Spatial Indicators Working Group

Manipulating spatial data

We'll start by grabbing some of the NWFSC trawl survey data for a single species.

```
devtools::install_github("nwfsc-assess/nwfscSurvey")
```

```
## Skipping install of 'nwfscSurvey' from a github remote, the SHA1 (43c7a1fa) has not changed since las
##   Use `force = TRUE` to force installation
```

```
library(nwfscSurvey)
# pull data for pacific ocean perch from NWFSC survey
if(!file.exists("nwfsc_pop.rds")) {
pop = PullCatch.fn(SciName="Sebastes alutus", SurveyName = "NWFSC.Combo")
saveRDS(pop, file="nwfsc_pop.rds")
}
pop = readRDS("nwfsc_pop.rds")
names(pop) = tolower(names(pop))
head(pop)
```

```
##       project     trawl_id year pass       vessel tow        date depth_m
## 1 NWFSC.Combo 200303006111 2003    1 Captain Jack 111 2003-Jul-29    79.9
## 2 NWFSC.Combo 200303008092 2003    2    Excalibur  92 2003-Sep-27   948.4
## 3 NWFSC.Combo 200303010119 2003    1    Ms. Julie 119 2003-Jul-30    63.6
## 4 NWFSC.Combo 200403008067 2004    2    Excalibur  67 2004-Sep-06   131.7
## 5 NWFSC.Combo 200503008205 2005    2    Excalibur 205 2005-Oct-16   581.3
## 6 NWFSC.Combo 200503010013 2005    1    Ms. Julie  13 2005-May-31   136.0
##   longitude_dd latitude_dd area_swept_ha scientific_name subsample_count
## 1    -123.1244    38.07694      1.837650 Sebastes alutus               0
## 2    -124.6703    41.48583      3.047135 Sebastes alutus               0
## 3    -122.5511    37.37694      1.651004 Sebastes alutus               0
## 4    -124.4600    43.67444      1.821803 Sebastes alutus               0
## 5    -118.6017    33.65722      1.178600 Sebastes alutus               0
## 6    -124.2775    45.83417      1.530574 Sebastes alutus               0
##   subsample_wt_kg total_catch_numbers total_catch_wt_kg cpue_kg_km2
## 1               0                   0                 0           0
## 2               0                   0                 0           0
## 3               0                   0                 0           0
## 4               0                   0                 0           0
## 5               0                   0                 0           0
## 6               0                   0                 0           0
```

We can start by converting the data to UTMs, just to demonstrate the transformation. This can be done pretty interchangably in `sp` or `sf`. I'll use `sf` going forward because it allows for easy conversion to `sp`. To do this with `sp`,

```
pop_ll = pop
coordinates(pop_ll) <- c("longitude_dd", "latitude_dd")
proj4string(pop_ll) <- CRS("+proj=longlat +datum=WGS84")
# convert to utm with spTransform
pop_utm = spTransform(pop_ll,
  CRS("+proj=utm +zone=10 +datum=WGS84 +units=km"))
# convert back from sp object to data frame
```

```r
pop_utm_df = as.data.frame(pop_utm)
```

Doing the same with `sf`,

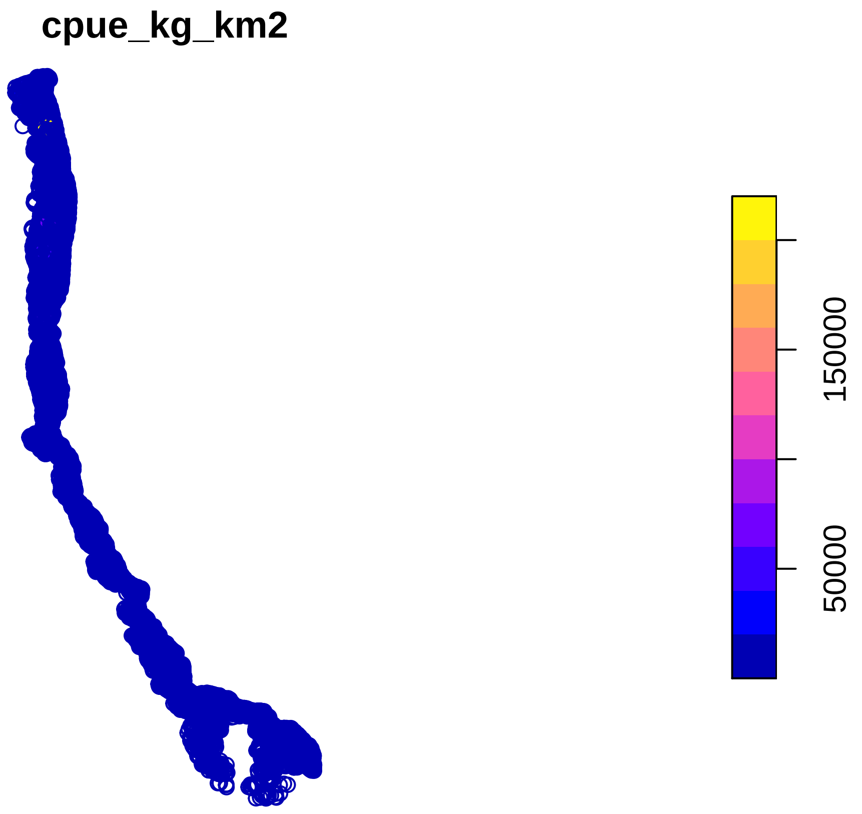
```r
pop_ll = st_as_sf(pop, coords = c("longitude_dd","latitude_dd"))
st_crs(pop_ll) = "+proj=longlat +datum=WGS84"
pop_utm = st_transform(pop_ll,
  st_crs("+proj=utm +zone=10 +datum=WGS84 +units=km"))

# convert to data frame via sp
pop_df = as(pop_utm, "Spatial") %>% data.frame %>%
  dplyr::rename(lon=coords.x1, lat=coords.x2)
```

Basic plotting

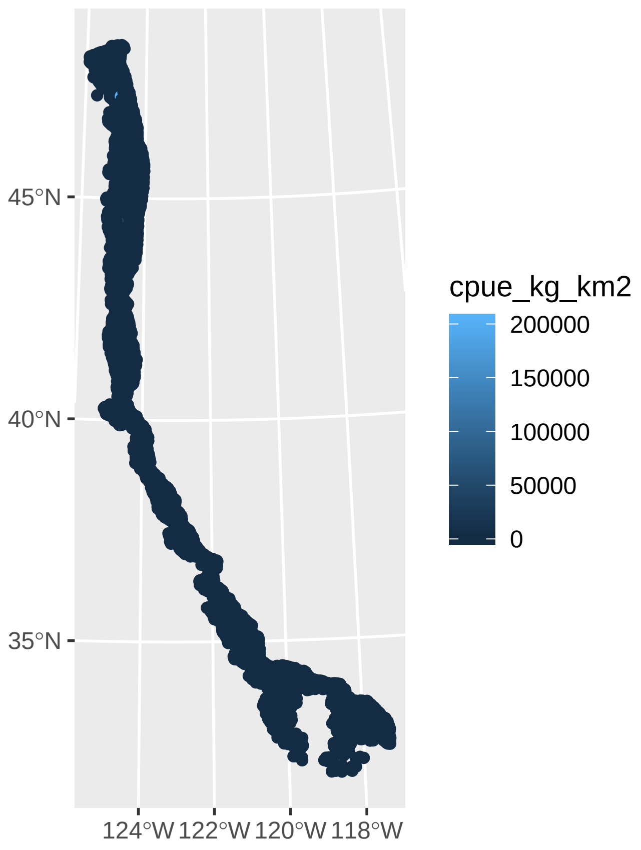
Working from the `sf` example, we can use base plot or ggplot to make some basic plots. In this case, we'll only show CPUE. The `plot` function in `sf` is the base plot version,

```r
plot(pop_utm[,"cpue_kg_km2"])
```



or with ggplot we can use the `geom_sf` geom,

```r
ggplot(data = pop_utm, aes(col=cpue_kg_km2)) +
    geom_sf()
```



Modeling options available

- GAMs
- random forest
- glmmfields
- INLA / VAST / sdmTMB

Working with output from data frames (gam, randomForest, glmmfields, sdmTMB, VAST)

GAMs and Random Forest Models

With output from mgcv or randomForest, the predict method returns a data frame. We'll fit some simple models with both and work with the predicted objects, and for simplicity just look at predicted data for the locations used to fit the model.

```
# fit random forest
rf_fit = randomForest(cpue_kg_km2 ~ lon + I(lon^2) + lat + I(lat^2) + year,
  data=pop_df)
rf_dat = pop_df
rf_dat$pred = predict(rf_fit, newdata=pop_df)

# fit GAM
gam_fit = gam(cpue_kg_km2 ~ s(lon,lat) + s(year),
  data=pop_df, family=tw(), method="REML")
```

```
gam_dat = pop_df
gam_dat$pred = predict(gam_fit, newdata=pop_df)
```

GAMs and random forests are both extremely fast. Random forests offer a benefit in not having to include interactions explicitly in the model arguments (the above example included linear and quadratic terms for lat and lon, but not the interaction). For the GAMs, there's a little more flexibility in how to include interactions. Examples include

- specifying a tensor smooth with te(). Note that te() includes the tensor smooth and interaction terms, wheras if you just want to include the interaction, you could use ti(). More info here

Including the main effects and the interaction separately could be done by including main effects with s() and interaction terms with ti().

- Specifying the number of knots

This is done by specifying the 'k' argument. Higher values of 'k' make the smooth more wiggly, e.g.

```
gam_coarse = gam(cpue_kg_km2 ~ s(lon,lat, k =c(10,10)) + s(year),
  data=pop_df, family=tw(), method="REML")

gam_smooth = gam(cpue_kg_km2 ~ s(lon,lat, k =c(50,50)) + s(year),
  data=pop_df, family=tw(), method="REML")
```

- Constructing separate smooths by year.

In this case, separate fields are fit by year. It's important to note that no structure is imposed on the fields year to year (e.g. AR(1)) - instead they're independent.

```
gam_year = gam(cpue_kg_km2 ~ s(lon,lat, by=year),
  data=pop_df, family=tw(), method="REML")
```

Here, both `rf_dat` and `gam_dat` are nearly identical to the original data frame, but have a column for predicted values (pred). Turning either of these into a `sp` or `sf` object is straightforward, e.g.

```
gam_dat_utm = st_as_sf(x = gam_dat,
  coords = c("lon", "lat"),
  crs = "+proj=utm +zone=10 +datum=WGS84 +units=km")
```

Working with output from glmmfields

The `glmmfields` package is similar to the output above in that the estimated model. This code isn't run (because of speed) but the idea is similar. The glmmfields::predict() function returns a dataframe (tibble) with columns for "estimate", "conf_low", and "conf_high".

```
gf_fit = glmmfields(cpue_kg_km2 ~ 1, time="year",
                    lon="lon", lat="lat",
                    data=pop_df, nknots = 12,
                    chains=1, iter=1000)
gf_dat = pop_df
gf_pred = glmmfields::predict(gf_fit, estimate_method="mean",
                              conf_level=0.95, interval="prediction",
                              type="response")
gf_dat = cbind(gf_dat, gf_pred)
```

Working with output from sdmTMB

`sdmTMB` is a newly developed package by Sean Anderson at Fisheries and Oceans Canada. This package uses TMB and INLA to do fast estimation for large spatiotemporal models. The `predict` function in `sdmTMB` is great in that it appends all of the quantities of interest to the original data frame.

In this case, the columns that are appended are "est" (estimates in link space), "est_non_rf" (estimate not including random field), "est_rf" (estimate from all random fields – spatial and spatiotemporal for example), "omega_s" (spatial intercept, constant through time), "zeta_s" (optional spatial slope random field), "epsilon_st" (optional spatiotemporal random field).

```
devtools::install_github("pbs-assess/sdmTMB")
```

```
## Skipping install of 'sdmTMB' from a github remote, the SHA1 (16484967) has not changed since last ins
##   Use `force = TRUE` to force installation
```

```
library(sdmTMB)
```

```
##
## Attaching package: 'sdmTMB'

## The following objects are masked from 'package:glmmfields':
##
##     lognormal, nbinom2
```

```
# make spde argument to pass in
pop_spde <- make_spde(pop_df$lon, pop_df$lat, n_knots = 50) # only 50 knots for speed

fit_sdm = sdmTMB(cpue_kg_km2 ~ 1,
                 data = pop_df, time="year",
                 spde = pop_spde)
```

```
## Warning in checkMatrixPackageVersion(): Package version inconsistency detected.
## TMB was built with Matrix version 1.2.15
## Current Matrix version is 1.2.17
## Please re-install 'TMB' from source using install.packages('TMB', type = 'source') or ask CRAN for a

## Warning: The model may not have converged: non-positive-definite Hessian
## matrix.
```

```
pred = predict(fit_sdm, newdata = pop_df,
        xy_cols = c("lon","lat"))
```

Working with output from VAST

Kelli Johnson (Northwest Fisheries Science Center) has put together a series of demos and utilities for using VAST with west coast species, so we'll use that repository.

```
devtools::install_github("nwfsc-assess/VASTWestCoast")
```

```
## Skipping install of 'VASTWestCoast' from a github remote, the SHA1 (2b6ead2e) has not changed since l
##   Use `force = TRUE` to force installation
```

```
library(VASTWestCoast)
```

```
## Loading required package: VAST

## ###########################################################################################

## Loading package VAST, developed by James Thorson for the Northwest Fisheries Science Center
## For details and citation guidance, please see http://github.com/james-thorson/VAST/

## ###########################################################################################

## Loading package `FishStatsUtils`

## Loading required package: maps

## Loading required package: mapdata

##
## Attaching package: 'FishStatsUtils'

## The following object is masked from 'package:sdmTMB':
##
##     plot_anisotropy
```

```
pop = get_data(survey="NWFSC.Combo",
               species="Sebastes alutus", data = NULL)
```

```
## Joining, by = "Trawl_id"

## Joining, by = c("Scientific_name", "Project", "Year", "Vessel", "Tow")
```

Calculating summaries and indicators from dataframes

Using any of the examples above, we can generate spatial summaries of metrics of interest (variability, synchrony, etc). One option for doing this is raster bricks / layers, which would let us change the scale quickly.

Maybe a downside of relying exclusively on these objects is that for some cases (Barnett et al. in prep) we'd want to do post-hoc clustering of metrics with the raw data frames from `predict`.

Questions / decisions

1. Each of the above estimation approaches generally returns a non-spatial data frame. It seems like the first step is to coerce these data frames to a common class for spatial manipulation/metrics. Thoughts on `sp` v `sf`? Other? Is there a better workflow for doing this?