Creating grid objects for Bering-Okhotsk Seal Surveys

January 23, 2013

This document describes creation of the BOSS (Bering-Okhotsk Seal Survey) survey grid object, and details the calculation of associated environmental and landscape coordinates. All programming is done with R, making heavy use of the `sp` package and associated libraries. Snippets of code shown below. The document is prepared with `knitr`; the source .Rnw document can be used to run the code directly in an environment such as RStudio (note directories need to be set in the setup code chunks in that case). Running this code verbatim requires access to NMML network drives.

We start by declaring dependence on external libraries, noting that any missing libraries can easily be installed with the `install.packages` command.

```
library(sp)
library(rgdal)

## Geospatial Data Abstraction Library extensions to R successfully
loaded
## Loaded GDAL runtime:  GDAL 1.9.1, released 2012/05/15
## Path to GDAL shared files:  C:/Users/paul.conn/R/win-library/2.15/rgdal/gdal
## Loaded PROJ.4 runtime:  Rel.  4.7.1, 23 September 2009, [PJ_VERSION:
470]
## Path to PROJ.4 shared files:  C:/Users/paul.conn/R/win-library/2.15/rgdal/proj

library(rgeos)

## Loading required package:  stringr
## Loading required package:  plyr
## rgeos:  (SVN revision 348)
## GEOS runtime version:  3.3.5-CAPI-1.7.5
## Polygon checking:  TRUE

library(raster)

## raster 2.0-08 (27-June-2012)
##
## Attaching package:  'raster'
```

```
## The following object(s) are masked from 'package:plyr':
##
## count
```

```
library(nPacMaps)  #from Josh London
library(maptools)
```

```
## Loading required package:  foreign
## Loading required package:  lattice
## Checking rgeos availability:  TRUE
```

```
library(gpclib)
```

```
## Warning:  A specification for class "gpc.poly" in package 'gpclib'
seems equivalent to one from package 'rgeos' and is not turning on
duplicate class definitions for this class
## Warning:  A specification for class "gpc.poly.nohole" in package
'gpclib' seems equivalent to one from package 'rgeos' and is not turning
on duplicate class definitions for this class
## General Polygon Clipper Library for R (version 1.5-1)
## Type 'class ?  gpc.poly' for help
##
## Attaching package:  'gpclib'
## The following object(s) are masked from 'package:rgeos':
##
## append.poly, area.poly, get.bbox, get.pts, read.polyfile,
## scale.poly, triangulate, tristrip, write.polyfile
```

```
library(automap)
```

```
## Loading required package:  gstat
## Loading required package:  spacetime
## Loading required package:  zoo
##
## Attaching package:  'zoo'
## The following object(s) are masked from 'package:base':
##
## as.Date, as.Date.numeric
## Loading required package:  xts
##
## Attaching package:  'xts'
## The following object(s) are masked from 'package:raster':
##
## reclass
```

```
library(ggplot2)
# library(bass)
```

```
DEBUG = TRUE
MEAN_ADJUST = TRUE

if (DEBUG) source("c:/users/paul.conn/git/bass/bass/R/bass.R")
```

Currently the option `library(bass)` is commented out and we load the file `bass.R` directly since it is in the development phase. Once it is finalized, we might instead load the bass package as a library and turn `DEBUG=FALSE`.

1 Bering Sea Grid (Russia + US)

We begin by constructing a survey grid object for the Bering Sea (U.S. and Russia combined). We want to include all areas that have ice to support seals at at least one time point during seal surveys, and to break up the surveys area into 25km by 25km grid cells (the same cells used to determine sea ice concentration by the U.S. National Sea Ice Data Center). Since 2012 had a record amount of ice in the Bering Sea, we chose to loosely base the grid on those cells that had sea ice on April 1, 2012. Note that the first day of U.S. seal surveys in 2012 was April 4, while the first day for Russia surveys was April 20. We thus start by loading in a geotif for this date:

```
r <- raster("//afsc/akc-nmml/Polar/Data/Environ/SeaIce/SSMI_SIC/2012/nt_20120401_f17_nrt_n.b
```

Ice concentration is rescaled to take on values of 0-100 using the user-defined function `fun`:

```
fun <- function(x) {
    ifelse(x < 251, x/2.5, NA)
}
sic_raster <- calc(r, fun)
```

We have chosen to use the EaseGrid 2.0 projection to work with these data; this projection minimizes distortion in the Arctic and sub-Arctic regions where we are working (see J. London's readme file for additional information on this projection). So, let's reproject the data:

```
laea_180_proj <- paste("+proj=laea +lat_0=90 +lon_0=180 +x_0=0 +y_0=0", "+datum=WGS84 +units
sic_raster <- projectRaster(sic_raster, res = c(25067.53, 25067.53), crs = laea_180_proj)
```

Additionally we load Alaskan and Russian landmass polygons from J. London's `nPacMaps` R library and reproject them:

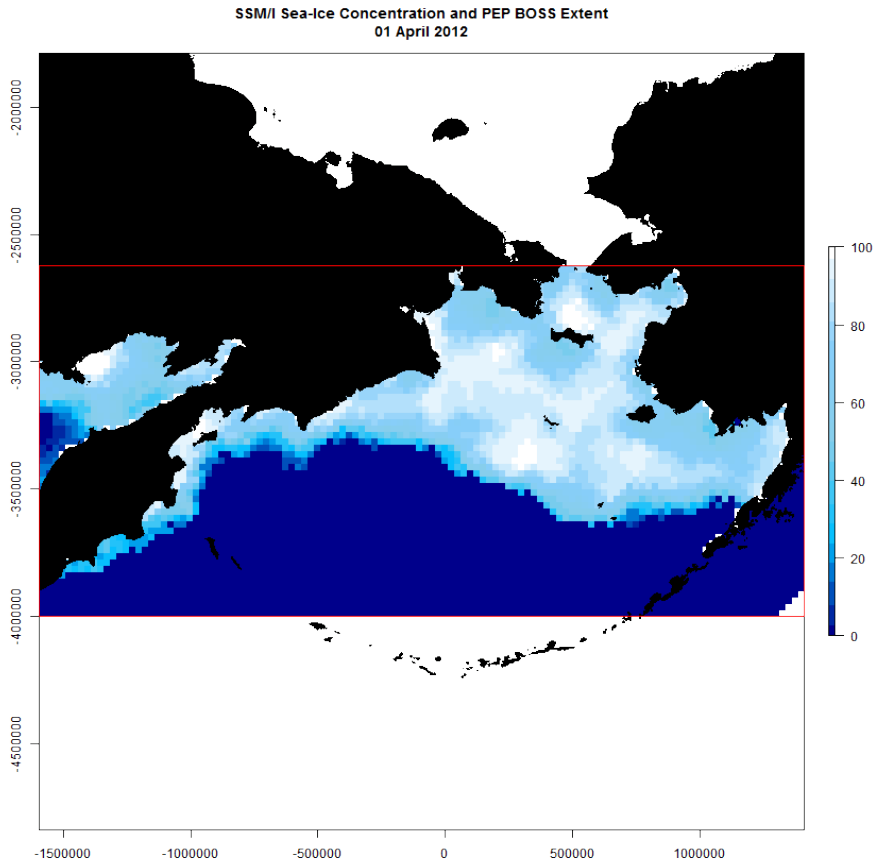
```
data(alaska_dcw)
data(russia_dcw)
alaska_dcw <- spTransform(alaska_dcw, CRS(laea_180_proj))
russia_dcw <- spTransform(russia_dcw, CRS(laea_180_proj))
```

The area encompassed by our ice imagery is much larger than we need (see J. London's readme file), so we'll limit the area under consideration to a more plausible bounding box:

```
x_min <- -1600000
x_max <- 1400000
y_min <- -4e+06
# y_max <- -1600000
y_max <- -2630000

pep_ext <- extent(x_min, x_max, y_min, y_max)
sic_raster <- crop(sic_raster, pep_ext, snap = "near")
dim_raster = dim(sic_raster)
pep_ext <- extent(sic_raster)
```

Now let's see what we've got:

```
plot(sic_raster, col = colorRampPalette(c("dark blue", "deepskyblue", "skyblue",
    "lightskyblue", "white"))(20), main = "SSM/I Sea-Ice Concentration and PEP BOSS Extent\n
plot(alaska_dcw, col = "black", add = TRUE)
plot(russia_dcw, col = "black", add = TRUE)
plot(pep_ext, col = "red", add = TRUE)
```



Here, the red bounding box encloses all locations that we might want to model, but there is still a large amount of habitat unsuitable for seals (large land masses) as well as a large amount of open water where seals will not be surveyed. We'll delete those cells later. For now, we'll attach some landscape level covariates to each 25km by 25km cell, starting with the proportion of land that occurs in each cell. To do this, we convert our grid from a raster to a spatial polygons data frame object, and call a function `add.prop.land` from `bass.R` to attach this covariate for each cell. Note that this can take quite awhile to run, but if you use `knitr` you should only have run things once. Alternatively one can just load the saved .Rdat file once it has been saved and begin there.

```
# Attach proportion land for each cell
Grid_poly <- rasterToPolygons(sic_raster, na.rm = FALSE)  #convert to SpPolyDF for compatibi
Land = list(alaska = alaska_dcw, russia = russia_dcw)

# the following takes awhile.  Instead, consider loading cur.Grid.Rdat
```

```
Grid_poly = add.prop.land(Grid = Grid_poly, Land = Land)
# remove initial layer
Grid_poly = Grid_poly[, -1]
save.image("cur.Bering.Grid.Rdat")
# load('cur.Bering.Grid.Rdat')
```

Now, we'll add distance to mainland and distance to land (including islands):

```
# define separate SpPolyDFs for mainland
Area_alaska = gArea(alaska_dcw, byid = TRUE)
Area_russia = gArea(russia_dcw, byid = TRUE)
Alaska_mainland = alaska_dcw[which(Area_alaska == max(Area_alaska)), ]
Russia_mainland = russia_dcw[which(Area_russia == max(Area_russia)), ]

# Attach distance to mainland for each cell (distance from grid cell
# centroid to landscape polygon)
Grid_points = gCentroid(Grid_poly, byid = TRUE)
Dist_AK = gDistance(Grid_points, Alaska_mainland, byid = TRUE)
Dist_Rus = gDistance(Grid_points, Russia_mainland, byid = TRUE)

## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.

Dist_mainland = apply(cbind(as.vector(Dist_AK), as.vector(Dist_Rus)), 1, "min")
if (MEAN_ADJUST) Dist_mainland = Dist_mainland/mean(Dist_mainland)
Grid_poly[["dist_mainland"]] = Dist_mainland

# Attach distance to land (including islands) for each cell
Dist_AK = apply(gDistance(Grid_points, alaska_dcw, byid = TRUE), 2, "min")
Dist_Rus = apply(gDistance(Grid_points, russia_dcw, byid = TRUE), 2, "min")

## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.
## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.
## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.

Dist_land = apply(cbind(as.vector(Dist_AK), as.vector(Dist_Rus)), 1, "min")
if (MEAN_ADJUST) Dist_land = Dist_land/mean(Dist_land)
Grid_poly[["dist_land"]] = Dist_land
```

and distance to shelf break (here defined as distance to a 1000m depth contour, which we read in as a shapefile):

```
# Attach distance to shelf break (1000m depth contour) for each cell
Shelf_break = readOGR("c:/users/paul.conn/git/bass/shapefiles/1000m_depth_contour.shp",
    layer = "1000m_depth_contour")

## OGR data source with driver: ESRI Shapefile
## Source: "c:/users/paul.conn/git/bass/shapefiles/1000m_depth_contour.shp", layer: "1000m_c
## with 3 features and 3 fields
## Feature type: wkbPolygon with 2 dimensions

Shelf_break = spTransform(Shelf_break, CRS(laea_180_proj))
Shelf_break1 = gBoundary(Shelf_break[1, ])  #US side of EEZ
Shelf_break2 = gBoundary(Shelf_break[2, ])  #russia side
# remove part of Shelf break that intersect with land or the EEZ line
Shelf_break1 = gDifference(Shelf_break1, gBuffer(alaska_dcw, width = 10000,
    byid = TRUE))
Shelf_break1 = gDifference(Shelf_break1, gBuffer(russia_dcw, width = 10000,
    byid = TRUE))

## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.
## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.
## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.

Shelf_break2 = gDifference(Shelf_break2, gBuffer(alaska_dcw, width = 10000,
    byid = TRUE))
Shelf_break2 = gDifference(Shelf_break2, gBuffer(russia_dcw, width = 10000,
    byid = TRUE))

## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.
## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.
## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.


EEZs = readOGR("c:/users/paul.conn/git/bass/shapefiles/EEZ_Albers83_BS.shp",
    layer = "EEZ_Albers83_BS")

## OGR data source with driver: ESRI Shapefile
## Source: "c:/users/paul.conn/git/bass/shapefiles/EEZ_Albers83_BS.shp", layer: "EEZ_Albers8
## with 2 features and 10 fields
## Feature type: wkbPolygon with 2 dimensions
```

```
EEZs = spTransform(EEZs, CRS(laea_180_proj))
EEZ_Alaska = EEZs[1, ]  #limit to Alaska EEZ
Shelf_break1 = gDifference(Shelf_break1, gBuffer(gBoundary(EEZ_Alaska), width = 30000))
Shelf_break2 = gDifference(Shelf_break2, gBuffer(gBoundary(EEZ_Alaska), width = 30000))

Dist_shelf1 = apply(gDistance(Grid_points, Shelf_break1, byid = TRUE), 2, "min")
Dist_shelf2 = apply(gDistance(Grid_points, Shelf_break2, byid = TRUE), 2, "min")
Dist_shelf = apply(cbind(Dist_shelf1, Dist_shelf2), 1, "min")

if (MEAN_ADJUST) Dist_shelf = Dist_shelf/mean(Dist_shelf)
Grid_poly[["dist_shelf"]] = Dist_shelf
```

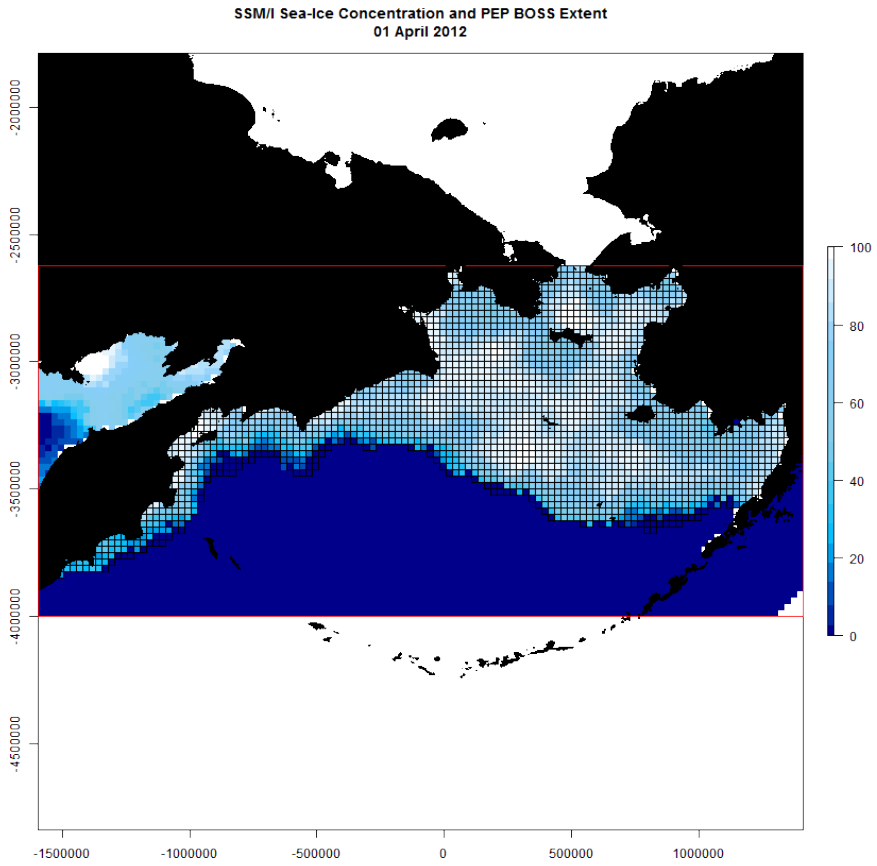
Now, we've attached all of the covariates that we plan to that are time constant (we still need to attach time varying covariates that are a function of changing sea ice concentration). Let's output our current grid before doing anything else. We can also calculate a King's move adjacency matrix easily at this stage since the grid is a perfect rectangle (for use with areal spatial models):

```
save(Grid_poly, file = "BOSS_Grid_Bering_static.Rdat")
# load('BOSS_Grid_Bering_static.Rdat')
Adj = rect_adj(x = dim_raster[2], y = dim_raster[1], byrow = TRUE)
```

The next step is to define which cells will be included in analysis. We take out cells with $> 99\%$ land, as well as cells that never have ice between April 4 and May 14 (roughly the times that were surveyed). Eventually we'd like to extend the date from May 14 to some time later but currently we're missing sea ice concentration data from 5/14/12-5/19/12. We also take out cells that are in the Sea of Okhotsk because these will be analyzed separately. Following these reductions, we reduce the spatial adjacency matrix to appropriate size and plot our current grid on top of our old land masses.

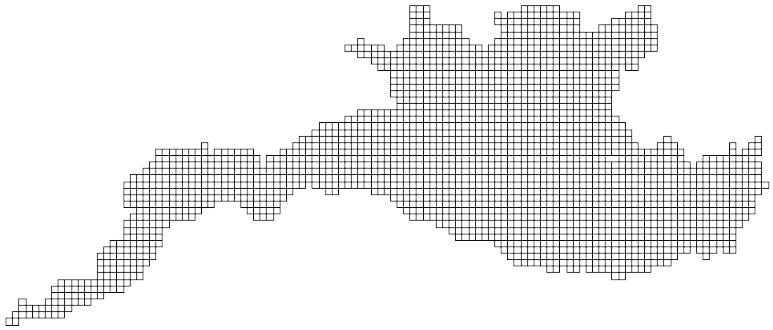
```
# define indicator vector for which cells are to be included in analysis
# (exclude all cells north of xx latitude (Bering Straight), those that
# are 100% land, and those that have 100% water over course of study) dont
# include cells above Bering Strait
Include = (coordinates(Grid_points)[, 2] < (-2630000))
# take out some cells in the bottom right quadrant that are over open
# water but near land so they have NA ice values
Include[which(coordinates(Grid_points)[, 1] > 8e+05 & coordinates(Grid_points)[,
    2] < (-3700000))] = 0
Include[which(coordinates(Grid_points)[, 1] > 1e+06 & coordinates(Grid_points)[,
    2] < (-3650000))] = 0
Include[which(coordinates(Grid_points)[, 1] > 1100000 & coordinates(Grid_points)[,
    2] < (-3600000))] = 0
Include[which(Grid_poly[["land_cover"]] > 0.99)] = 0  #dont include points with > 99% land
```

```
# scroll through sea ice data to determine which cells always occur in
# open water
Date = c(paste("040", c(4:9), sep = ""), paste("04", c(10:30), sep = ""), paste("050",
    c(1:9), sep = ""), paste("05", c(10:14), sep = ""))  #limit dates to 4/4-5/22 (first-las
# Date=c(paste('04',c(20:27),sep='')) #limit dates to 4/20-4/27
# (first-last BOSS flights used in power analysis)
str1 = "//afsc/akc-nmml/Polar/Data/Environ/SeaIce/SSMI_SIC/2012/nt_2012"
str2 = "_f17_nrt_n.bin.reproj.tif"
Ice1 = rep(0, length(Include))
for (idate in 1:length(Date)) {
    filename = paste(str1, Date[idate], str2, sep = "")
    r <- raster(filename)
    tmp_raster <- calc(r, fun)
    tmp_raster <- projectRaster(tmp_raster, res = c(25067.53, 25067.53), crs = laea_180_proj
    tmp_raster <- crop(tmp_raster, pep_ext, snap = "near")
    tmp_raster[is.na(tmp_raster)] = 1
    Ice1 = Ice1 + values(tmp_raster > 0)
}
Include[which(Ice1 == 0)] = 0

# remove cells in Sea of Okhotsk
I.Okhotsk = 1 - gIntersects(Shelf_break[3, ], Grid_poly, byid = TRUE)
Include[which(I.Okhotsk == 0)] = 0

# reduce 'grid' to those cells of interest for analysis, reformatting
# spatial connectivity matrix and declaring list of spatial polygon
# dataframe objects
Grid_reduced = Grid_poly[Include == 1, ]

Adj_reduced = Adj[Include == 1, Include == 1]
plot(sic_raster, col = colorRampPalette(c("dark blue", "deepskyblue", "skyblue",
    "lightskyblue", "white"))(20), main = "SSM/I Sea-Ice Concentration and PEP BOSS Extent\n
plot(alaska_dcw, col = "black", add = TRUE)
plot(russia_dcw, col = "black", add = TRUE)
plot(pep_ext, col = "red", add = TRUE)
plot(Grid_reduced, add = TRUE)
```



We can also take a look at the grid without land masses

```
plot(Grid_reduced)
```



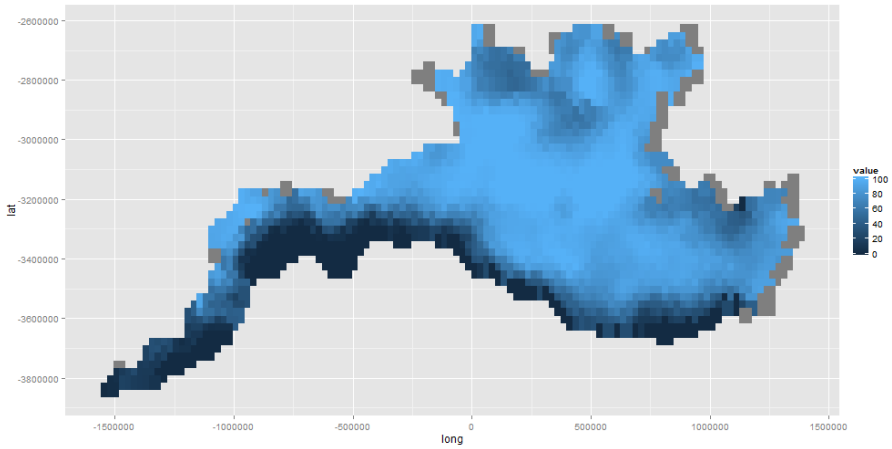
To accomodate time-varying covariates, we'll define a data structure `Data` that is a list object. We'll start by including two objects at the first level of the list, `Adj` and `Grid`. We'll then have the element `Grid` itself be a list, each element of which is an `SpatialPolygonsDataFrame` that includes all relevant covariates (both static and time varying) for the date in question (so there will be a separate `SpatialPolygonsDataFrame`) for each day. Let's define this structure and initialize the covariates for each daily SPDF to each have the same static covariates (distance from land, etc.):

```
Data = list(Adj = Adj_reduced)
Data$Grid = vector("list", length(Date))  #allocate space for one SpatialPolygonsDataframe f
for (idate in 1:length(Date)) Data$Grid[[idate]] = Grid_reduced
```

Now it's time to start adding in time-varying covariates, which should go a little quicker now that we've reduced the size of our grid. In particular, we'll attach daily sea ice concentration, distance to 10% sea-ice concentration contour (a proxy for distance from water) and distance from southern ice edge

for each cell. Before we do this however, it is worthwhile to note that there are sometimes gaps between land masses and water where sea ice concentration values are missing (they appear as NA values). For instance, let's take a look at our sea ice data on April 4:

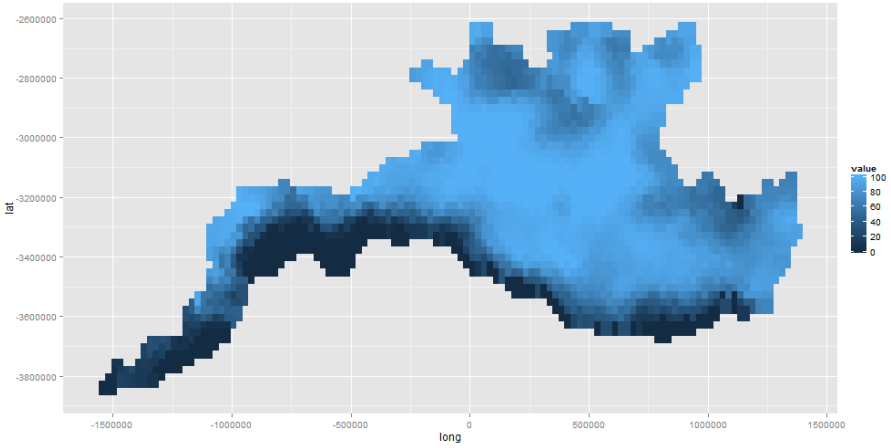
```
filename = paste(str1, Date[1], str2, sep = "")
r <- raster(filename)
tmp_raster <- calc(r, fun)
tmp_raster <- projectRaster(tmp_raster, res = c(25067.53, 25067.53), crs = laea_180_proj)
tmp_raster <- crop(tmp_raster, pep_ext, snap = "near")
ice_conc = values(tmp_raster)[Include == 1]
sum(is.na(ice_conc))

## [1] 80

sic_poly <- rasterToPolygons(tmp_raster, na.rm = FALSE)  #convert to SpPolygonsDF (conversio
ice.poly = sic_poly[Include == 1, ]
ice.poly@data$id = rownames(ice.poly@data)
library(ggplot2)
tmp1 <- fortify(ice.poly, region = "id")
tmp2 <- join(tmp1, ice.poly@data, by = "id")
ggplot(tmp2) + aes(long, lat, fill = value) + geom_raster()
```



Here, grey cells represent sea ice concentrations that were given NA values (note that there are 80 of them). Let's use the function `autoKrige` in the `automap` package to do some kriging and interpolate missing values:

```
ice.conc = sic_poly@data[, 1]
sic_points = gCentroid(sic_poly, byid = TRUE)
Include2 = Include
Include2[is.na(ice.conc)] = 0
Include3 = Include
```

```
Include3[is.na(ice.conc) == FALSE] = 0
n.na = sum(Include3)
na.dm = data.frame(matrix(rep(0, n.na), ncol = 1))
colnames(na.dm) = "ice.conc"
ice.conc = data.frame(matrix(ice.conc[Include2 == 1], ncol = 1))
colnames(ice.conc) = "ice.conc"
coords = coordinates(sic_points)[Include2 == 1, ]
coords.pred = coordinates(sic_points)[Include3 == 1, ]
rownames(coords) = c(1:nrow(coords))
rownames(coords.pred) = c(1:nrow(coords.pred))
sic_points = SpatialPointsDataFrame(coords, ice.conc, proj4string = CRS(laea_180_proj))
pred_loc = SpatialPointsDataFrame(coords.pred, na.dm, proj4string = CRS(laea_180_proj))
krige_out = autoKrige(ice.conc ~ 1, input_data = sic_points, new_data = pred_loc)$krige_outp

## [using ordinary kriging]

ice.poly = sic_poly[Include == 1, ]
ice.poly[[1]][which(is.na(ice.poly[[1]] == TRUE))] = krige_out
ice.poly@data$id = rownames(ice.poly@data)
tmp.df <- fortify(ice.poly, region = "id")
tmp.df <- join(tmp.df, ice.poly@data, by = "id")
ggplot(tmp.df) + aes(long, lat, fill = value) + geom_raster()
```



So kriging seems to work as a strategy for filling in missing sea ice concentration covariates. Let's go ahead and fill these in for all dates.

```
for (idate in 1:length(Date)) {
    filename = paste(str1, Date[idate], str2, sep = "")
    r <- raster(filename)
    tmp_raster <- calc(r, fun)
    tmp_raster <- projectRaster(tmp_raster, res = c(25067.53, 25067.53), crs = laea_180_proj
```

```
    tmp_raster <- crop(tmp_raster, pep_ext, snap = "near")
    sic_poly <- rasterToPolygons(tmp_raster, na.rm = FALSE)
    sic_points = gCentroid(sic_poly, byid = TRUE)
    ice.conc = sic_poly@data[, 1]
    Include2 = Include
    Include2[is.na(ice.conc)] = 0
    Include3 = Include
    Include3[is.na(ice.conc) == FALSE] = 0
    n.na = sum(Include3)
    na.dm = data.frame(matrix(rep(0, n.na), ncol = 1))
    colnames(na.dm) = "ice.conc"
    ice.conc = data.frame(matrix(ice.conc[Include2 == 1], ncol = 1))
    colnames(ice.conc) = "ice.conc"
    coords = coordinates(sic_points)[Include2 == 1, ]
    coords.pred = coordinates(sic_points)[Include3 == 1, ]
    rownames(coords) = c(1:nrow(coords))
    rownames(coords.pred) = c(1:nrow(coords.pred))
    sic_points = SpatialPointsDataFrame(coords, ice.conc, proj4string = CRS(laea_180_proj))
    pred_loc = SpatialPointsDataFrame(coords.pred, na.dm, proj4string = CRS(laea_180_proj))
    krige_out = autoKrige(ice.conc ~ 1, input_data = sic_points, new_data = pred_loc)$krige_
    ice.poly = sic_poly[Include == 1, ]
    ice.poly[[1]][which(is.na(ice.poly[[1]] == TRUE))] = krige_out
    ice_conc = ice.poly[[1]]/100
    Data$Grid[[idate]] = spCbind(Data$Grid[[idate]], ice_conc)
}
```

```
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
```

```
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
```

Now that that's done, it's time to add some additional time-varying covariates. One thing that we'd like to do is to calculate distance from the center of the current cell to 'open water.' Here, we use a 10% sea ice contour to represent the edge between ice and water. To determine these contours, we translate each SpPolyDF into a raster and use the function `rasterToContour` to generate countour lines at 10% ice concentration. Then we use some functions in the `rgeos` package to calculate a distance from each cell's centroid to these contour lines:

```
tmp.raster <- raster(nrow = dim_raster[1], ncol = dim_raster[2])
extent(tmp.raster) = extent(Data$Grid[[1]])
for (idate in 1:length(Date)) {
    tmp.raster = rasterize(Data$Grid[[idate]], tmp.raster, "ice_conc")
    sic_contour = rasterToContour(tmp.raster, levels = c(0.1))
    dist_contour = as.vector(gDistance(Grid_points[Include == 1, ], sic_contour,
        byid = TRUE))
    if (MEAN_ADJUST)
        dist_contour = dist_contour/mean(dist_contour)
    Data$Grid[[idate]] = spCbind(Data$Grid[[idate]], dist_contour)
}
```

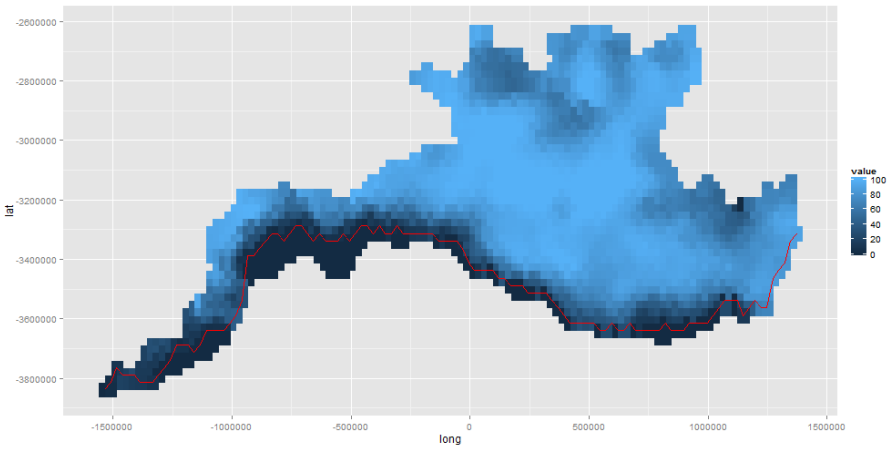
```
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
```

```
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
## Found 2132 region(s) and 2132 polygon(s)
```

One issue with using this covariate in the Bering, however, is that there are often polynia; what we might want instead is distance to the southernmost ice edge. I have developed a function `get.edge.Bering` in `bass.R` to calculate a single line associated with the southern ice edge. The function works simply taking the southern ice edge for each unique longitude (using centroids for each cell) and connecting those points to define a line. Using data from April 1 2012,

here is an example of what this line looks like:

```
tmp = get.edge.Bering(SpDF = Data$Grid[[1]], proj = laea_180_proj)
tmp.plot = ggplot(tmp.df) + aes(long, lat, fill = value) + geom_raster()
tmp.line = data.frame(coordinates(tmp))
tmp.line = cbind(tmp.line, 2)
colnames(tmp.line) = c("long", "lat", "value")
tmp.plot + geom_line(data = tmp.line, col = "red")
```



So, let's once again calculate distance from the centroid of each 25km by 25km to southern ice edge for each day:

```
for (idate in 1:length(Date)) {
    edge.line = get.edge.Bering(SpDF = Data$Grid[[idate]], proj = laea_180_proj)
    dist_edge = as.vector(gDistance(Grid_points[Include == 1, ], edge.line,
        byid = TRUE))
    if (MEAN_ADJUST)
        dist_edge = dist_edge/mean(dist_edge)
    Data$Grid[[idate]] = spCbind(Data$Grid[[idate]], dist_edge)
}
```

Now we're done with the full Bering dataset. Let's save our completed list object in `BeringData.Rdat` which can be loaded directly into R. We'll also output a shapefile for use with GIS applications.

```
save(Data, file = "BeringData.Rdat")
writeOGR(Grid_reduced, dsn = ".", layer = "BeringGrid2012", driver = "ESRI Shapefile")

## Warning:  Field names abbreviated for ESRI Shapefile driver
```

To isolate grid cells from this object that are in the U.S. EEZ, we can simply subset the data:

```
I.Alaska = as.vector(gIntersects(Data$Grid[[1]], EEZ_Alaska, byid = TRUE))
for (idate in 1:length(Date)) {
    Data$Grid[[idate]] = Data$Grid[[idate]][I.Alaska, ]
}
Data$Adj = Data$Adj[I.Alaska, I.Alaska]
save(Data, file = "AlaskaBeringData.Rdat")
```

So, we now have an object to base analysis on, at least once we are able to integrate hotspot/photo data and effort information. An advantage of our approach is we should be able to use some of the spatial tools in `rgeos` to attach hotspots and flight tracks to gridlines.

2 Okhotsk Grid

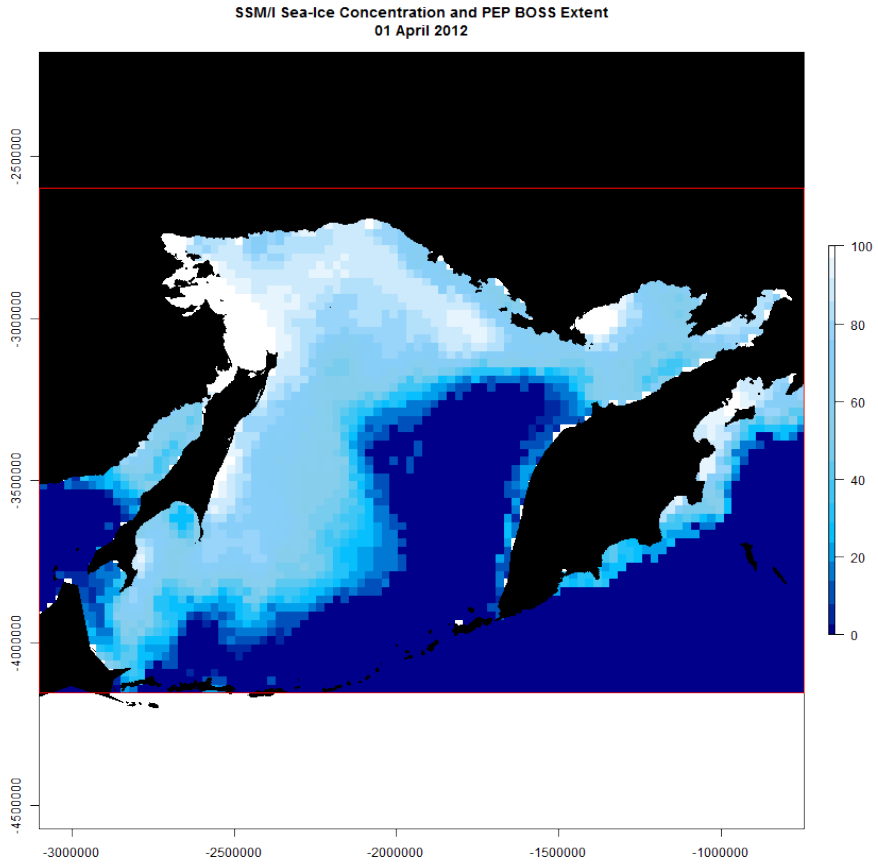
It is a similar process for developing Spatial Polygon Data Frames for the Okhostk Sea. This time, our land layers include Russia (already defined) and Japan. To get Japan, let's load the dataset `wrld_simpl` from the `maptools` library in R. We

```
data(wrld_simpl)
japan <- wrld_simpl[89, ]
japan <- spTransform(japan, CRS(laea_180_proj))
```

Let's reload the large sea ice raster and cut it down to approximate size for the Okhotsk.

```
r <- raster("//afsc/akc-nmml/Polar/Data/Environ/SeaIce/SSMI_SIC/2012/nt_20120401_f17_nrt_n.b
fun <- function(x) {
    ifelse(x < 251, x/2.5, NA)
}
sic_raster <- calc(r, fun)
sic_raster <- projectRaster(sic_raster, res = c(25067.53, 25067.53), crs = laea_180_proj)

x_min <- -3100000
x_max <- -750000
y_min <- -4150000
y_max <- -2600000

pep_ext <- extent(x_min, x_max, y_min, y_max)
sic_raster <- crop(sic_raster, pep_ext, snap = "near")
dim_raster = dim(sic_raster)
pep_ext <- extent(sic_raster)

plot(sic_raster, col = colorRampPalette(c("dark blue", "deepskyblue", "skyblue",
    "lightskyblue", "white"))(20), main = "SSM/I Sea-Ice Concentration and PEP BOSS Extent\n
```

```
plot(japan, col = "black", add = TRUE)
plot(russia_dcw, col = "black", add = TRUE)
plot(pep_ext, col = "red", add = TRUE)
```



Now, let's attach some static (time constant) covariates as we did for the Bering, including distance from land and distance from shelf break. For the Okhotsk, we won't differentiate between distance from mainland and distand from land, as most land masses are large (at least where there tends to be ice). Note that this step takes a long time (up to an hour) to run.

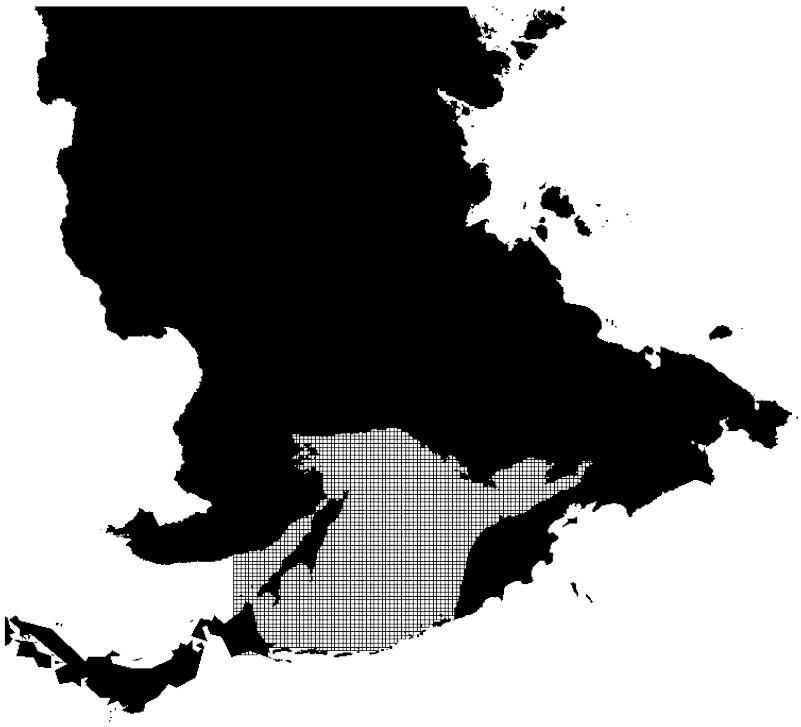
```
# Attach proportion land for each cell
Grid_poly <- rasterToPolygons(sic_raster, na.rm = FALSE)  #convert to SpPolyDF for compatibi
Land = list(japan = japan, russia = russia_dcw)

# the following takes awhile.  Instead, consider loading cur.Grid.Rdat
Grid_poly = add.prop.land(Grid = Grid_poly, Land = Land)
# remove initial layer
```

```
Grid_poly = Grid_poly[, -1]
save.image("cur.Okhotsk.Grid.Rdat")
# load('cur.Bering.Grid.Rdat')

Grid_points = gCentroid(Grid_poly, byid = TRUE)

# Attach distance to land (including islands) for each cell
Dist_Japan = apply(gDistance(Grid_points, japan, byid = TRUE), 2, "min")
Dist_Rus = apply(gDistance(Grid_points, russia_dcw, byid = TRUE), 2, "min")

## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.
## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.
## Warning:  Polygons object missing comment attribute ignoring hole(s).
See function createSPComment.

Dist_land = apply(cbind(as.vector(Dist_Japan), as.vector(Dist_Rus)), 1, "min")
if (MEAN_ADJUST) Dist_land = Dist_land/mean(Dist_land)
Grid_poly[["dist_land"]] = Dist_land

# Attach distance to shelf break (1000m depth contour) for each cell
Shelf_break = readOGR("c:/users/paul.conn/git/bass/shapefiles/1000m_depth_contour.shp",
    layer = "1000m_depth_contour")

## OGR data source with driver: ESRI Shapefile
## Source: "c:/users/paul.conn/git/bass/shapefiles/1000m_depth_contour.shp", layer: "1000m_c
## with 3 features and 3 fields
## Feature type: wkbPolygon with 2 dimensions

Shelf_break = spTransform(Shelf_break, CRS(laea_180_proj))

Shelf_break = gBoundary(Shelf_break[3, ])  #Okhotsk only
Shelf_break = gDifference(Shelf_break, gBuffer(japan, width = 10000))

Dist_shelf = apply(gDistance(Grid_points, Shelf_break, byid = TRUE), 2, "min")
if (MEAN_ADJUST) Dist_shelf = Dist_shelf/mean(Dist_shelf)
Grid_poly[["dist_shelf"]] = Dist_shelf


# output grid - this outputs spatial polygons dataframe for 'large' grid
# with all time-constant covariates (e.g. distance to land).  We'll define
# time-dependent covariates on a smaller grid since this will have to be a
# list of spatial polygon DFs and will take up considerably more disk
# space
save(Grid_poly, file = "BOSS_Grid_Okhotsk_static.Rdat")
```

```
# load('BOSS_Grid_Okhotsk_static.Rdat')

# create spatial connectivity matrix for CAR/ICAR modeling
Adj = rect_adj(x = dim_raster[2], y = dim_raster[1], byrow = TRUE)
```

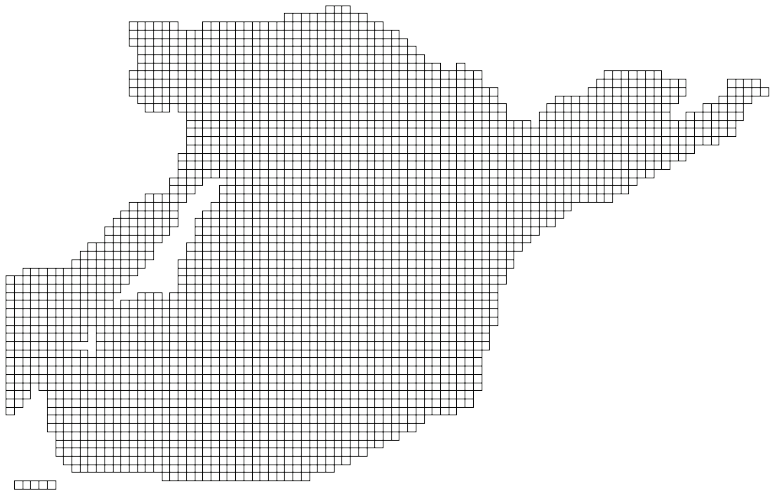
Now that the static grid is defined, let's further cut the grid down to size by taking out cells that include >99% land, as well as those that overlap with the Bering Sea. We used a self-drawn shapefile `Okhotsk_buffer.shp` to help remove portions of the initial grid that occurred in the Bering:

```
# define indicator vector for which cells are to be included in analysis
# (exclude all cells north of xx latitude (Bering Straight), those that
# are 100% land, and those that have 100% water over course of study)
Include = rep(1, length(Grid_points))  #dont include cells above Bering Strait
Include[which(Grid_poly[["land_cover"]] > 0.99)] = 0  #dont include points with > 99% land
# remove cells in Bering
Bering.poly = readOGR("c:/users/paul.conn/git/bass/shapefiles/Okhotsk_buffer.shp",
    layer = "Okhotsk_buffer")

## OGR data source with driver: ESRI Shapefile
## Source: "c:/users/paul.conn/git/bass/shapefiles/Okhotsk_buffer.shp", layer: "Okhotsk_buf
## with 1 features and 1 fields
## Feature type: wkbPolygon with 2 dimensions

Bering.poly = spTransform(Bering.poly, CRS(laea_180_proj))
I.Bering = 1 - gIntersects(Bering.poly, Grid_poly, byid = TRUE)
Include[which(I.Bering == 0)] = 0

# reduce 'grid' to those cells of interest for analysis, reformatting
# spatial connectivity matrix and declaring list of spatial polygon
# dataframe objects
Grid_reduced = Grid_poly[Include == 1, ]
Adj_reduced = Adj[Include == 1, Include == 1]
plot(russia_dcw, col = "black")
plot(japan, col = "black", add = TRUE)
plot(Grid_reduced, add = TRUE)
```



Taking out land, the Okhotsk grid simply looks like this:

```
plot(Grid_reduced)
```



Now it's time to once again attach temporally varying covariates. This time, we'll focus simply on ice concentration and distance to ice edge. There seems less reason to differentiate between difference to water and difference to southern ice edge in this instance because polynia occur less often in the Okhotsk and there is less of a difference as to where these occur (whereas in the Bering, polynia often occurred at great distances from the southern ice edge). We'll once again perform kriging on sea ice concentration levels to fill in in missing values before recording covariates.

```
Data = list(Adj = Adj_reduced)
Date = c(paste("040", c(4:9), sep = ""), paste("04", c(10:30), sep = ""), paste("050",
    c(1:9), sep = ""), paste("05", c(10:14), sep = ""))  #limit dates to 4/4-5/22 (first-las
Data$Grid = vector("list", length(Date))  #allocate space for one SpatialPolygonsDataframe f
for (idate in 1:length(Date)) Data$Grid[[idate]] = Grid_reduced
str1 = "//afsc/akc-nmml/Polar/Data/Environ/SeaIce/SSMI_SIC/2012/nt_2012"
str2 = "_f17_nrt_n.bin.reproj.tif"
```

```r
# attach daily sea ice concentration and distance to 10% sea-ice
# concentration contour
for (idate in 1:length(Date)) {
    cat(paste("\n date", idate, "of", length(Date), "\n"))
    filename = paste(str1, Date[idate], str2, sep = "")
    r <- raster(filename)
    tmp_raster <- calc(r, fun)
    tmp_raster <- projectRaster(tmp_raster, res = c(25067.53, 25067.53), crs = laea_180_proj
    tmp_raster <- crop(tmp_raster, pep_ext, snap = "near")
    sic_poly <- rasterToPolygons(tmp_raster, na.rm = FALSE)  #convert to SpPolygonsDF (conve
    sic_points = gCentroid(sic_poly, byid = TRUE)
    ice.conc = sic_poly@data[, 1]
    Include2 = Include
    Include2[is.na(ice.conc)] = 0
    Include3 = Include
    Include3[is.na(ice.conc) == FALSE] = 0
    n.na = sum(Include3)
    ice.conc = data.frame(matrix(ice.conc[Include2 == 1], ncol = 1))
    colnames(ice.conc) = "ice.conc"
    coords = coordinates(sic_points)[Include2 == 1, ]
    coords.pred = coordinates(sic_points)[Include3 == 1, ]
    rownames(coords) = c(1:nrow(coords))
    rownames(coords.pred) = c(1:nrow(coords.pred))
    sic_points = SpatialPointsDataFrame(coords, ice.conc, proj4string = CRS(laea_180_proj))
    na.dm = data.frame(matrix(rep(0.5, n.na), ncol = 1))
    colnames(na.dm) = "ice.conc"
    pred_loc = SpatialPointsDataFrame(coords.pred, na.dm, proj4string = CRS(laea_180_proj))
    krige_out = autoKrige(ice.conc ~ 1, input_data = sic_points, new_data = pred_loc)$krige_
    ice.poly = sic_poly[Include == 1, ]
    ice.poly[[1]][which(is.na(ice.poly[[1]] == TRUE))] = krige_out
    ice_conc = ice.poly[[1]]/100
    Data$Grid[[idate]] = spCbind(Data$Grid[[idate]], ice_conc)
}
```

```
##
##  date 1 of 41
## [using ordinary kriging]
##
##  date 2 of 41
## [using ordinary kriging]
##
##  date 3 of 41
## [using ordinary kriging]
##
##  date 4 of 41
## [using ordinary kriging]
```

```
##
##  date 5 of 41
## [using ordinary kriging]
##
##  date 6 of 41
## [using ordinary kriging]
##
##  date 7 of 41
## [using ordinary kriging]
##
##  date 8 of 41
## [using ordinary kriging]
##
##  date 9 of 41
## [using ordinary kriging]
##
##  date 10 of 41
## [using ordinary kriging]
##
##  date 11 of 41
## [using ordinary kriging]
##
##  date 12 of 41
## [using ordinary kriging]
##
##  date 13 of 41
## [using ordinary kriging]
##
##  date 14 of 41
## [using ordinary kriging]
##
##  date 15 of 41
## [using ordinary kriging]
##
##  date 16 of 41
## [using ordinary kriging]
##
##  date 17 of 41
## [using ordinary kriging]
##
##  date 18 of 41
## [using ordinary kriging]
##
##  date 19 of 41
## [using ordinary kriging]
```

```
##
##  date 20 of 41
## [using ordinary kriging]
##
##  date 21 of 41
## [using ordinary kriging]
##
##  date 22 of 41
## [using ordinary kriging]
##
##  date 23 of 41
## [using ordinary kriging]
##
##  date 24 of 41
## [using ordinary kriging]
##
##  date 25 of 41
## [using ordinary kriging]
##
##  date 26 of 41
## [using ordinary kriging]
##
##  date 27 of 41
## [using ordinary kriging]
##
##  date 28 of 41
## [using ordinary kriging]
##
##  date 29 of 41
## [using ordinary kriging]
##
##  date 30 of 41
## [using ordinary kriging]
##
##  date 31 of 41
## [using ordinary kriging]
##
##  date 32 of 41
## [using ordinary kriging]
##
##  date 33 of 41
## [using ordinary kriging]
##
##  date 34 of 41
## [using ordinary kriging]
```

```
##
##  date 35 of 41
## [using ordinary kriging]
##
##  date 36 of 41
## [using ordinary kriging]
##
##  date 37 of 41
## [using ordinary kriging]
##
##  date 38 of 41
## [using ordinary kriging]
##
##  date 39 of 41
## [using ordinary kriging]
##
##  date 40 of 41
## [using ordinary kriging]
##
##  date 41 of 41
## [using ordinary kriging]
```

Missing ice values are now filled in for all dates under consideration, so now we'll attach distance to ice edge contour:

```
tmp.raster <- raster(nrow = dim_raster[1], ncol = dim_raster[2])
extent(tmp.raster) = extent(Data$Grid[[1]])
for (idate in 1:length(Date)) {
    tmp.raster = rasterize(Data$Grid[[idate]], tmp.raster, "ice_conc")
    sic_contour = rasterToContour(tmp.raster, levels = c(0.1))
    dist_contour = as.vector(gDistance(Grid_points[Include == 1, ], sic_contour,
        byid = TRUE))
    if (MEAN_ADJUST)
        dist_contour = dist_contour/mean(dist_contour)
    Data$Grid[[idate]] = spCbind(Data$Grid[[idate]], dist_contour)
}
```

```
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
```

```
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
## Found 2887 region(s) and 2887 polygon(s)
```

Finally, we'll output an .rdat and shapefile.

```
# add some metadata
Data$Meta = list(date.start = "4Apr2012", date.end = "14May2012")
Data$Meta$info = "2012 PEP BOSS survey grid data for Okhotsk Sea"

# save dataset
save(Data, file = "Okhotsk_Grid_2012.rdat")

# output grid data from one year as a shapefile for Mike & other Arc users
writeOGR(Grid_reduced, dsn = ".", layer = "OkhotskGrid2012", driver = "ESRI Shapefile")
```