Analyse fulmars from the seabirds-at-sea data

David L Miller, David A Fifield, Ewan Wakefield and Douglas B Sigourney

This vignette gives an example of modelling data from a seabirds-at-sea survey using the `dsm` package. The data here are for fulmars and were collected using a line transect for birds on the water and strip transect for flying birds.

```
library(Distance)
```

```
## Loading required package: mrds
```

```
## This is mrds 2.2.4
## Built: R 4.0.2; ; 2020-11-30 17:31:53 UTC; unix
```

```
##
## Attaching package: 'Distance'
```

```
## The following object is masked from 'package:mrds':
##
##     create.bins
```

```
library(dsm)
```

```
## Loading required package: mgcv
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-34. For overview type 'help("mgcv-package")'.
```

```
## Loading required package: numDeriv
```

```
## This is dsm 2.3.1
## Built: R 4.0.2; ; 2021-03-23 21:16:18 UTC; unix
```

```
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:nlme':
##
##     collapse
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(patchwork)
library(sf)
```

```
## Linking to GEOS 3.8.1, GDAL 3.1.4, PROJ 6.3.1
```

Then load the data, already in `dsm` format. See the vignette to Miller et al., (2013), the `?"dsm-data"` manual page and this guide for more details on this data format.

```
load("RData/fulmars.RData")
ls()
```

```
## [1] "dists_fly"  "dists_swim" "obs"        "pred"       "pred_sf"
## [6] "segs"
```

A summary of the loaded objects

- `dists_fly`: distance data for the flying fulmars, used to fit the detection function
- `dists_swim`: "distance" data for the fulmars on the water, the `distance` column is not used but the data is used to set up a dummy detection function
- `obs`: the observation data frame that matches the `dists_` data to the sample units (transect segments)
- `segs`: segment data, giving each sample location that was visited, the effort expended and the observation covariates collected at the segment level.
- `pred`: prediction grid.
- `pred_sf`: prediction grid in R `sf` format, for nicer printing (not essential)

Detection function analysis

First setting up the data for the swimming birds. Data are collected in bins, so we first need to set up the distance bins:

```
dbins <- c(0, 50, 100, 200, 300)
```

We can now fit detection functions for birds on the water:

```
df_swim_hn <- ds(dists_swim, key='hn', formula=~precip2+vis, adjustment=NULL,
                 cutpoints=dbins)
```

```
## Fitting half-normal key function
```

```
## AIC= 1030.036
```

```
## No survey area information supplied, only estimating detection function.
```

```
df_swim_hr <- ds(dists_swim, key='hr', formula=~precip2+vis, adjustment=NULL,
                 cutpoints=dbins)
```

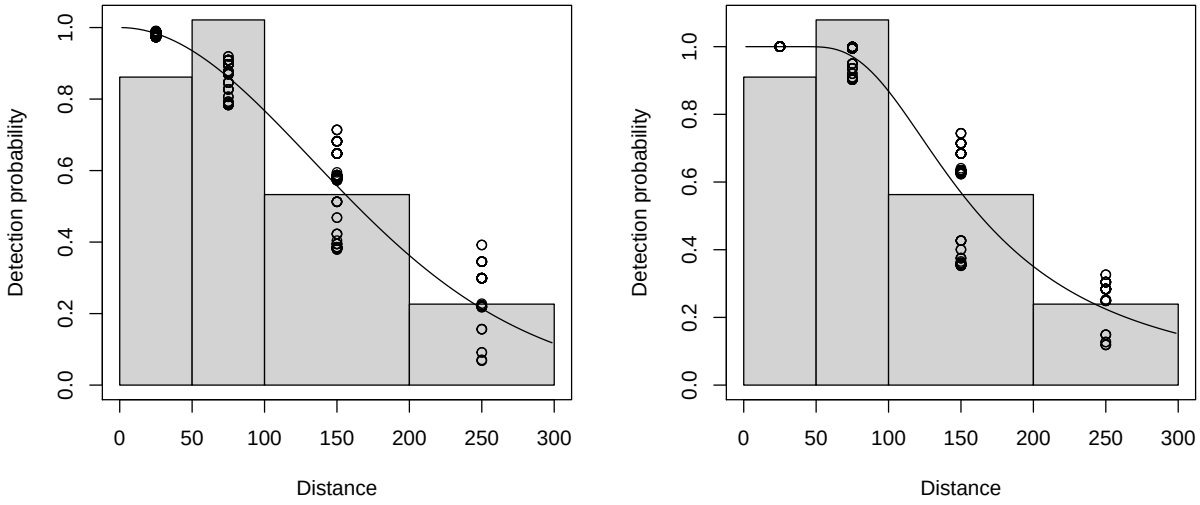
```
## Fitting hazard-rate key function
```

```
## AIC= 1031.668
```

```
## No survey area information supplied, only estimating detection function.
```

We can plot the fitted detection functions:

```
par(mfrow=c(1,2))
plot(df_swim_hn)
plot(df_swim_hr)
```



Comparing models by AIC, we see the half-normal is preferred (but not by much):

```
AIC(df_swim_hn, df_swim_hr)
```

```
##            df      AIC
## df_swim_hn  3 1030.036
## df_swim_hr  4 1031.668
```

(Note we are not able to do a χ^2 goodness-of-fit test as we do not have enough degrees of freedom, as the data are binned.)

We can look at the summary of that model:

```
summary(df_swim_hn)
```

```
##
## Summary for distance analysis
## Number of observations :  383
## Distance range         :  0  -  300
##
## Model : Half-normal key function
## AIC   : 1030.036
##
## Detection function parameters
## Scale coefficient(s):
##                estimate         se
## (Intercept)  4.95362188 0.10566336
## precip21    -0.27832397 0.11724187
## vis          0.01269992 0.01012255
##
##                         Estimate          SE         CV
## Average p              0.5668975  0.02649756 0.04674136
## N in covered region 675.6071023 39.04742187 0.05779605
```

We now construct the dummy detection function to incorporate birds on the water:

```
df_fly <- dummy_ddf(object=dists_fly$object, size=dists_fly$size, width=300)
```

We can use `summary` to check that the model is set-up correctly (at least the number of observations and truncation are correct):

```
summary(df_fly)
```

```
##
## Summary for dummy ds object
## Number of observations :  460
## Distance range         :  0  -  300
##
## Model : No detection function, strip transect
##
## AIC   : NA
```

Setup data

Now we need to setup the segment data to use the multi-platform approach. The first part of this consists of duplicating the segment data to have set of segments for flying birds and another for birds on the water. We also need to create a column indicating which detection function will be used for a given segment (`ddfobj`) and ensure that the `Sample.Label` is unique.

```
# first duplicate the segment table
segs2 <- rbind(segs, segs)

# create a ddfobj field to keep track of which detection function to use
segs2$ddfobj <- c(rep(1, nrow(segs)), rep(2, nrow(segs)))

# create a flying vs swimming, which has the same information as ddfobj
# but has a friendlier name for modelling
segs2$FlySwim <- factor(c(rep("swim", nrow(segs)),
                          rep("fly", nrow(segs))),
                        c("swim", "fly"))

# ensure that that Sample.Labels are unique, concatenating the current
# label with the ddfobj value
segs2$Sample.Label[segs2$ddfobj == 1] <- paste0(segs2$Sample.Label[segs2$ddfobj == 1],
                                                "-1")
segs2$Sample.Label[segs2$ddfobj == 2] <- paste0(segs2$Sample.Label[segs2$ddfobj == 2],
                                                "-2")
```

We now need to half the effort because only one side of the transect was observed:

```
segs2$Effort <- segs2$Effort/2
```

Our next task is to re-write the observation table to ensure that the `Sample.Labels` match those in the segment tables and include the `ddfobj` column as for the segment data.

```
obs$Sample.Label[obs$FlySwim == "W"] <- paste0(obs$Sample.Label[obs$FlySwim == "W"], "-1")
obs$Sample.Label[obs$FlySwim == "F"] <- paste0(obs$Sample.Label[obs$FlySwim == "F"], "-2")

# add ddfobj column
obs$ddfobj <- c(1,2)[(obs$FlySwim=="F") + 1]

# simplify columns
```

```
obs <- obs[, c("ddfobj", "Sample.Label", "size", "distance", "object")]
```

Now we have the data setup, we can start to fit models.

Fitting the spatial model

We start with the model with no extra terms accounting for platform:

```
dsm_nofactor <- dsm(count~s(x, y, k=100),
                 ddf.obj=list(df_swim_hn, df_fly),
                 segment.data=segs2, observation.data=obs, family=nb())
summary(dsm_nofactor)
```

```
##
## Family: Negative Binomial(1.481)
## Link function: log
##
## Formula:
## count ~ s(x, y, k = 100) + offset(off.set)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -16.750      0.486  -34.47   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df Chi.sq p-value
## s(x,y) 39.84  49.93  560.6  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.404   Deviance explained = 63.4%
## -REML = 1294.6  Scale est. = 1         n = 1932
```

Note here that we include a `list` of detection functions. The order of the list relates to the `ddfobj` column in the data (`df_swim_hn` is `ddfobj==1` and `df_fly` is `ddfobj==2`).

Now with a factor for on water/flying:

```
dsm_factor <- dsm(count~ FlySwim + s(x, y, k=100, bs="ts"),
                  ddf.obj=list(df_swim_hn, df_fly),
                  segment.data=segs2, observation.data=obs, family=nb())
summary(dsm_factor)
```

```
##
## Family: Negative Binomial(1.664)
## Link function: log
##
## Formula:
## count ~ FlySwim + s(x, y, k = 100, bs = "ts") + offset(off.set)
##
## Parametric coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -16.42569    0.46707 -35.168  < 2e-16 ***
## FlySwimfly   -0.52787    0.08727  -6.049 1.46e-09 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df Chi.sq p-value
## s(x,y) 39.28     99  571.3  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.429   Deviance explained = 64.6%
## -REML = 1291.9  Scale est. = 1         n = 1932
```

And finally the factor-smooth interaction

```
dsm_factorsmooth <- dsm(count~ s(x, y, FlySwim, bs="fs", k=100),
                        ddf.obj=list(df_swim_hn, df_fly),
                        segment.data=segs2, observation.data=obs, family=nb())
summary(dsm_factorsmooth)
```

```
##
## Family: Negative Binomial(1.863)
## Link function: log
##
## Formula:
## count ~ s(x, y, FlySwim, bs = "fs", k = 100) + offset(off.set)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -2.732     22.511  -0.121    0.903
##
## Approximate significance of smooth terms:
##                  edf Ref.df Chi.sq p-value
## s(x,y,FlySwim) 65.43    199  630.2  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.449   Deviance explained = 66.9%
## -REML = 1296.6  Scale est. = 1         n = 1932
```

Propagate uncertainty

We can now use the `dsm_varprop` function to propagate the variance from the detection function through to the spatial model using the method of Bravington et al. (2021). The returned object has a `$refit` element which is a density surface model with a covariance matrix including the uncertainty from the detection function. We don't supply any data (which prompts a warning from `dsm_varprop`) since we are only interested in the refitted model.

```
vp_nofactor <- dsm_varprop(dsm_nofactor)
```

```
## Warning in dsm_varprop(dsm_nofactor): No newdata provided, no predictions
## returned.
```

```
vp_factor <- dsm_varprop(dsm_factor)
```

```
## Warning in dsm_varprop(dsm_factor): No newdata provided, no predictions
## returned.
```

```
vp_fs <- dsm_varprop(dsm_factorsmooth)
```

```
## Warning in dsm_varprop(dsm_factorsmooth): No newdata provided, no predictions
## returned.
```

We can now look at comparing the observed versus the expected values and plot them in a table:

```
knitr::kable(cbind(t(obs_exp(vp_nofactor$refit, "FlySwim")),
                   t(obs_exp(vp_factor$refit, "FlySwim"))[, 2, drop=FALSE],
                   t(obs_exp(vp_fs$refit, "FlySwim"))[, 2, drop=FALSE]),
             format="latex", digits=0,
             col.names=c("Observed", "Null",
                         "Factor", "Factor-smooth"))
```

	Observed	Null	Factor	Factor-smooth
swim	501	405	501	490
fly	533	632	516	517

We can also compare AIC of these models:

```
AIC(vp_nofactor$refit, vp_factor$refit, vp_fs$refit)
```

```
##                          df      AIC
## vp_nofactor$refit 43.60258 2515.726
## vp_factor$refit   43.76449 2495.230
## vp_fs$refit       69.30821 2483.713
```

So by AIC we prefer the factor-smooth model.

Prediction

Now we can make predictions for each of these models. This code is a little complex as it involved binding the predictions onto `pred_sf` which is a spatial object with projection information. We create a `data.frame` that includes combined and uncombined results for the factor and factor-smooth models.

```
# create an extra column to account for the variance propagation model
pred$XX <- matrix(0, nrow=nrow(pred), ncol=3)

# nofactor model
pred_nofactor <- pred
pred_nofactor$Nhat <- predict(vp_nofactor$refit, newdata=pred_nofactor,
                              off.set=pred_nofactor$area)
pred_nofactor$model <- "No factor"
pred_nofactor$subset <- "Combined"

# bind on the spatial data
pred_nofactor_plots <- st_sf(pred_sf, pred_nofactor[,c("model","subset","Nhat")])


# factor model
pred_factor <- rbind(pred, pred)
pred_factor$FlySwim <- factor(rep(c("fly", "swim"), c(nrow(pred), nrow(pred))),
                              c("swim","fly"))
pred_factor$Nhat <- predict(vp_factor$refit, newdata=pred_factor,
                            off.set=pred_factor$area)
pred_factor$model <- "Factor"
pred_factor$subset <- as.character(pred_factor$FlySwim)
```

```
# bind on the spatial data
pred_factor_plots <- st_sf(pred_sf, pred_factor[,c("model", "subset", "Nhat")])
```

```
## Warning in data.frame(..., check.names = FALSE): row names were found from a
## short variable and have been discarded
```

```
# prepare the combined/difference columns
pred_factor_plots2 <- rbind(pred_factor_plots, pred_factor_plots)
pred_factor_plots2$subset <- rep(c("Combined", "Difference"),
                                 c(nrow(pred), nrow(pred)))
# combined estimate
pred_factor_plots2$Nhat[pred_factor_plots2$subset=="Combined"] <-
  (pred_factor_plots$Nhat[1:nrow(pred)] +
   pred_factor_plots$Nhat[(nrow(pred)+1):(2*nrow(pred))])/2
# difference in estimates
pred_factor_plots2$Nhat[pred_factor_plots2$subset=="Difference"] <-
  pred_factor_plots$Nhat[1:nrow(pred)] -
   pred_factor_plots$Nhat[(nrow(pred)+1):(2*nrow(pred))]

# bind them together
pred_factor_plots <- rbind(pred_factor_plots, pred_factor_plots2)


# factor-smooth model
pred_fs <- pred_factor
pred_fs$Nhat <- predict(vp_fs$refit, newdata=pred_fs,
                        off.set=pred_factor$area)
pred_fs$model <- "Factor-smooth"
pred_fs$subset <- as.character(pred_fs$FlySwim)

# bind on the spatial data
pred_fs_plots <- st_sf(pred_sf, pred_fs[,c("model", "subset", "Nhat")])
```

```
## Warning in data.frame(..., check.names = FALSE): row names were found from a
## short variable and have been discarded
```

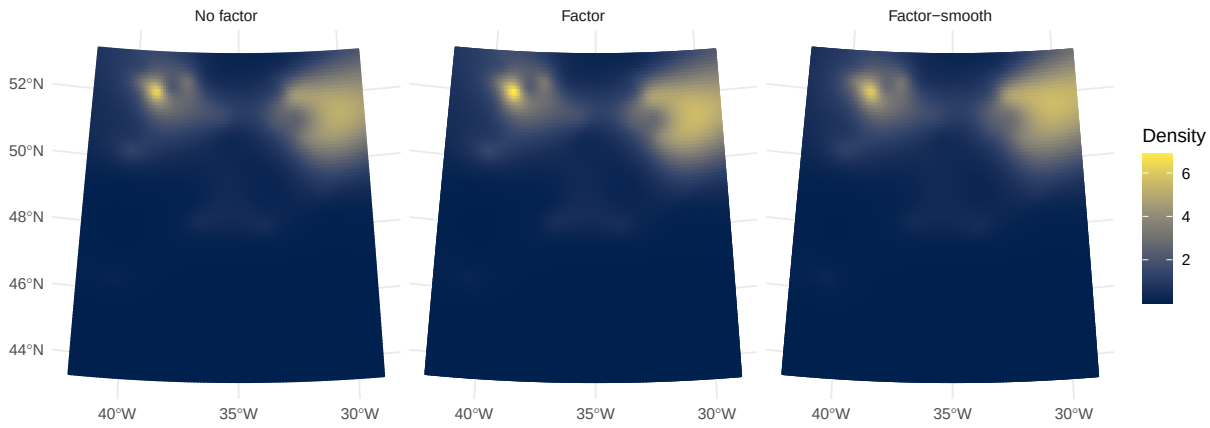
```
# prepare the combined/difference columns
pred_fs_plots2 <- rbind(pred_fs_plots, pred_fs_plots)
pred_fs_plots2$subset <- rep(c("Combined", "Difference"),
                             c(nrow(pred), nrow(pred)))

# combined estimate
pred_fs_plots2$Nhat[pred_fs_plots2$subset=="Combined"] <-
  (pred_fs_plots$Nhat[1:nrow(pred)] +
   pred_fs_plots$Nhat[(nrow(pred)+1):(2*nrow(pred))])/2
# difference in estimates
pred_fs_plots2$Nhat[pred_fs_plots2$subset=="Difference"] <-
  pred_fs_plots$Nhat[1:nrow(pred)] -
   pred_fs_plots$Nhat[(nrow(pred)+1):(2*nrow(pred))]

# bind them together
pred_fs_plots <- rbind(pred_fs_plots, pred_fs_plots2)
```

Once we've put all this together, we calculate a density for plotting (by dividing by grid cell area

```
# concatenate everything
all_Nhats <- rbind(pred_nofactor_plots, pred_factor_plots, pred_fs_plots)
# convert abundance to density
all_Nhats$Density <- all_Nhats$Nhat/10^2

# reorder rows to give preferred plotting order
all_Nhats$model <- factor(all_Nhats$model,
                          c("No factor", "Factor", "Factor-smooth"))
```

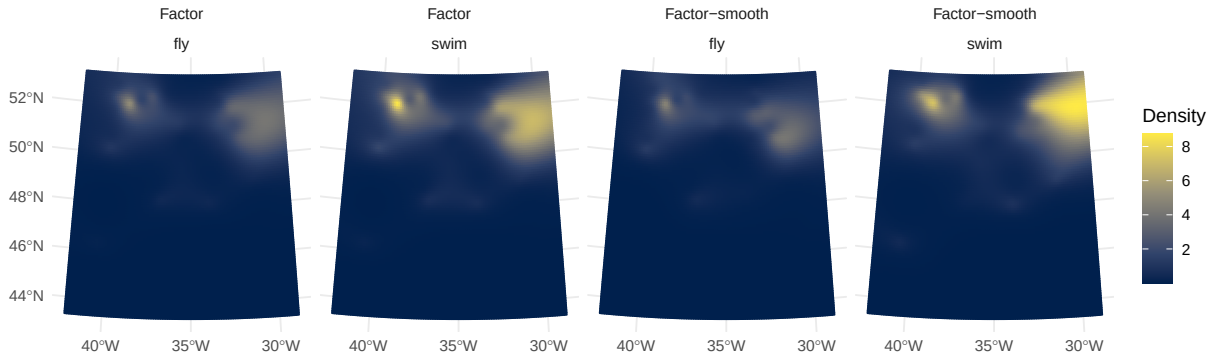
We can now plot of the combined results:

```
combined_data <- subset(all_Nhats, subset=="Combined")

combined_plot <- ggplot() +
  geom_sf(data=combined_data, aes(colour=Density, fill=Density)) +
  facet_wrap(~model, drop=TRUE, nrow=1) +
  labs(x="", y="", fill="Density") +
  theme_minimal() +
  scale_colour_viridis_c(option="E") +
  scale_fill_viridis_c(option="E")
combined_plot
```



And the uncombined predictions:

```
uncombined_data <- subset(all_Nhats, subset %in% c("swim", "fly"))

uncombined_plot <- ggplot() +
  geom_sf(data=uncombined_data, aes(colour=Density, fill=Density)) +
  facet_wrap(model~subset, drop=TRUE, nrow=1) +
  labs(x="", y="", fill="Density") +
  theme_minimal() +
  scale_colour_viridis_c(option="E") +
  scale_fill_viridis_c(option="E")
uncombined_plot
```



Estimating variance

To estimate variance of the predictions we first need to construct the matrix that maps the parameters to the predictions on the linear predictor scale, $\mathbf{X}_p$.

```
lp_nofactor <- predict(vp_nofactor$refit, newdata=pred, type="lpmatrix")
lp_factor <- predict(vp_factor$refit, newdata=pred_factor, type="lpmatrix")
lp_fs <- predict(vp_fs$refit, newdata=pred_factor, type="lpmatrix")
```

We can then generate parameter samples using the Metropolis-Hasting sampler in `mgcv`. We need to use tools from this github repo to ensure that the sampling works for the variance-propagated DSM.

```
# load additional code
source("likelihood_tools.R")
source("gam.mh_fix.R")
source("ttools.R")
samples_nofactor <- gam.mh(vp_nofactor$refit, ns=10000, burn=1000, t.df=20,
                           rw.scale=.05, thin=10)
samples_factor <- gam.mh(vp_factor$refit, ns=10000, burn=1000, t.df=20,
                         rw.scale=.05, thin=10)
samples_fs <- gam.mh(vp_fs$refit, ns=10000, burn=1000, t.df=20,
                     rw.scale=.025, thin=10)
```

We can now make grids of abundance estimates per cell per posterior sample:

```
nofactor_Ngrid <- pred_nofactor$area * exp(lp_nofactor %*% t(samples_nofactor$bs))
factor_Ngrid <- pred_factor$area * exp(lp_factor %*% t(samples_factor$bs))
fs_Ngrid <- pred_factor$area * exp(lp_fs %*% t(samples_fs$bs))
```

Once we have the per grid cell, per sample results we can build the data for plotting. This involves taking the empirical variance of these predictions over the samples.

```
all_Nhats$var <- NA
all_Nhats$var[all_Nhats$model == "No factor" &
              all_Nhats$subset == "Combined"] <- apply(nofactor_Ngrid, 1, var)
```

```
# for the factor model, we have the first 10000 entries for the first level
# of the factor and the second 10000 for the second level, we need a variance
# for each of those subsets
all_Nhats$var[all_Nhats$model == "Factor" &
              all_Nhats$subset == "fly"] <- apply(factor_Ngrid[1:10000,], 1, var)
all_Nhats$var[all_Nhats$model == "Factor" &
              all_Nhats$subset == "swim"] <- apply(factor_Ngrid[10001:20000,],
                                                   1, var)
all_Nhats$var[all_Nhats$model == "Factor" &
              all_Nhats$subset == "Combined"] <-  apply((factor_Ngrid[1:10000,] +
                            factor_Ngrid[10001:20000,])/2, 1, var)
all_Nhats$var[all_Nhats$model == "Factor" &
              all_Nhats$subset == "Difference"] <-
  sqrt(all_Nhats$var[all_Nhats$model == "Factor" &
                     all_Nhats$subset == "fly"])-
  sqrt(all_Nhats$var[all_Nhats$model == "Factor" &
                     all_Nhats$subset == "swim"])

# same with the factor-smooth model
all_Nhats$var[all_Nhats$model == "Factor-smooth" &
              all_Nhats$subset == "fly"] <- apply(factor_Ngrid[1:10000,], 1, var)
all_Nhats$var[all_Nhats$model == "Factor-smooth" &
              all_Nhats$subset == "swim"] <- apply(factor_Ngrid[10001:20000,],
                                                   1, var)
all_Nhats$var[all_Nhats$model == "Factor-smooth" &
              all_Nhats$subset == "Combined"] <-  apply((factor_Ngrid[1:10000,] +
                            factor_Ngrid[10001:20000,])/2, 1, var)
all_Nhats$var[all_Nhats$model == "Factor-smooth" &
              all_Nhats$subset == "Difference"] <-
  sqrt(all_Nhats$var[all_Nhats$model == "Factor-smooth" &
                     all_Nhats$subset == "fly"])-
  sqrt(all_Nhats$var[all_Nhats$model == "Factor-smooth" &
                     all_Nhats$subset == "swim"])

# make a CV column
all_Nhats$CV <- (sqrt(all_Nhats$var))/all_Nhats$Nhat
```

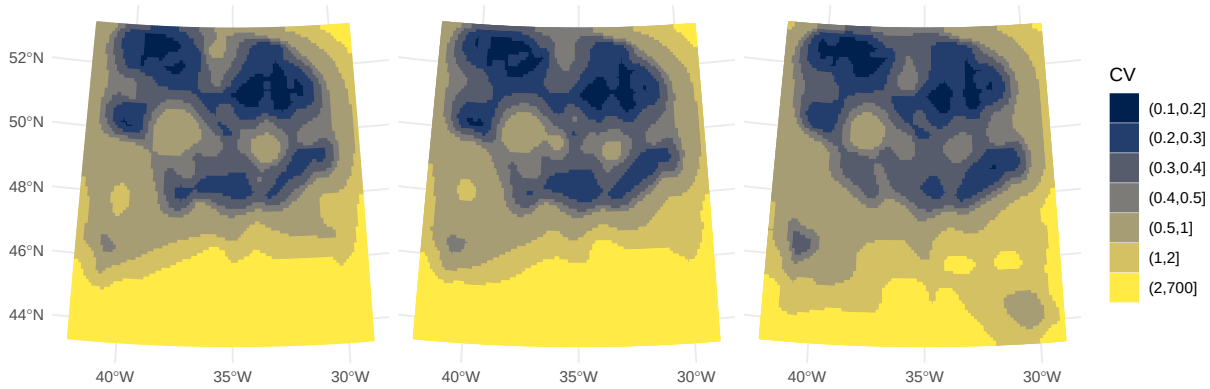
```
## Warning in sqrt(all_Nhats$var): NaNs produced
```

```
# discretize
all_Nhats$CV_d <- cut(all_Nhats$CV,
                      c(0, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 700))
```

Now we can make a plot of the combined effects

```
combined_CV_data <- subset(all_Nhats, subset=="Combined")

combined_CV_plot <- ggplot() +
  geom_sf(data=combined_CV_data, aes(fill=CV_d, colour=CV_d)) +
  #geom_point(data=segs_plot, alpha=0.2, shape=3, colour="grey60")+
  #facet_grid(model~subset, drop=TRUE) +
  facet_wrap(~model, drop=TRUE, nrow=1) +
  labs(x="", y="", fill="CV", colour="CV") +
  theme_minimal() +
```

```
  theme(strip.text.x = element_blank()) +
  scale_fill_viridis_d(option="E") +
  scale_colour_viridis_d(option="E")
combined_CV_plot
```




```
# write out the plot for the paper
ggsave(combined_plot / combined_CV_plot, file="figures/fulmar_combined.pdf",
       width=13, height=9)
```

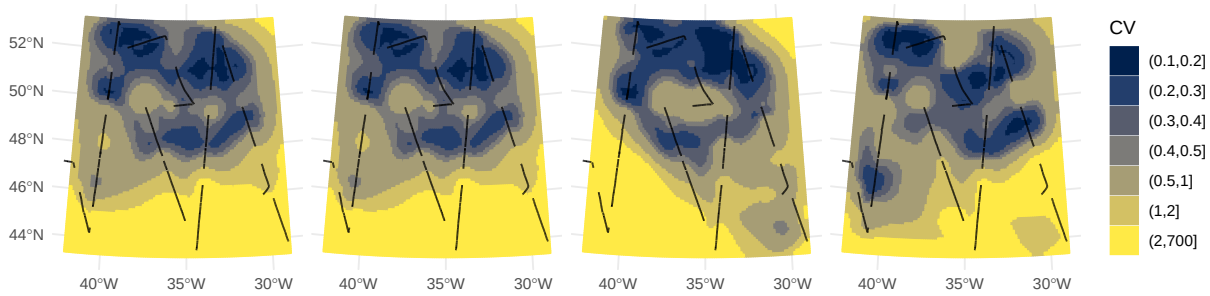
And the uncombined CV plots:

```
# load track to overplot
load("RData/track_sf.RData")
track_sf <- st_transform(track_sf, st_crs(pred_sf))
track_sf <- st_crop(track_sf, pred_sf)
```

```
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
```

```
uncombined_CV_data <- subset(all_Nhats, subset %in% c("swim", "fly"))

uncombined_CV_plot <- ggplot() +
  geom_sf(data=uncombined_CV_data, aes(fill=CV_d, colour=CV_d)) +
  geom_sf(data=track_sf) +
  facet_wrap(model~subset, drop=TRUE, nrow=1) +
  labs(x="", y="", fill="CV", colour="CV") +
  theme_minimal() +
  theme(strip.text.x=element_blank()) +
  scale_fill_viridis_d(option="E") +
  scale_colour_viridis_d(option="E")
uncombined_CV_plot
```

```
## Warning in strip_mat[panel_pos] <- unlist(unname(strips), recursive = FALSE)
## [[params$strip.position]]: number of items to replace is not a multiple of
## replacement length
```

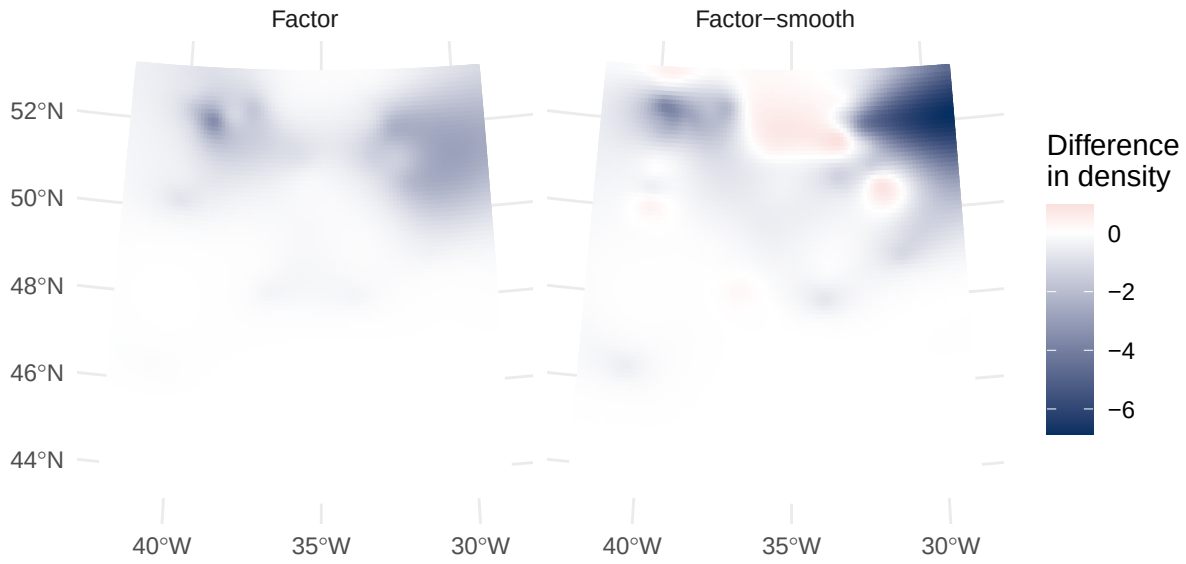



```
# write out the plot for the paper
ggsave(uncombined_plot / uncombined_CV_plot, file="figures/fulmar_uncombined.pdf",
       width=13, height=7.5)
```

```
## Warning in strip_mat[panel_pos] <- unlist(unname(strips), recursive = FALSE)
## [[params$strip.position]]: number of items to replace is not a multiple of
## replacement length
```

Finally we can plot the differences between the platforms within each model:

```
diff_data <- subset(all_Nhats, subset=="Difference")

diff_plot <- ggplot() +
  geom_sf(data=diff_data, aes(fill=Nhat/10^2, colour=Nhat/10^2)) +
  facet_wrap(~model, drop=TRUE, nrow=1) +
  labs(x="", y="", fill="Difference\nin density", colour="Difference\nin density") +
  theme_minimal() +
  scale_fill_gradient2(low="#053061", high="#b2182b") +
  scale_colour_gradient2(low="#053061", high="#b2182b")
diff_plot
```




```
# write out the plot for the paper
ggsave(diff_plot, file="figures/fulmar_diff.pdf", width=13, height=7.5)
```