e.g, 0.15, 0.30) to negative
# ICE_MODIS_CONC_COMP_2013_RES_2000m_temp <-
ICE_MODIS_CONC_COMP_2013_RES_2000m
#
ICE_MODIS_CONC_COMP_2013_RES_2000m_temp[ICE_MODIS_CONC_COMP_2013_RES_2000
m_temp >= 0.30] <- -1
#
ICE_MODIS_CONC_COMP_2013_RES_2000m_temp[ICE_MODIS_CONC_COMP_2013_RES_2000
m_temp > -1] <- 1
#
# # Multiply DISTANCE raster by this raster so that distance
# # within ice are negative and outside ice are positive
# DIST_ICE_MODIS_RES_2000m[["X2013_30"]] <- DIST_ICE_MODIS_RES_2000m[["X2013_30"]]
*
#   ICE_MODIS_CONC_COMP_2013_RES_2000m_temp
# remove(ICE_MODIS_CONC_COMP_2013_RES_2000m_temp)
#
#
# # Create a temporary layer to set ICE_MODIS_CONC_COMP values greater
# # than threshold (e.g, 0.15, 0.30) to negative
# ICE_MODIS_CONC_COMP_2016_RES_2000m_temp <-
ICE_MODIS_CONC_COMP_2016_RES_2000m
#
ICE_MODIS_CONC_COMP_2016_RES_2000m_temp[ICE_MODIS_CONC_COMP_2016_RES_2000
m_temp >= 0.15] <- -1
```

```
#
ICE_MODIS_CONC_COMP_2016_RES_2000m_temp[ICE_MODIS_CONC_COMP_2016_RES_2000
m_temp > -1] <- 1
#
# # Multiply DISTANCE raster by this raster so that distance
# # within ice are negative and outside ice are positive
# DIST_ICE_MODIS_RES_2000m[["X2016_15"]] <- DIST_ICE_MODIS_RES_2000m[["X2016_15"]]
*
#   ICE_MODIS_CONC_COMP_2016_RES_2000m_temp
# remove(ICE_MODIS_CONC_COMP_2016_RES_2000m_temp)
#
#
# # Create a temporary layer to set ICE_MODIS_CONC_COMP values greater
# # than threshold (e.g, 0.15, 0.30) to negative
# ICE_MODIS_CONC_COMP_2016_RES_2000m_temp <-
ICE_MODIS_CONC_COMP_2016_RES_2000m
#
ICE_MODIS_CONC_COMP_2016_RES_2000m_temp[ICE_MODIS_CONC_COMP_2016_RES_2000
m_temp >= 0.30] <- -1
#
ICE_MODIS_CONC_COMP_2016_RES_2000m_temp[ICE_MODIS_CONC_COMP_2016_RES_2000
m_temp > -1] <- 1
#
# # Multiply DISTANCE raster by this raster so that distance
# # within ice are negative and outside ice are positive
# DIST_ICE_MODIS_RES_2000m[["X2016_30"]] <- DIST_ICE_MODIS_RES_2000m[["X2016_30"]]
*
#   ICE_MODIS_CONC_COMP_2016_RES_2000m_temp
# remove(ICE_MODIS_CONC_COMP_2016_RES_2000m_temp)
#
# # Write external .tif raster copy of DIST_ICE_MODIS_RES_2000m
# raster::writeRaster(DIST_ICE_MODIS_RES_2000m,
#              filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                        "1_2016_Antarctic Whale Tracking/Data/Ice
Data/DIST_ICE_MODIS_RES_2000m/",
#                        "DIST_ICE_MODIS_RES_2000m.tif",sep=""),
#              format = "GTiff",bylayer = T,suffix = "names")

# Reconstitute DIST_ICE_MODIS_RES_2000m from individual rasters

DIST_ICE_MODIS_RES_2000m<- raster::stack()

for(i in list.files(paste("/Users/trevorjoyce/Grad School/Research/",
```

```
                    "1_2016_Antarctic Whale Tracking/Data/Ice
Data/DIST_ICE_MODIS_RES_2000m/",sep=""))){


DIST_ICE_MODIS_RES_2000m_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce
/Grad School/Research/",
                                             "1_2016_Antarctic Whale Tracking/Data/Ice
Data/DIST_ICE_MODIS_RES_2000m/",i,sep="")))

  names(DIST_ICE_MODIS_RES_2000m_temp) <- substr(i,26,33)

  DIST_ICE_MODIS_RES_2000m <-
raster::stack(DIST_ICE_MODIS_RES_2000m,DIST_ICE_MODIS_RES_2000m_temp)

}

remove(DIST_ICE_MODIS_RES_2000m_temp,i)

####### BATHY_IBCSO_POLAR #########

BATHY_IBCSO_POLAR=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Grad
School/Research/",
                                          "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",

"IBCSO_v1_is_PS71_500m_tif/","ibcso_v1_is_Clipped_Polar.tif",sep = "")))
raster::projection(BATHY_IBCSO_POLAR) <- sp::CRS("+init=epsg:3031")

#remove land by setting raster values > 0 to NA
BATHY_IBCSO_POLAR[BATHY_IBCSO_POLAR>0] <- NA

####### BATHY_IBCSO_POLAR_MASK #########

BATHY_IBCSO_POLAR_MASK <- BATHY_IBCSO_POLAR

BATHY_IBCSO_POLAR_MASK[BATHY_IBCSO_POLAR_MASK<=0] <- 1

####### SLOPE_IBCSO_POLAR #########

SLOPE_IBCSO_POLAR=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Grad
School/Research/",
                                          "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",

"IBCSO_v1_is_PS71_500m_tif/","Slope_ibsco_v1_PolarStereo_Clipped.tif",sep = "")))
raster::projection(SLOPE_IBCSO_POLAR) <- sp::CRS("+init=epsg:3031")
```

```
#remove land by multiplying by BATHY_IBCSO_POLAR_MASK

SLOPE_IBCSO_POLAR <- SLOPE_IBCSO_POLAR*BATHY_IBCSO_POLAR_MASK

####### PROFILE_IBCSO_POLAR #########

PROFILE_IBCSO_POLAR=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Grad School/Research/",
                                    "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",

"IBCSO_v1_is_PS71_500m_tif/","Profile_Curvature_ibsco_v1_PolarStereo_Clipped.tif",sep = "")))
raster::projection(PROFILE_IBCSO_POLAR) <- sp::CRS("+init=epsg:3031")

#remove land by multiplying by BATHY_IBCSO_POLAR_MASK
PROFILE_IBCSO_POLAR <- PROFILE_IBCSO_POLAR*BATHY_IBCSO_POLAR_MASK

####### BATHY_IBCSO_RES_2000m #########

BATHY_IBCSO_RES_2000m <- raster::projectRaster(from = BATHY_IBCSO_POLAR,
                              to = ICE_MODIS_LAND_MASK_RES_2000m,
                              crs = CRS("+init=epsg:4326"),
                              method = "bilinear")

# Write external .tif raster copy of BATHY_IBCSO_RES_2000m
raster::writeRaster(BATHY_IBCSO_RES_2000m,
            filename=paste("/Users/trevorjoyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                    "BATHY_IBCSO_RES_2000m.tif",sep=""),
            format = "GTiff", overwrite=TRUE)

# Reconstitute BATHY_IBCSO_RES_2000m from saved raster
BATHY_IBCSO_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad School/Research/",
                                 "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                                 "BATHY_IBCSO_RES_2000m.tif",sep=""))

####### SLOPE_IBCSO_RES_2000m #########

SLOPE_IBCSO_RES_2000m <- raster::projectRaster(from = SLOPE_IBCSO_POLAR,
                              to = ICE_MODIS_LAND_MASK_RES_2000m,
                              crs = CRS("+init=epsg:4326"),
                              method = "bilinear")
```

```
# Write external .tif raster copy of SLOPE_IBCSO_RES_2000m
raster::writeRaster(SLOPE_IBCSO_RES_2000m,
            filename=paste("/Users/trevorjoyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                    "SLOPE_IBCSO_RES_2000m.tif",sep=""),
            format = "GTiff", overwrite=TRUE)

# Reconstitute SLOPE_IBCSO_RES_2000m from saved raster
SLOPE_IBCSO_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad School/Research/",
                            "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                            "SLOPE_IBCSO_RES_2000m.tif",sep=""))

####### PROFILE_IBCSO_RES_2000m #########

PROFILE_IBCSO_RES_2000m <- raster::projectRaster(from = PROFILE_IBCSO_POLAR,
                          to = ICE_MODIS_LAND_MASK_RES_2000m,
                          crs = CRS("+init=epsg:4326"),
                          method = "bilinear")

# Write external .tif raster copy of PROFILE_IBCSO_RES_2000m
raster::writeRaster(PROFILE_IBCSO_RES_2000m,
            filename=paste("/Users/trevorjoyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                    "PROFILE_IBCSO_RES_2000m.tif",sep=""),
            format = "GTiff", overwrite=TRUE)

# Reconstitute PROFILE_IBCSO_RES_2000m from saved raster
PROFILE_IBCSO_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                            "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                            "PROFILE_IBCSO_RES_2000m.tif",sep=""))

####### CONTINENTAL_SHELF #########


# # Create a contour line at the 450m isobath
# arcpy.gp.ContourList_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Bathy/ibcso_v1_is_Clipped_Polar.tif",
#              "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Bathy/CONTINENTAL_SHELF_450m.shp",
#              "-450")
#
```

```
CONTINENTAL_SHELF <- rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad
School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",sep =
""),
                        layer="CONTINENTAL_SHELF_450m")

####### CONTINENTAL_SHELF_OUTER #########

# # Manually dissect this contour line into inner (shelf deeps)
# # and outer (shelf margin and valleys) lines using Select and Export Data
CONTINENTAL_SHELF_OUTER <- rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad
School/Research/",
                                  "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",sep =
""),
                            layer="continental_shelf_450m_outer")
raster::projection(CONTINENTAL_SHELF_OUTER) <- sp::CRS("+init=epsg:3031")


####### CONTINENTAL_SHELF_INNER #########

# # Manually dissect this contour line into inner (shelf deeps)
# # and outer (shelf margin and valleys) lines using Select and Export Data
CONTINENTAL_SHELF_INNER <- rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad
School/Research/",
                                  "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",sep =
""),
                            layer="continental_shelf_450m_inner")
raster::projection(CONTINENTAL_SHELF_INNER) <- sp::CRS("+init=epsg:3031")


####### CONTINENTAL_SHELF_OUTER_POLYGONS #########

# Convert outer (shelf margin and valleys) contour lines into SpatialPolygons
CONTINENTAL_SHELF_OUTER_POLYGONS <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(CONTINENTAL_SHELF_OUT
ER))

# Run buffer = 0 command to deal with topology exception errors in later gSymdifference
command
CONTINENTAL_SHELF_OUTER_POLYGONS <-
rgeos::gBuffer(CONTINENTAL_SHELF_OUTER_POLYGONS,width = 0, byid = T,
                                 id = names(CONTINENTAL_SHELF_OUTER_POLYGONS))

# Convert SpatialPolygons into SpatialPolygonsDataFrame
```

```
CONTINENTAL_SHELF_OUTER_POLYGONS <-
sp::SpatialPolygonsDataFrame(CONTINENTAL_SHELF_OUTER_POLYGONS,
                                         data.frame(ID =
names(CONTINENTAL_SHELF_OUTER_POLYGONS)),
                                         match.ID = F)
# Define projection
raster::projection(CONTINENTAL_SHELF_OUTER_POLYGONS) <- sp::CRS("+init=epsg:3031")


####### CONTINENTAL_SHELF_INNER_POLYGONS #########

# Convert inner (shelf deeps) contour lines into SpatialPolygons
CONTINENTAL_SHELF_INNER_POLYGONS <-
maptools::PolySet2SpatialPolygons(maptools::SpatialLines2PolySet(CONTINENTAL_SHELF_INNE
R))

# Run buffer = 0 command to deal with topology exception errors in later gSymdifference
command
CONTINENTAL_SHELF_INNER_POLYGONS <-
rgeos::gBuffer(CONTINENTAL_SHELF_INNER_POLYGONS,width = 0, byid = T,
                               id = names(CONTINENTAL_SHELF_INNER_POLYGONS))

# Convert SpatialPolygons into SpatialPolygonsDataFrame
CONTINENTAL_SHELF_INNER_POLYGONS <-
sp::SpatialPolygonsDataFrame(CONTINENTAL_SHELF_INNER_POLYGONS,
                                         data.frame(ID =
names(CONTINENTAL_SHELF_INNER_POLYGONS)),
                                         match.ID = F)
# Define projection
raster::projection(CONTINENTAL_SHELF_INNER_POLYGONS) <- sp::CRS("+init=epsg:3031")


####### CONTINENTAL_SHELF_POLYGONS #########

# Create SpatialPolygons with holes to exclude inner contour lines (shelf deeps)
CONTINENTAL_SHELF_POLYGONS <-
rgeos::gSymdifference(CONTINENTAL_SHELF_OUTER_POLYGONS,
                               CONTINENTAL_SHELF_INNER_POLYGONS)

# Convert SpatialPolygons into SpatialPolygonsDataFrame
CONTINENTAL_SHELF_POLYGONS <-
sp::SpatialPolygonsDataFrame(CONTINENTAL_SHELF_POLYGONS,
                                      data.frame(ID = 1))
```

```
####### DIST_CONTINENTAL_SHELF_POLAR #########

# # Manually add a long tail off the end of the contour line
# # to cover the range of projected easting longitudes (avoids an error
# # where PathDistance function only calculates to the bounding box of the feature)
#
# # Calculate the path distance to the 450m isobath (i.e. including internal structure within the
shelf)
# arcpy.gp.PathDistance_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Bathy/continental_shelf_450m.shp",
#                "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Bathy/continental_shelf_450m_PathDist_POLAR.tif",
#                "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ICE_MODIS_LAND_MASK_POLAR.tif",
#                "", "", "BINARY 1 45", "", "BINARY 1 -30 30", "", "", "", "", "", "")


# Reconstitute DIST_CONTINENTAL_SHELF_POLAR from saved raster
DIST_CONTINENTAL_SHELF_POLAR=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/
Grad School/Research/",
                        "1_2016_Antarctic Whale Tracking/Data/Bathymetric
Data/",
                        "continental_shelf_450m_PathDistance.tif",sep = "")))
raster::projection(DIST_CONTINENTAL_SHELF_POLAR) <- sp::CRS("+init=epsg:3031")


####### DIST_CONTINENTAL_SHELF_OUTER_POLAR #########

# # Manually add a long tail off the end of the contour line
# # to cover the range of projected easting longitudes (avoids an error
# # where PathDistance function only calculates to the bounding box of the feature)

# # Calculate the path distance to the 450m isobath using the outer shelf margin and valleys
definition
# # (i.e. ignoring internal structure within the shelf)
# arcpy.gp.PathDistance_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Bathy/continental_shelf_450m_outer.shp",
#                "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Bathy/continental_shelf_450m_outer_PathDist_POLAR.tif",
#                "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ICE_MODIS_LAND_MASK_POLAR.tif",
```

```
#                    "", "", "BINARY 1 45", "", "BINARY 1 -30 30", "", "", "", "", "", "")
#

# Reconstitute DIST_CONTINENTAL_SHELF_OUTER_POLAR from saved raster
DIST_CONTINENTAL_SHELF_OUTER_POLAR=raster::raster(rgdal::readGDAL(paste("/Users/trevo
rjoyce/Grad School/Research/",
                                      "1_2016_Antarctic Whale Tracking/Data/Bathymetric
Data/",
                                      "continental_shelf_450m_outer_PathDistance.tif",sep =
"")))
raster::projection(DIST_CONTINENTAL_SHELF_OUTER_POLAR) <- sp::CRS("+init=epsg:3031")



###### DIST_CONTINENTAL_SHELF_RES_2000m #########

DIST_CONTINENTAL_SHELF_RES_2000m <- raster::projectRaster(from =
DIST_CONTINENTAL_SHELF_POLAR,
                                  to = ICE_MODIS_LAND_MASK_RES_2000m,
                                  crs = sp::CRS("+init=epsg:4326"),
                                  method = "bilinear")

# Convert CONTINENTAL_SHELF_OUTER_POLYGONS into a raster that will be used to
# multiply DIST_CONTINENTAL_SHELF values inside CONTINENTAL_SHELF_OUTER_POLYGONS
# by negative numbers and exterior values by positive numbers
CONTINENTAL_SHELF_POLYGONS_RES_2000m_temp <-
raster::rasterize(sp::spTransform(CONTINENTAL_SHELF_POLYGONS,
                                          sp::CRS("+init=epsg:4326")),
                                    ICE_MODIS_LAND_MASK_RES_2000m)

# Values inside CONTINENTAL_SHELF_POLYGONS = -1, outside = 1
CONTINENTAL_SHELF_POLYGONS_RES_2000m_temp[CONTINENTAL_SHELF_POLYGONS_RES_2
000m_temp > 0] <- -1
CONTINENTAL_SHELF_POLYGONS_RES_2000m_temp[is.na(CONTINENTAL_SHELF_POLYGONS_
RES_2000m_temp)] <- 1

# Multiply DISTANCE raster by this raster so that distance
# within CONTINENTAL_SHELF are negative and outside CONTINENTAL_SHELF are positive
DIST_CONTINENTAL_SHELF_RES_2000m <- DIST_CONTINENTAL_SHELF_RES_2000m *
  CONTINENTAL_SHELF_POLYGONS_RES_2000m_temp
remove(CONTINENTAL_SHELF_POLYGONS_RES_2000m_temp)

# Write external .tif raster copy of DIST_CONTINENTAL_SHELF_RES_2000m
```

```
raster::writeRaster(DIST_CONTINENTAL_SHELF_RES_2000m,
          filename=paste("/Users/trevorjoyce/Grad School/Research/",
                  "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                  "DIST_CONTINENTAL_SHELF_RES_2000m.tif",sep=""),
          format = "GTiff", overwrite=TRUE)

# Reconstitute DIST_CONTINENTAL_SHELF_RES_2000m from saved raster
DIST_CONTINENTAL_SHELF_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                             "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                             "DIST_CONTINENTAL_SHELF_RES_2000m.tif",sep=""))

###### DIST_CONTINENTAL_SHELF_OUTER_RES_2000m #########

DIST_CONTINENTAL_SHELF_OUTER_RES_2000m <- raster::projectRaster(from =
DIST_CONTINENTAL_SHELF_OUTER_POLAR,
                              to = ICE_MODIS_LAND_MASK_RES_2000m,
                              crs = sp::CRS("+init=epsg:4326"),
                              method = "bilinear")

# Convert CONTINENTAL_SHELF_OUTER_OUTER_POLYGONS into a raster that will be used to
# multiply DIST_CONTINENTAL_SHELF values inside
CONTINENTAL_SHELF_OUTER_OUTER_POLYGONS
# by negative numbers and exterior values by positive numbers
CONTINENTAL_SHELF_OUTER_POLYGONS_RES_2000m_temp <-
raster::rasterize(sp::spTransform(CONTINENTAL_SHELF_OUTER_POLYGONS,
                                             sp::CRS("+init=epsg:4326")),
                                 ICE_MODIS_LAND_MASK_RES_2000m)

# Values inside CONTINENTAL_SHELF_OUTER_POLYGONS = -1, outside = 1
CONTINENTAL_SHELF_OUTER_POLYGONS_RES_2000m_temp[CONTINENTAL_SHELF_OUTER_P
OLYGONS_RES_2000m_temp > 0] <- -1
CONTINENTAL_SHELF_OUTER_POLYGONS_RES_2000m_temp[is.na(CONTINENTAL_SHELF_OUT
ER_POLYGONS_RES_2000m_temp)] <- 1

# Multiply DISTANCE raster by this raster so that distance
# within CONTINENTAL_SHELF are negative and outside CONTINENTAL_SHELF are positive
DIST_CONTINENTAL_SHELF_OUTER_RES_2000m <-
DIST_CONTINENTAL_SHELF_OUTER_RES_2000m *
  CONTINENTAL_SHELF_OUTER_POLYGONS_RES_2000m_temp
remove(CONTINENTAL_SHELF_OUTER_POLYGONS_RES_2000m_temp)

# Write external .tif raster copy of DIST_CONTINENTAL_SHELF_OUTER_RES_2000m
raster::writeRaster(DIST_CONTINENTAL_SHELF_OUTER_RES_2000m,
```

```
            filename=paste("/Users/trevorjoyce/Grad School/Research/",
                    "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                    "DIST_CONTINENTAL_SHELF_OUTER_RES_2000m.tif",sep=""),
            format = "GTiff", overwrite=TRUE)

# Reconstitute DIST_CONTINENTAL_SHELF_OUTER_RES_2000m from saved raster
DIST_CONTINENTAL_SHELF_OUTER_RES_2000m =
raster::raster(paste("/Users/trevorjoyce/Grad School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Bathymetric Data/",
                              "DIST_CONTINENTAL_SHELF_OUTER_RES_2000m.tif",sep=""))

####### DIST_LAND_POLAR #########

# arcpy.Clip_management(in_raster="Coastline_high_res_polygon_P",
#             rectangle="-3147000 432500.000000002 -1640500 1981000",
#             out_raster="C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Coastline/Coastline_high_res_polygon/Coastline_high_res_polygon_Clipped.tif",
#             in_template_dataset="ibcso_v1_is_Clipped_Polar.tif",
#             nodata_value="0", clipping_geometry="NONE",
#             maintain_clipping_extent="NO_MAINTAIN_EXTENT")
#
# arcpy.gp.PathDistance_sa("C:/Users/trevor.joyce/Documents/Grad
School/Research/1_2016_Antarctic Whale Tracking/Data/GIS
Data/Coastline/Coastline_high_res_polygon/Coastline_high_res_polygon_Clipped.tif",
#              "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS
Data/Coastline/Coastline_high_res_polygon/Coastline_high_res_polygon_Clipped_PathDist_PO
LAR.tif",
#              "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic
Whale Tracking/Data/GIS Data/Ice/ICE_MODIS_LAND_MASK_POLAR.tif",
#              "", "", "BINARY 1 45", "", "BINARY 1 -30 30", "", "", "", "", "", "")

DIST_LAND_POLAR=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Grad
School/Research/",
                             "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                             "Coastline_high_res_polygon_Clipped_PathDist_POLAR.tif",sep =
"")))
raster::projection(DIST_LAND_POLAR) <- sp::CRS("+init=epsg:3031")


###### DIST_LAND_RES_2000m #########

# DIST_LAND_RES_2000m <- raster::projectRaster(from = DIST_LAND_POLAR,
```

```
#                               to = ICE_MODIS_LAND_MASK_RES_2000m,
#                               crs = CRS("+init=epsg:4326"),
#                               method = "bilinear")
#
# # Write external .tif raster copy of DIST_LAND_RES_2000m
# raster::writeRaster(DIST_LAND_RES_2000m,
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                         "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#                         "DIST_LAND_RES_2000m.tif",sep=""),
#               format = "GTiff", overwrite=TRUE)

# Reconstitute DIST_LAND_RES_2000m from saved raster
DIST_LAND_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                              "DIST_LAND_RES_2000m.tif",sep=""))


#######  LAND_MASK_RES_2000m_BUFFER_100km  ########
# # Create a 100km buffer in OCEAN cells as defined by ICE_MODIS_LAND_MASK_RES_2000m
#
# # Convert OCEAN into NAs over water and 1s over LAND
# LAND_MASK_RES_2000m_BUFFER_100km <- ICE_MODIS_LAND_MASK_RES_2000m
# LAND_MASK_RES_2000m_BUFFER_100km[!is.na(LAND_MASK_RES_2000m_BUFFER_100km)]
<- -1
# LAND_MASK_RES_2000m_BUFFER_100km[is.na(LAND_MASK_RES_2000m_BUFFER_100km)]
<- 1
# LAND_MASK_RES_2000m_BUFFER_100km[LAND_MASK_RES_2000m_BUFFER_100km<0] <-
NA
#
# #Calculate euclidean distances using built-in raster function
# LAND_MASK_RES_2000m_BUFFER_100km <-
raster::distance(LAND_MASK_RES_2000m_BUFFER_100km)
#
# #Remove distance cell values >100km
# LAND_MASK_RES_2000m_BUFFER_100km[LAND_MASK_RES_2000m_BUFFER_100km >
100*1000] <- NA
# LAND_MASK_RES_2000m_BUFFER_100km[!is.na(LAND_MASK_RES_2000m_BUFFER_100km)]
<- 1
#
# #Remove LAND cell values
# LAND_MASK_RES_2000m_BUFFER_100km <- LAND_MASK_RES_2000m_BUFFER_100km *
ICE_MODIS_LAND_MASK_RES_2000m
#
#
```

```
# # Write external .tif raster copy of LAND_MASK_RES_2000m_BUFFER_100km
# raster::writeRaster(LAND_MASK_RES_2000m_BUFFER_100km,
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                          "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#                          "LAND_MASK_RES_2000m_BUFFER_100km.tif",sep=""),
#               format = "GTiff", overwrite=TRUE)

# Reconstitute LAND_MASK_RES_2000m_BUFFER_100km from saved raster
LAND_MASK_RES_2000m_BUFFER_100km = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                              "LAND_MASK_RES_2000m_BUFFER_100km.tif",sep=""))

####### COAST_MED_WAP #########

#Import a COAST outline file derived from the British Antarctic Survey Geodata Portal -
Antarctic Digital Database server output
#dowloaded as a shapefile from
(http://add.antarctica.ac.uk/repository/entry/show?entryid=f477219b-9121-44d6-afa6-
d8552762dc45)
#that has been projected from its native Antarctic Polar Stereographic (ESPG: 3031) projected
coordinate system
#into GCS_WGS84 (EPSG: 4326) in ESRI ArcGIS v10.3
#
# COAST_MED_WAP=rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad
School/Research/1_2016_Antarctic Whale Tracking/",
#                          "Data/Coastline/Coastline_medium_res_polygon",sep=""),
#                   layer="Coastline_medium_res_polygon_GCS_WGS84")
#
#
# # Crop polygons to the region of interest
# COAST_MED_WAP <- raster::crop(COAST_MED_WAP,raster::extent(c(xmin = -78,
#                                      xmax = -52,
#                                      ymin = -71,
#                                      ymax = -61)))
#
# # Unify polygons to remove boundaries between Iceshelves and Land
# COAST_MED_WAP = rgeos::gUnaryUnion(COAST_MED_WAP)
#
# saveRDS(object = COAST_MED_WAP,
#       file = paste("/Users/trevorjoyce/Grad School/Research/",
#               "1_2016_Antarctic Whale Tracking/Data/Output Data/",
#               "COAST_MED_WAP.rds",sep=""))
#
```

```
# saveRDS(object = COAST_MED_WAP,
#       file = paste("/Users/trevorjoyce/Grad School/Research/",
#               "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#               "COAST_MED_WAP.rds",sep=""))
#
# # Import RDS files into local R environment in Ahi
# COAST_MED_WAP <- readRDS(file =
paste("/home/tjoyce/1_2016_Antarctic_Whale_Tracking/Data/Output_Data/",
#                          "COAST_MED_WAP.rds",sep=""))
#
# # Import RDS files into local R environment in Ahi
# COAST_MED_WAP <- readRDS(file = paste("/Users/trevorjoyce/Grad School/Research/",
#                          "1_2016_Antarctic Whale Tracking/Data/Output Data/",
#                          "COAST_MED_WAP.rds",sep=""))

# Import RDS files into local R environment in Ahi
COAST_MED_WAP <- readRDS(file = paste("/Users/trevorjoyce/Grad School/Research/",
                         "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                         "COAST_MED_WAP.rds",sep=""))

######   DIST_DIR_SL_CALC   ######

#Function to convert each row of a section of DIST_DIR cell coordinates into a
#SpatialLinesDataFrame

DIST_DIR_SL_CALC <- function(X,DATA){
 DATA_DF <- rbind(DATA[[X]][,c("lon","lat")],
          geosphere::destPointRhumb(p = DATA[[X]][,c("lon","lat")],
                         b = DATA[[X]][,c("BEARING")],
                         d = DATA[[X]][,c("DISTANCE")]))
 sp::coordinates(DATA_DF) <- ~ lon + lat
 DATA_SL <- as(DATA_DF,"SpatialLines")
 DATA_SLDF <- sp::SpatialLinesDataFrame(DATA_SL,
                         data.frame(CELL = X,
                              LONG = DATA[[X]][,c("lon")],
                              LAT = DATA[[X]][,c("lat")],
                              BEARING = DATA[[X]][,c("BEARING")],
                              stringsAsFactors = F),
                         match.ID = F)
 return(DATA_SLDF)
}


###### MIN_DIST_DIR_CALC ######
```

```
# Function that calculates the point intersections (if they exist) between the lines defined above
# and a coastline geometry (COAST_MED_WAP), then calculates the minimum distance
# between these intersections and each DIST_DIR point

MIN_DIST_DIR_CALC <- function(X, LINES, COAST){

  # find intersections between the line and a coastline geometry (COAST)
  INTERSECTIONS_POINTS = rgeos::gIntersection(COAST,LINES[LINES$CELL == X,])

  # return NA if there are no intersections
  if(is.null(INTERSECTIONS_POINTS)){

    LINES_DF <- as.data.frame(LINES[LINES$CELL == X,])
    LINES_DF[,c("MIN_DIST_INT","LONG_INT","LAT_INT")] <- NA
    return(LINES_DF)
  }

  # if there are intersections
  if(!is.null(INTERSECTIONS_POINTS)){

    INTERSECTIONS_POINTS = as(INTERSECTIONS_POINTS,"SpatialPoints")
    INTERSECTIONS_POINTS = sp::SpatialPointsDataFrame(INTERSECTIONS_POINTS,
                                data.frame(LONG_INT =
sp::coordinates(INTERSECTIONS_POINTS)[,1],
                                      LAT_INT = sp::coordinates(INTERSECTIONS_POINTS)[,2]))

    #calculate the geodesic distance between origin point and each intersection
    INTERSECTIONS_POINTS$DIST_INT = geosphere::distGeo(p1 =
as.data.frame(LINES[LINES$CELL == X,c("LONG","LAT")]),
                                p2 = INTERSECTIONS_POINTS)

    LINES_DF = as.data.frame(LINES[LINES$CELL == X,])

    LINES_DF$MIN_DIST_INT <- min(INTERSECTIONS_POINTS$DIST_INT)
    LINES_DF[,c("LONG_INT","LAT_INT")] <-
as.data.frame(INTERSECTIONS_POINTS[which.min(INTERSECTIONS_POINTS$DIST_INT),])[,c("LO
NG_INT","LAT_INT")]

    return(LINES_DF)
  }

}
```

```
####### DIST_DIR ########

#Change from Raster to SpatialPointsDataFrame to dataframe
DIST_DIR <-
as.data.frame(as(LAND_MASK_RES_2000m_BUFFER_100km,"SpatialPointsDataFrame"))

#Add CELL numbers for later link to Raster
DIST_DIR$CELL <- raster::cellFromXY(LAND_MASK_RES_2000m_BUFFER_100km,
                    xy= DIST_DIR[,c("x","y")])

DIST_DIR[,c("lon","lat")] <- DIST_DIR[,c("x","y")]


###### MIN_DIST_DIR_LIST ######

# #list to house MIN_DIST calculations for each direction
# MIN_DIST_DIR_LIST <- list()
#
# # loop through each 36 directions
# for(j in c(0,30,60,90,120,150,180,210,240,270,300,330,
#        10,40,70,100,130,160,190,220,250,280,310,340,
#        20,50,80,110,140,170,200,230,260,290,320,350)){
#
#   # data.frame to house pieces of MIN_DIST_DIR calculated piecemeal
#   MIN_DIST_DIR_DF_temp <- data.frame(stringsAsFactors = F)
#
#   # loop through the rows of DIST_DIR grabbing manageable chunks of data
#   for(i in seq(1,nrow(DIST_DIR),by = 2000)){
#
#    #handle end data range values that are not multiples of nrow(DIST_DIR)
#    INDEX_temp <- ifelse(i+1999 < nrow(DIST_DIR), i+1999, nrow(DIST_DIR))
#
#    # Print information on DIR and ROWS to track progress
#    print(paste("DIR:",j,", ROWS:",i,"-",INDEX_temp,", TIME:", Sys.time()))
#
#    # Grab a manageable chunk of cell coordinates from DIST_DIR to
#    # avoid the system choking in creating SpatialLinesDataFrame
#    DIST_DIR_temp <- DIST_DIR[i:INDEX_temp,c("CELL","lon","lat")]
#    DIST_DIR_temp$BEARING <- j
#    DIST_DIR_temp$DISTANCE <- 1000*1000
#
#    # Change data.frame into a list consisting of data.frames
#    # for each row in DIST_DIR_temp
```

```
#    DIST_DIR_SL_temp <- split(x = DIST_DIR_temp,
#                    f = as.factor(DIST_DIR_temp$CELL))
#
#    # Calculate initial system time for monitoring process
#    PROCESSING_TIME_temp <- proc.time()
#
#    # Run the DIST_DIR_SL_CALC function to create a list of SpatialLinesDataFrames
#    # for each row in DIST_DIR_SL_temp
#    DIST_DIR_SL_temp <- parallel::mclapply(X = as.character(DIST_DIR_temp$CELL),
#                              DATA = DIST_DIR_SL_temp,
#                              FUN = DIST_DIR_SL_CALC,
#                              mc.cores = 12)
#
#    # Compile this list into one overall SpatialLinesDataFrames
#    DIST_DIR_SL_temp <- do.call("rbind",DIST_DIR_SL_temp)
#
#    # Change the IDs of DIST_DIR_SL_temp to the cell numbers from STPP_COVAR
#    DIST_DIR_SL_temp <- sp::spChFIDs(DIST_DIR_SL_temp,as.character(DIST_DIR_temp$CELL))
#
#    # Print information on PROCESSING_TIME
#    print(paste("DIST_DIR_SL calculation time:"))
#    print(proc.time() - PROCESSING_TIME_temp)
#
#    # Calculate initial system time for monitoring process
#    PROCESSING_TIME_temp <- proc.time()
#
#    # Run the MIN_DIST_DIR_CALC function to create a data frame with
#    # MIN_DIST_DIR calculated for each cell represented in DIST_DIR_SL_temp
#    MIN_DIST_DIR_temp <- parallel::mclapply(X = DIST_DIR_SL_temp$CELL,
#                              LINES = DIST_DIR_SL_temp,
#                              COAST = COAST_MED_WAP,
#                              FUN = MIN_DIST_DIR_CALC,
#                              mc.cores = 12)
#
#    # Compile this list into one overall SpatialLinesDataFrames
#    MIN_DIST_DIR_temp <- do.call("rbind",MIN_DIST_DIR_temp)
#
#    # Print information on PROCESSING_TIME
#    print(paste("MIN_DIST calculation time:"))
#    print(proc.time() - PROCESSING_TIME_temp)
#
#    # Add rows from chunk of cell coordinates from DIST_DIR
#    # to overall data.frame MIN_DIST_DIR_DF_temp
#    MIN_DIST_DIR_DF_temp <- rbind(MIN_DIST_DIR_DF_temp,
```

```
#                          MIN_DIST_DIR_temp)
#
#    #Clean-up
#    remove(MIN_DIST_DIR_temp, PROCESSING_TIME_temp,
#         DIST_DIR_temp, DIST_DIR_SL_temp, INDEX_temp); gc()
#  }
#
#  #Add data.frame of MIN_DIST_DIR calculated for each cell in DIST_DIR to the overall list
#  MIN_DIST_DIR_LIST[[as.character(j)]] <- MIN_DIST_DIR_DF_temp
#
#  # Save kernel intensity values associated with each simulated TRACK to a separate .csv
#  # (in case loop crashes or system times out)
#  write.csv(MIN_DIST_DIR_DF_temp,
#          paste("/home/tjoyce/1_2016_Antarctic_Whale_Tracking/Data/Output_Data/",
#              "MIN_DIST_DIR_",j,".csv",sep=""))
#
#  # # Save kernel intensity values associated with each simulated TRACK to a separate .csv
#  # # (in case loop crashes or system times out)
#  # write.csv(MIN_DIST_DIR_DF_temp,
#  #         paste("/Users/trevorjoyce/Grad School/Research/",
#  #             "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#  #             "MIN_DIST_DIR_",i,".csv",sep=""))
#
# }
#
#
# #Clean-up
# remove(MIN_DIST_DIR_DF_temp,i,j); gc()
#
# # Save kernel intensity values to a single overall .rds file
# saveRDS(object = MIN_DIST_DIR_LIST,
#       file = paste("/home/tjoyce/1_2016_Antarctic_Whale_Tracking/Data/Output_Data/",
#               "MIN_DIST_DIR_LIST.rds",sep=""))
#
# # Save kernel intensity values to a single overall .rds file
# saveRDS(object = MIN_DIST_DIR_LIST,
#       file = paste("/Users/trevorjoyce/Grad School/Research/",
#               "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#               "MIN_DIST_DIR_LIST.rds",sep=""))

# Import RDS files into local R environment in Ahi
MIN_DIST_DIR_LIST <- readRDS(file = paste("/Users/trevorjoyce/Grad School/Research/",
                          "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                          "MIN_DIST_DIR_LIST.rds",sep=""))
```

```
###### MIN_DIST_DIR #####

MIN_DIST_DIR <- raster::brick(ICE_MODIS_LAND_MASK_RES_2000m)

for(i in names(MIN_DIST_DIR_LIST)){
  MIN_DIST_DIR[[paste("DIR_",i, sep = "")]] <- ICE_MODIS_LAND_MASK_RES_2000m
  MIN_DIST_DIR[[paste("DIR_",i, sep = "")]][] <- NA
  MIN_DIST_DIR[[paste("DIR_",i, sep = "")]][as.numeric(MIN_DIST_DIR_LIST[[i]]$CELL)] <-
MIN_DIST_DIR_LIST[[i]]$MIN_DIST_INT
}

MIN_DIST_DIR <- raster::dropLayer(x = MIN_DIST_DIR,i = which(names(MIN_DIST_DIR) ==
"ICE_MODIS_LAND_MASK_RES_2000m"))

for(i in names(MIN_DIST_DIR)){
  MIN_DIST_DIR[[i]][MIN_DIST_DIR[[i]] >= 80*1000] <- 80*1000
  MIN_DIST_DIR[[i]][is.na(MIN_DIST_DIR[[i]])] <- 80*1000
  MIN_DIST_DIR[[i]] <- MIN_DIST_DIR[[i]] * ICE_MODIS_LAND_MASK_RES_2000m
}

#### MIN_DIST_DIR_MEAN_RES_2000m #####

# MIN_DIST_DIR_MEAN_RES_2000m <- raster::calc(MIN_DIST_DIR, mean, na.rm =
T)/(80*1000)
#
# # Write external .tif raster copy of MIN_DIST_DIR_MEAN_RES_2000m
# raster::writeRaster(MIN_DIST_DIR_MEAN_RES_2000m,
#             filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                     "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#                     "MIN_DIST_DIR_MEAN_RES_2000m.tif",sep=""),
#             format = "GTiff", overwrite=TRUE)

# Reconstitute MIN_DIST_DIR_MEAN_RES_2000m from saved raster
MIN_DIST_DIR_MEAN_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                        "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                        "MIN_DIST_DIR_MEAN_RES_2000m.tif",sep=""))


#### MIN_DIST_DIR_MED_RES_2000m #####
```

```
# MIN_DIST_DIR_MED_RES_2000m <- raster::calc(MIN_DIST_DIR, median, na.rm =
T)/(80*1000)
#
#
# # Write external .tif raster copy of MIN_DIST_DIR_MED_RES_2000m
# raster::writeRaster(MIN_DIST_DIR_MED_RES_2000m,
#                 filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                            "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#                            "MIN_DIST_DIR_MED_RES_2000m.tif",sep=""),
#                 format = "GTiff", overwrite=TRUE)

# Reconstitute MIN_DIST_DIR_MED_RES_2000m from saved raster
MIN_DIST_DIR_MED_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                                  "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                                  "MIN_DIST_DIR_MED_RES_2000m.tif",sep=""))



###### COAST_CLASS #######

COAST_CLASS <- rgdal::readOGR(dsn=paste("/Users/trevorjoyce/Grad School/Research/",
                            "1_2016_Antarctic Whale Tracking/Data/Coastline",sep=""),
                 layer="Coastline_Feature_Classification")


###### COAST_CLASS_RES_2000m #######
#
# COAST_CLASS_RES_2000m <- raster::rasterize(x = COAST_CLASS,
#                               y = ICE_MODIS_LAND_MASK_RES_2000m,
#                               field = "CLASS")
#
# # Write external .tif raster copy of COAST_CLASS_RES_2000m
# raster::writeRaster(COAST_CLASS_RES_2000m,
#                 filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                            "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#                            "COAST_CLASS_RES_2000m.tif",sep=""),
#                 format = "GTiff", overwrite=TRUE)

# Reconstitute COAST_CLASS_RES_2000m from saved raster
COAST_CLASS_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad School/Research/",
                                  "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                                  "COAST_CLASS_RES_2000m.tif",sep=""))
```

```
###### COAST_CLASS_RES_2000m #######

# COAST_CLASS_GEN_RES_2000m <- raster::rasterize(x = COAST_CLASS,
#                               y = ICE_MODIS_LAND_MASK_RES_2000m,
#                               field = "CLASS_GEN")
#
# # Write external .tif raster copy of COAST_CLASS_GEN_RES_2000m
# raster::writeRaster(COAST_CLASS_GEN_RES_2000m,
#                filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                          "1_2016_Antarctic Whale Tracking/Data/Coastline/",
#                          "COAST_CLASS_GEN_RES_2000m.tif",sep=""),
#                format = "GTiff", overwrite=TRUE)

# Reconstitute COAST_CLASS_GEN_RES_2000m from saved raster
COAST_CLASS_GEN_RES_2000m = raster::raster(paste("/Users/trevorjoyce/Grad
School/Research/",
                                 "1_2016_Antarctic Whale Tracking/Data/Coastline/",
                                 "COAST_CLASS_GEN_RES_2000m.tif",sep=""))


#### CHL ####

# CHL <- raster::stack()
#
# for(i in list.files(paste("/Users/trevorjoyce/Grad School/Research/",
#                   "1_2016_Antarctic Whale Tracking/Data/Chl Data",sep=""))){
#
#   #Import NetCDF file as a raster
#   CHL_temp=raster::raster(paste("/Users/trevorjoyce/Grad School/Research/",
#                       "1_2016_Antarctic Whale Tracking/Data/Chl Data/",
#                       i,sep=""), varname='chlor_a')
#
#   #Assign a name based on year and platform (e.g. X2009_A for Aqua MODIS winter 2008-
2009)
#   names(CHL_temp) <- paste("X",substr(i,9,12),"_",substr(i,1,1),sep = "")
#
#   #Add to CHL stack
#   CHL <- raster::stack(CHL,CHL_temp)
# }
#
# # Crop chlorophyll (CHL) data to the extent of study area (WAP MODIS analysis area)
# CHL <- raster::crop(CHL,raster::extent(c(xmin = -78,xmax = -52,ymin = -71,ymax = -61)))
#
#
```

```
# #Loop to combine Aqua MODIS and Terra MODIS rasters into a single raster per year
# for(i in names(CHL)[substr(names(CHL),7,7) == "A"]){
#
#   #Average Aqua MODIS and Terra MODIS seasonal climatologies
#   CHL_temp <- calc(x = CHL[[which(substr(names(CHL),2,5)%in%substr(i,2,5))]],
#                fun = mean, na.rm = T)
#
#   #Rename single raster per year
#   names(CHL_temp) <- substr(i,1,5)
#
#   #Add these rasters to CHL stack
#   CHL <- raster::stack(CHL,CHL_temp)
# }
#
# #Clean up temp variables
# remove(CHL_temp)
#
# #Remove platform specific rasters from CHL (leaving mean rasters)
#
# CHL <- CHL[[which(nchar(names(CHL))<=5)]]
#
# # Save to individual raster files in common geotiff format (more secure and transferable than
.rds)
# raster::writeRaster(CHL,
#                filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                        "1_2016_Antarctic Whale Tracking/Data/Chl Data/CHL_Cropped_Mean",
#                        "CHL.tif",sep=""),
#                format = "GTiff",bylayer = T,suffix = "names")

# Reconstitute CHL from saved individual rasters

CHL<- raster::stack()

for(i in list.files(paste("/Users/trevorjoyce/Grad School/Research/",
                "1_2016_Antarctic Whale Tracking/Data/Chl
Data/CHL_Cropped_Mean/",sep=""))){

 CHL_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Grad School/Research/",
                          "1_2016_Antarctic Whale Tracking/Data/Chl
Data/CHL_Cropped_Mean/",i,sep="")))

 names(CHL_temp) <- substr(i,5,9)

 CHL <- raster::stack(CHL,CHL_temp)
```

```
}

remove(CHL_temp)


###### CHL_IDW_INTERP #############

# # Convert each CHL raster into a point object (that can be interpolated to fill unsampled areas)
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009.tif",
#                        out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2009_points.shp",
#                        raster_field="Value")
#
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2010.tif",
#                        out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2010_points.shp",
#                        raster_field="Value")
#
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2011.tif",
#                        out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2011_points.shp",
#                        raster_field="Value")
#
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2012.tif",
#                        out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2012_points.shp",
#                        raster_field="Value")
#
```

```
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2013.tif",
#                    out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2013_points.shp",
#                    raster_field="Value")
#
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2014.tif",
#                    out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2014_points.shp",
#                    raster_field="Value")
#
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2015.tif",
#                    out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2015_points.shp",
#                    raster_field="Value")
#
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2016.tif",
#                    out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2016_points.shp",
#                    raster_field="Value")
#
# arcpy.RasterToPoint_conversion(in_raster="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2017.tif",
#                    out_point_features="C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2017_points.shp",
#                    raster_field="Value")
#
#
# # Use inverse distance weighting algorithm to fill areas
# # not covered because of coarse global land mask (i.e., bays)
```

```
# # Fixed interpolation range set to 0.35 degrees, and power set to 3 (to minimize influence of distant points)
# # First run extent to match CHL_X2009.tif under
# # Environments > Processing Extent > Extent > "Same as layer CHL_X2009.tif"
# # Thereafter this same environment is referenced in generating other years
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2009_points.shp",
#           "GRID_CODE",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif",
#           "C:\Users\trevor.joyce\Documents\Grad School\Research\1_2016_Antarctic Whale Tracking\Data\GIS Data\Chlorophyll\CHL_Cropped_Mean\CHL_X2009.tif",
#           "3",
#           "FIXED 0.35",
#           "")
#
# arcpy.env.workspace = "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif"
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2010_points.shp",
#           "GRID_CODE",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2010_IDW_INTERP.tif",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2010.tif",
#           "3",
#           "FIXED 0.35",
#           "")
#
# arcpy.env.workspace = "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif"
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2011_points.shp",
#           "GRID_CODE",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2011_IDW_INTERP.tif",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2011.tif",
#           "3",
```

```
#           "FIXED 0.35",
#           "")
#
# arcpy.env.workspace = "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif"
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2012_points.shp",
#           "GRID_CODE",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2012_IDW_INTERP.tif",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2012.tif",
#           "3",
#           "FIXED 0.35",
#           "")
#
# arcpy.env.workspace = "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif"
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2013_points.shp",
#           "GRID_CODE",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2013_IDW_INTERP.tif",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2013.tif",
#           "3",
#           "FIXED 0.35",
#           "")
#
# arcpy.env.workspace = "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif"
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2014_points.shp",
#           "GRID_CODE",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2014_IDW_INTERP.tif",
#           "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2014.tif",
```

```
#		"3",
#		"FIXED 0.35",
#		"")
#
# arcpy.env.workspace = "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif"
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2015_points.shp",
#		"GRID_CODE",
#		"C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2015_IDW_INTERP.tif",
#		"C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2015.tif",
#		"3",
#		"FIXED 0.35",
#		"")
#
# arcpy.env.workspace = "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif"
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2016_points.shp",
#		"GRID_CODE",
#		"C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2016_IDW_INTERP.tif",
#		"C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2016.tif",
#		"3",
#		"FIXED 0.35",
#		"")
#
# arcpy.env.workspace = "C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2009_IDW_INTERP.tif"
#
# arcpy.gp.Idw_sa("C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/chl_x2017_points.shp",
#		"GRID_CODE",
#		"C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2017_IDW_INTERP.tif",
```

```
#		"C:/Users/trevor.joyce/Documents/Grad School/Research/1_2016_Antarctic Whale Tracking/Data/GIS Data/Chlorophyll/CHL_Cropped_Mean/CHL_X2017.tif",
#		"3",
#		"FIXED 0.35",
#		"")

# Import interpolated rasters

CHL_IDW_INTERP<- raster::stack()

for(i in list.files(paste("/Users/trevorjoyce/Grad School/Research/",
             "1_2016_Antarctic Whale Tracking/Data/Chl Data/CHL_IDW_INTERP",sep=""))){

  CHL_IDW_INTERP_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce/Grad School/Research/",
                              "1_2016_Antarctic Whale Tracking/Data/Chl Data/CHL_IDW_INTERP/",i,sep="")))

  names(CHL_IDW_INTERP_temp) <- substr(i,5,9)

  CHL_IDW_INTERP <- raster::stack(CHL_IDW_INTERP,CHL_IDW_INTERP_temp)

}

remove(CHL_IDW_INTERP_temp)

raster::projection(CHL_IDW_INTERP) <- CRS("+init=epsg:4326")


###### CHL_IDW_INTERP_RES_2000m #########

CHL_IDW_INTERP_RES_2000m <- raster::projectRaster(from = CHL_IDW_INTERP,
                          to = ICE_MODIS_LAND_MASK_RES_2000m,
                          crs = CRS("+init=epsg:4326"),
                          method = "bilinear")

CHL_IDW_INTERP_RES_2000m <- CHL_IDW_INTERP_RES_2000m * ICE_MODIS_LAND_MASK_RES_2000m

names(CHL_IDW_INTERP_RES_2000m) <- names(CHL_IDW_INTERP)


# # Write external .tif raster copy of CHL_IDW_INTERP_RES_2000m
# raster::writeRaster(CHL_IDW_INTERP_RES_2000m,
```

```
#               filename=paste("/Users/trevorjoyce/Grad School/Research/",
#                         "1_2016_Antarctic Whale Tracking/Data/Chl
Data/CHL_IDW_INTERP_RES_2000m/",
#                         "CHL_IDW_INTERP_RES_2000m.tif",sep=""),
#               format = "GTiff",bylayer = T,suffix = "names")

# Reconstitute CHL_IDW_INTERP_RES_2000m from individual saved rasters
CHL_IDW_INTERP_RES_2000m<- raster::stack()

for(i in list.files(paste("/Users/trevorjoyce/Grad School/Research/",
                 "1_2016_Antarctic Whale Tracking/Data/Chl
Data/CHL_IDW_INTERP_RES_2000m/",sep=""))){


CHL_IDW_INTERP_RES_2000m_temp=raster::raster(rgdal::readGDAL(paste("/Users/trevorjoyce
/Grad School/Research/",
                                        "1_2016_Antarctic Whale Tracking/Data/Chl
Data/CHL_IDW_INTERP_RES_2000m/",i,sep="")))

  names(CHL_IDW_INTERP_RES_2000m_temp) <- substr(i,26,30)

  CHL_IDW_INTERP_RES_2000m <-
raster::stack(CHL_IDW_INTERP_RES_2000m,CHL_IDW_INTERP_RES_2000m_temp)

}

remove(CHL_IDW_INTERP_RES_2000m_temp,i)
```