Petrel SSF analysis

April 29, 2025

This document presents the analysis of a GPS data set collected on Antarctic petrels by Descamps et al. (2016), "At-sea distribution and prey selection of Antarctic petrels and commercial krill fisheries", Movebank Data Repository (https://doi.org/10.5441/001/1.q4gn4q56). We are very grateful to the authors for sharing their data publicly.

Set-up and data

First, we load the required packages: `terra` and `amt` for data processing, `mgcv` for model fitting, and `gratia` for visualisation.

```r
library(terra)
library(amt)
library(mgcv)
library(gratia)
library(ggplot2)
library(CircStats)
theme_set(theme_bw())
```

Then, we load the petrel data. The data has column for petrel ID, time, and the spatial coordinates in easting/northing (x, y). The original data had several individuals, but we are only looking at a single petrel (PET-C).

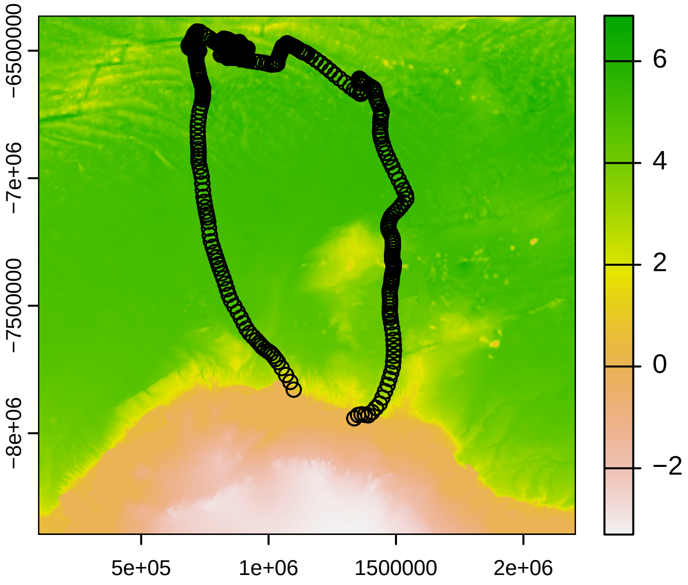
```r
# load processed locations abd subset to single individual
data <- read.csv("../data/petrel.csv")
head(data)
```

```
     X    ID                time       x        y
1 1109 PET-C 2011-12-14 19:54:48 1098759 -7829087
2 1110 PET-C 2011-12-14 20:24:48 1085342 -7798888
3 1111 PET-C 2011-12-14 20:54:48 1068499 -7771682
4 1112 PET-C 2011-12-14 21:24:48 1051679 -7744494
```

```
5 1113 PET-C 2011-12-14 21:54:48 1036662 -7720786
6 1114 PET-C 2011-12-14 22:24:48 1027830 -7709081
```

We obtained a bathymetry raster from the GEBCO project (https://www.gebco.net/). We transform it to depth (negative bathymetry) and divide by 1000 to convert to kilometers. Larger values of the covariate now correspond to deeper areas, and negative values are above sea level. Plotting the petrel track over a map reveals that the bird started close to the Antarctic continental shelf, travelled North towards deeper waters, and later returned towards its starting point.

```r
# load depth and convert to km
depth <- rast("../data/bathymetry_raster.tif")
depth <- - depth / 1000
names(depth) <- "depth"

# map of depth and petrel locations
plot(depth); points(data$x, data$y)
```



Data processing

To implement a step selection analysis, we need to generate random points on the landscape, and we will use the R package `amt` for this. The workflow for that package requires that we first convert the data to a specific format, where each row corresponds to a movement step rather than a single location. Then, `x1_` and `y1_` are the start points of the step, `x2_` and `y2_` are the end points, `sl_` is the turning angle, and `ta_` is the turning angle.

```r
# format times
data$time <- as.POSIXct(data$time)

# make amt track formatted as steps
data <- make_track(data, .x = x, .y = y, .t = time) |>
  steps()

# look at data
head(data)
```

```
# A tibble: 6 x 10
      x1_    x2_     y1_     y2_    sl_ direction_p      ta_ t1_
*   <dbl>  <dbl>   <dbl>   <dbl>  <dbl>       <dbl>    <dbl> <dttm>
1  1.10e6 1.09e6 -7.83e6 -7.80e6 33045.        1.99 NA       2011-12-14 19:54:48
2  1.09e6 1.07e6 -7.80e6 -7.77e6 31998.        2.13  1.36e-1 2011-12-14 20:24:48
3  1.07e6 1.05e6 -7.77e6 -7.74e6 31970.        2.12 -2.67e-4 2011-12-14 20:54:48
4  1.05e6 1.04e6 -7.74e6 -7.72e6 28064.        2.14  1.06e-2 2011-12-14 21:24:48
5  1.04e6 1.03e6 -7.72e6 -7.71e6 14662.        2.22  8.18e-2 2011-12-14 21:54:48
6  1.03e6 1.02e6 -7.71e6 -7.70e6 14657.        2.22 -1.71e-4 2011-12-14 22:24:48
# i 2 more variables: t2_ <dttm>, dt_ <drtn>
```

Generating random locations

One goal of this analysis is to model a non-parametric movement kernel. Therefore, we will sample spatially uniform random points on a disc with some chosen radius R (rather than sampling from a parametric distribution of step lengths and/or turning angles, for example). Specifically, the steps are:

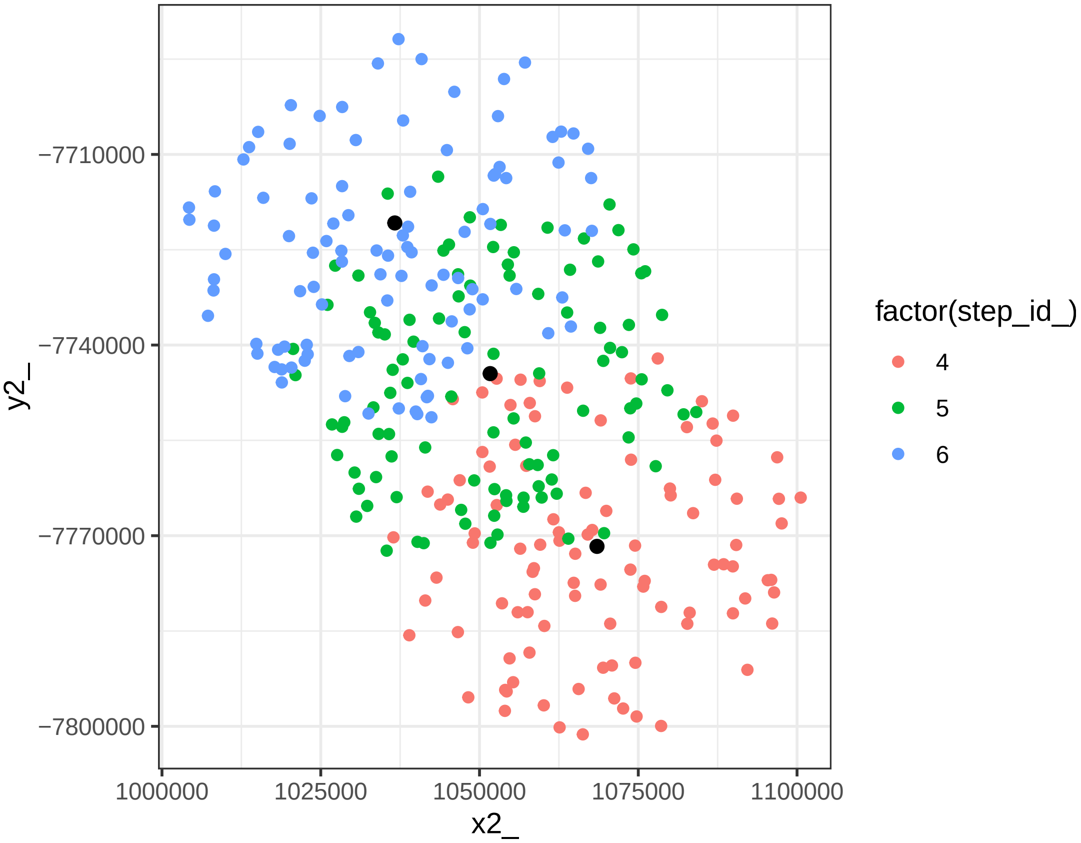
1. define R as the maximum observed step length
2. sample each step length as the square root of a random draw of $\text{Unif}(0, R^2)$
3. sample each turning angle from $\text{Unif}(-\pi, \pi)$

We can do this in `amt` by generating many possible step lengths and turning angles, and sampling from these in the `random_steps` function. The choice of the number of random locations is a trade-off between numerical stability and computational speed, and here we use 100 random locations for each observed step. This is more than many SSF analyses to ensure adequate coverage of the discs.

```r
# simulate many random distances and turning angles
set.seed(25)
rmax <- max(data$sl_)
sl_star <- sqrt(runif(n = 1e5, 0, rmax^2))
ta_star <- runif(n = 1e5, -pi, pi)

# add random locations to data
data <- random_steps(data, n_control = 100,
                     rand_sl = sl_star,
                     rand_ta = ta_star)

# view data
print(data, n = 6, width = Inf)
```

```
# A tibble: 103,828 x 11
       x1_      x2_       y1_       y2_    sl_    ta_ t1_
*    <dbl>    <dbl>     <dbl>     <dbl>  <dbl>  <dbl> <dttm>
1 1085342. 1068499. -7798888. -7771682. 31998.  0.136 2011-12-14 20:24:48
2 1085342. 1079424. -7798888. -7790047. 10639.  0.172 2011-12-14 20:24:48
3 1085342. 1101258. -7798888. -7825513. 31019. -3.02  2011-12-14 20:24:48
4 1085342. 1090482. -7798888. -7780728. 18874. -0.694 2011-12-14 20:24:48
5 1085342. 1107206. -7798888. -7819999. 30392. -2.76  2011-12-14 20:24:48
6 1085342. 1066170. -7798888. -7779272. 27429.  0.356 2011-12-14 20:24:48
  t2_                 dt_     case_ step_id_
* <dttm>              <drtn>  <lgl>    <dbl>
1 2011-12-14 20:54:48 30 mins TRUE         3
2 2011-12-14 20:54:48 30 mins FALSE        3
3 2011-12-14 20:54:48 30 mins FALSE        3
4 2011-12-14 20:54:48 30 mins FALSE        3
5 2011-12-14 20:54:48 30 mins FALSE        3
6 2011-12-14 20:54:48 30 mins FALSE        3
# i 103,822 more rows
```

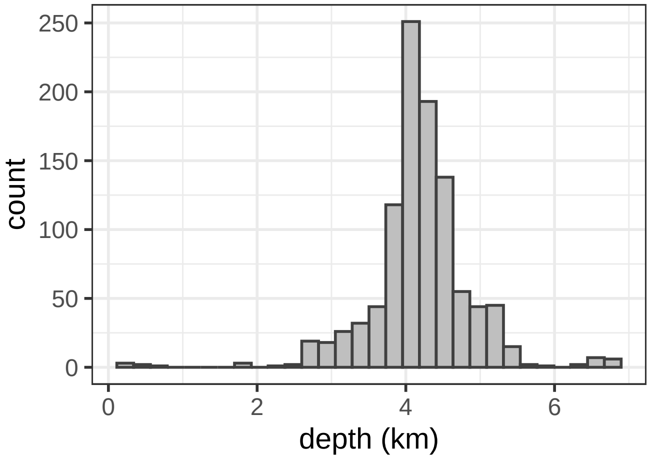
```r
# plot subset of data to visualise random steps
ggplot(subset(data, step_id_ %in% 4:6 & case_ == FALSE),
       aes(x = x2_, y = y2_, color = factor(step_id_))) +
  geom_point() +
  geom_point(aes(x = x2_, y = y2_),
             data = subset(data, step_id_ %in% 3:5 & case_ == TRUE),
             col = 1, size = 2) +
  coord_equal()
```



Extracting and processing covariates

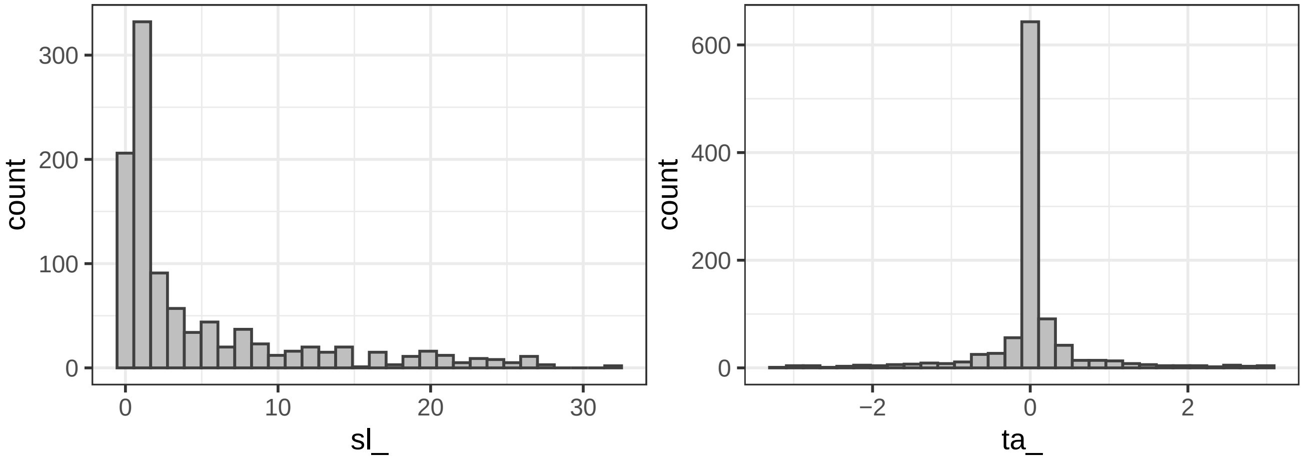
We also want to extract the bathymetry covariate at each (observed and random) location. There are many ways to do this, such as `extract` in the `terra` package. Here, we continue the workflow in `amt` by using the function `extract_covariates` to extract the values at the end of each step.

```
# get bathymetry values at each point
data <- extract_covariates(x = data, covariates = depth, where = "end")

# look at histogram to get a sense of covariate range at observed locs
obs <- subset(data, case_ == 1)
ggplot(obs, aes(x = depth)) +
  geom_histogram(fill = "grey75", color = "grey25") + xlab("depth (km)")
```



We also convert step length from meters to kilometers, such that all covariates have a similar range (and for interpretability). Let's plot the empirical distributions of step lengths and turning angles. It's notable that both distributions have heavy tails, which may be difficult to capture with the available parametric distributions within the exponential family (i.e., von Mises for turning angles or exponential/gamma/log-normal for step lengths).

```
data$sl_ <- data$sl_ / 1000
obs <- subset(data, case_ == 1)

ggplot(obs, aes(x = sl_)) + geom_histogram(fill = "grey75", color = "grey25")
ggplot(obs, aes(x = ta_)) + geom_histogram(fill = "grey75", color = "grey25")
```



Model 1: linear covariate effects

We first describe a simple linear SSF, and show how it can be formulated with mgcv syntax. For all models, we first need to define a dummy times column that is constant across all locations. This variable does not have an interpretation in the context of an SSF, but is required by mgcv for the implementation as a Cox proportional hazards (Cox PH) model. We can set this dummy variable to 1 for all locations.

```
# format for mgcv
data$dummy_times <- 1
```

The dummy time column, in conjunction with the stratum ID (`step_id_`), forms the response of the Cox PH model. We use the `weights` argument to specify the column which encodes whether the location is an observed or random location (`case_`), and we choose `family = cox.ph`. This is the general syntax that will allow us to fit an SSF as a Cox PH model in `mgcv`.

For the movement component of the model, we choose to model step length with a gamma distribution with fixed shape ($a = 2$) and unknown scale b; this can be specified by including the step length as a covariate in the linear predictor of the model. We choose a von Mises distribution (with a mean of 0 or π) by including the cosine of the turning angle as another covariate. Finally, we assess selection for depth by adding it to the formula as a linear effect. Later, we will consider non-linear relationships.

```
fit_linear <- gam(cbind(dummy_times, step_id_) ~
                    sl_ +
                    cos(ta_) +
                    depth,
                  data = data,
                  family = cox.ph,
                  weights = case_)
```

This model is quick to fit, and could have also been fitted with other software such as `clogit` in the `survival` package or in the `glmmTMB` package. Let's look at the outputs.

```
summary(fit_linear)
```

```

Family: Cox PH
Link function: identity

Formula:
cbind(dummy_times, step_id_) ~ sl_ + cos(ta_) + depth

Parametric coefficients:
          Estimate Std. Error z value Pr(>|z|)
```

```
sl_       -0.314194   0.007916 -39.690   <2e-16 ***
cos(ta_)   3.049048   0.134132  22.732   <2e-16 ***
depth     -0.285901   0.134989  -2.118   0.0342 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


Deviance explained = 60.1%
-REML = 1700.3  Scale est. = 1         n = 103828
```

All covariate effects are statistically significant. The coefficient for depth is negative ($\beta_{depth} = -0.286$), indicating overall avoidance of deeper waters. We can calculate the relative selection strength by taking the exponential of the coefficient, $\exp(\beta_{depth})$. This indicates that, holding all else equal, the petrel is 0.75 times as likely to take a step for each increase of 1km in depth.

```r
# effect of depth
exp(coefficients(fit_linear)["depth"])
```

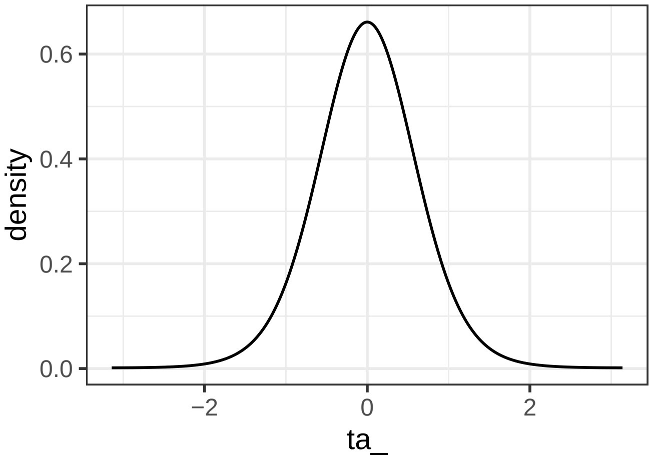
```
    depth
0.7513373
```

The coefficient for the cosine of the turning angle corresponds to the angular concentration parameter (κ) of the von Mises distribution. Here, we see it is 3.05, indicating relatively strong directional persistence. Since the coefficient is positive, this indicates a mean of 0 (which is nearly always the case).

```r
# angular concentration
kappa <- as.numeric(coefficients(fit_linear)["cos(ta_)"])
kappa
```

```
[1] 3.049048
```

```r
# plot angle distribution
angle_grid <- seq(-pi, pi, by = 0.01)
ang_df <- data.frame(ta_ = angle_grid, density = dvm(angle_grid, mu = 0, kappa = kappa))
ggplot(ang_df, aes(x = ta_, y = density)) + geom_line()
```

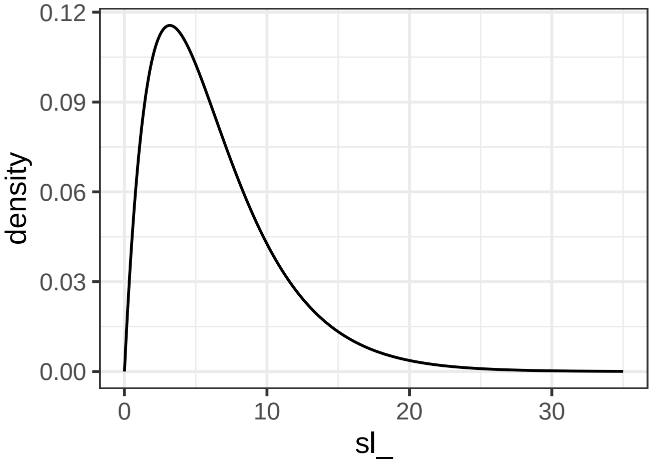


The coefficient for step length is slightly harder to interpret, but it can be used to derive the estimated distribution of step lengths. As mentioned, we chose a gamma distribution with shape $a = 2$ and scale b. We can derive b from the estimated coefficient: $b = -1/\beta_{sl}$. In turn, we can derive the mean and standard deviation of the step length distribution with standard formulas known for the gamma distribution; we find a mean step length of 6.4 km per 30 min.

```r
beta_sl <- as.numeric(coefficients(fit_linear)["sl_"])
shape <- 2
scale <- - 1 / beta_sl

# derive mean and standard deviation of sl distribution
mean_step <- shape * scale
sd_step <- sqrt(shape) * scale
c(mean_step, sd_step)
```

```
[1] 6.365492 4.501083
```

```r
# make plot
step_grid <- seq(0, 35, by= 0.1)
sl_df <- data.frame(sl_ = step_grid, density = dgamma(step_grid, shape = shape, scale =
ggplot(sl_df, aes(x = sl_, y = density)) + geom_line()
```



Model 2: smooth covariate effects

We can fit the same model, but with all terms as smooths. That is, we model the movement kernel by including smooth relationships between selection and movement variables (step length and turning angle). We use cyclic splines (`bs = "cc"`) for turning angle, as it is a circular variable. The `knots` argument sets the outer knots (i.e., the range) of the cyclic variable.

```
fit_smooth <- gam(cbind(dummy_times, step_id_) ~
                    s(sl_) +
                    s(ta_, bs = "cc") +
                    s(depth),
                  data = data,
                  knots = list(ta_ = c(-pi, pi)),
                  family = cox.ph,
                  weights = case_)
```

The model summary gives approximate *p*-values and effective degrees of freedom for the estimated relationships. Like for other mgcv models, it also includes percentage of deviance explained, but it is not a meaningful output here because the Cox PH model is only a trick to fit the SSF (notice the deviance explained is even negative here). We also do not recommend attempting to interpret residual plots from `gam.check()` or `appraise()`, as these are designed to test assumptions of the Cox PH model which are not relevant for SSFs.

```
summary(fit_smooth)
```

```
Family: Cox PH
```

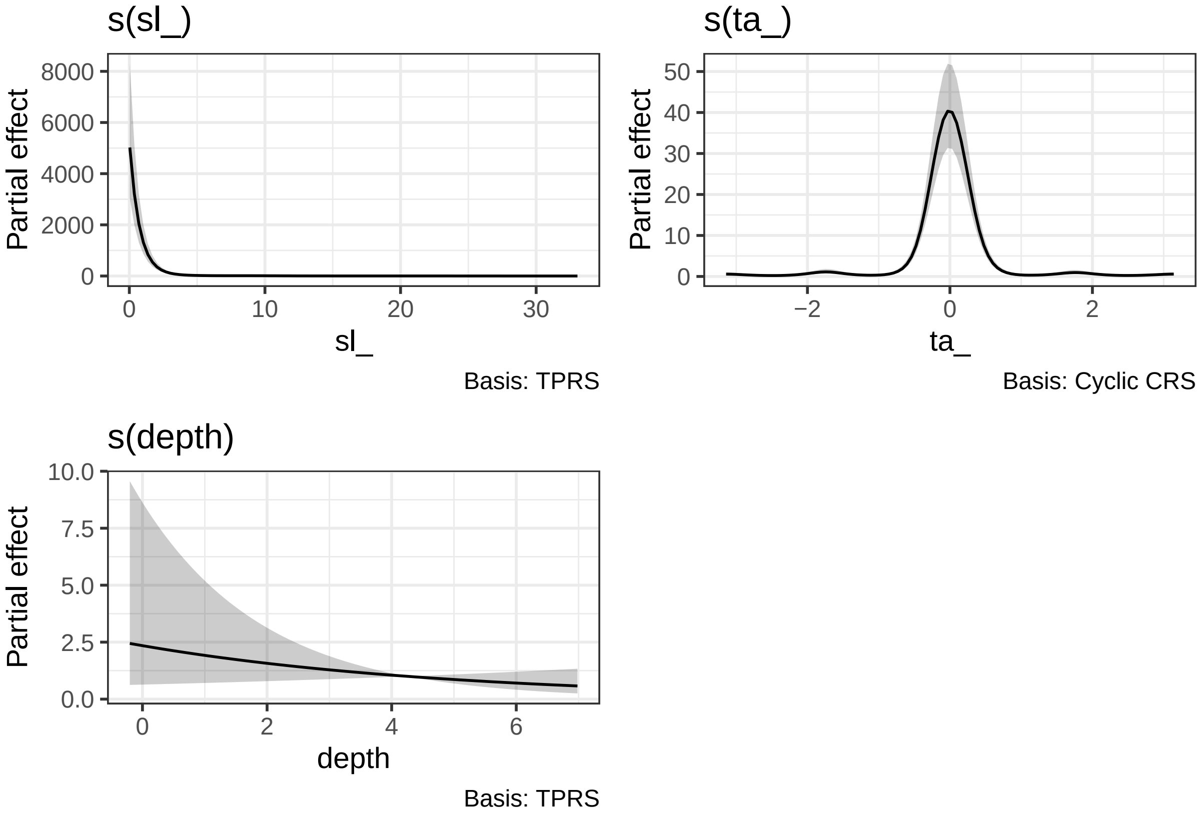
```
Link function: identity

Formula:
cbind(dummy_times, step_id_) ~ s(sl_) + s(ta_, bs = "cc") + s(depth)

Approximate significance of smooth terms:
            edf Ref.df  Chi.sq p-value
s(sl_)    7.409  8.174 1640.97  <2e-16 ***
s(ta_)    7.689  8.000  839.81  <2e-16 ***
s(depth) 1.001  1.002    1.65   0.199
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Deviance explained = -148%
-REML =   1223  Scale est. = 1         n = 103828
```

The smoothness parameter for the depth covariate is large (≈ 942) and the effective degrees of freedom are close to 1, suggesting that a linear relationship might be sufficient. We can visualise the estimated relationships, and plot them on the exponential scale for interpretability. Here, any two points along the smooth can be compared to calculate the RSS for any of the covariates. Here, we predict values of the smooth using `predict` and calculate the RSS for depth; since the relationship is very close to linear, these results are similar to those of the first model.

```r
# smoothness parameters
fit_smooth$sp
```

```
      s(sl_)       s(ta_)     s(depth)
6.357795e-03 5.789823e-01 9.419429e+02
```

```r
# estimated non-linear relationships
gratia::draw(fit_smooth, rug = FALSE, fun = exp)
```



```
# calculate relative selection strength
pred <- predict(fit_smooth,
                newdata = data.frame(sl_ = c(0, 0),
                                     ta_ = c(0, 0),
                                     depth = c(0, 1)))
rss <- exp(pred[2]) / exp(pred[1])
rss
```

```
        2
0.8183145
```

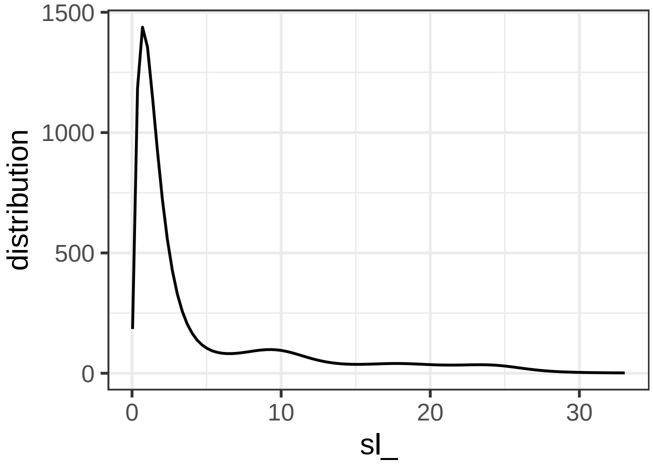
The relationship between turning angle and selection is directly proportional to the (non-parametric) turning angle distribution, and we see that the flexibility is used to capture the heavy tails. The step length distribution is not directly proportional to the relationship between selection and step length; instead, we need to look at the estimated smooth relationship *multiplied by step length*. This is to account for the disproportionate availability of long steps. The resulting non-parametric step length distribution seems to better capture the long tail present in the data.

```
# derive step length distribution
sm <- smooth_estimates(fit_smooth, select = "s(sl_)")
```

```
head(sm, 4)
```

```
# A tibble: 4 x 6
  .smooth .type .by   .estimate   .se    sl_
  <chr>   <chr> <chr>     <dbl> <dbl>  <dbl>
1 s(sl_)  TPRS  <NA>       8.52 0.256 0.0366
2 s(sl_)  TPRS  <NA>       8.07 0.240 0.370
3 s(sl_)  TPRS  <NA>       7.62 0.226 0.703
4 s(sl_)  TPRS  <NA>       7.18 0.214 1.04
```

```
ggplot(sm, aes(x = sl_, y = exp(.estimate) * sl_)) +
  geom_line() + ylab("distribution")
```



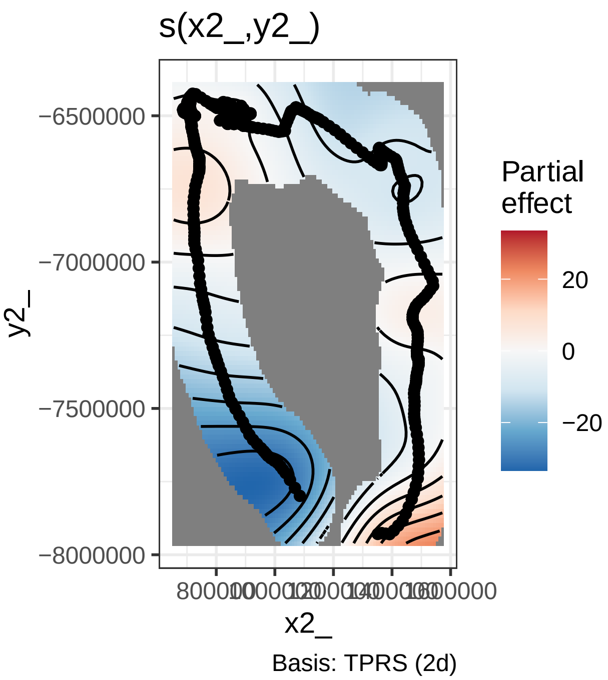
Model 3: spatial smoothing

Spatial smoothing (i.e., spatial random effects) can be useful to capture spatial variability that is unexplained by other covariates in the model. A spatial smooth can be included as an interaction between the x and y coordinates (at the *end* of the step). As long as the coordinates are projected (i.e., x and y are isotropic), we can use two-dimensional thin plate regression splines (the default in mgcv).

```
fit_ss <- gam(cbind(dummy_times, step_id_) ~
                s(sl_) +
                s(ta_, bs = "cc") +
                depth +
                s(x2_, y2_),
              data = data,
              knots = list(ta_ = c(-pi, pi)),
              family = cox.ph,
              weights = case_)
```

Plotting the spatial smooth shows spatial patterns of selection, once movement and bathymetry have been accounted for. The spatial smooth can also be interpreted in terms of relative selection strength, where we compare between two spatial locations (holding all else equal). Note the resulting smooth cannot be interpreted as the animal's spatial distribution.

```
# look at spatial smooth with observed points
gratia::draw(fit_ss, select = 3, rug = FALSE) +
  geom_point(aes(x1_, y1_), data = subset(data, case_ == 1))
```



```
# calculate relative selection strength
pred <- predict(fit_ss,
                newdata = data.frame(sl_ = c(0, 0),
                                     ta_ = c(0, 0),
                                     depth = c(0, 0),
                                     x2_ = c(8e5, 8e5),
                                     y2_ = c(-6.8e6, -6.7e6)))
rss <- exp(pred[2]) / exp(pred[1])
rss
```

```
       2
1.142027
```

Model 4: varying-coeffcient model

Interactions in SSFs are useful ways to allow for spatially or temporally varying patterns of movement and/or habitat selection. In mgcv, there are several options for including interactions (e.g., two-dimensional smoothers, tensor products). Here, we briefly describe a varying-coefficient model, where the linear effect of a covariate is allowed to smoothly vary with another. Specifically, we allow selection for depth to vary with time since departure to account for variations in selection through the petrel's foraging trip. We derive a simple time covariate to measure hours since the start of the time series.

```
data$time <- data$step_id_ / 2
fit_vc <- gam(cbind(dummy_times, step_id_) ~
                sl_ +
                cos(ta_) +
                s(time, by = depth, k = 12),
              data = data,
              family = cox.ph,
              weights = case_)
```

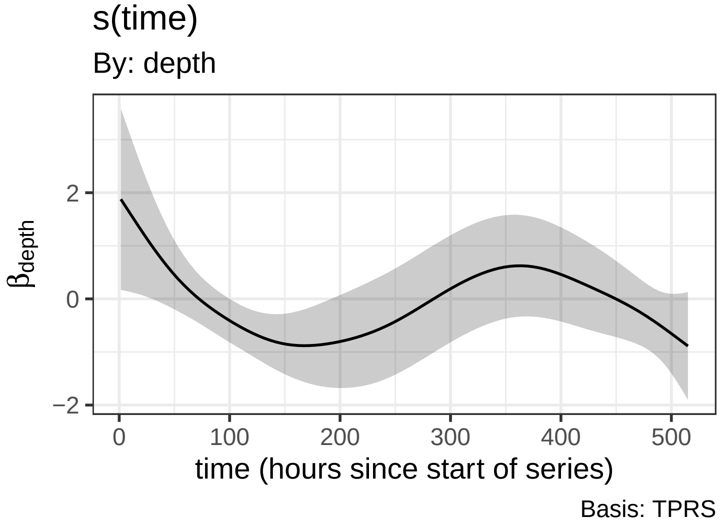
The results suggest that the petrel's selection for depth varied through time, likely corresponding to the phases of its foraging trip. The linear coefficient for depth ranged from roughly 2 (max) at hour 0 to -1 at hour 175 (minimum).

```
summary(fit_vc)
```

```

Family: Cox PH
Link function: identity

Formula:
cbind(dummy_times, step_id_) ~ sl_ + cos(ta_) + s(time, by = depth,
    k = 12)

Parametric coefficients:
          Estimate Std. Error z value Pr(>|z|)
sl_      -0.312819   0.007959  -39.30   <2e-16 ***
cos(ta_)  3.049987   0.134529   22.67   <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(time):depth 4.93  5.849  15.68  0.0143 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Deviance explained =  NaN%
-REML = 1697.8  Scale est. = 1         n = 103828
```

```
gratia::draw(fit_vc, rug = FALSE) +
  labs(x = "time (hours since start of series)", y = expression(beta[depth]))
```



Other notes

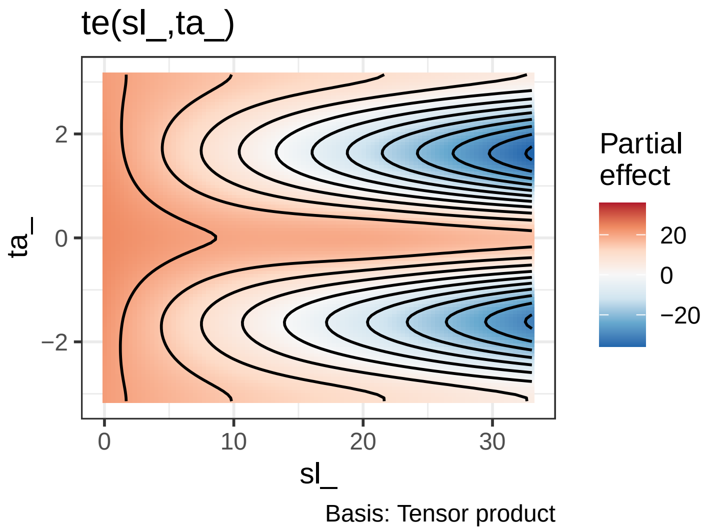
Model checking: As mentioned, model checking tools are not available for SSFs in mgcv. Although we are fitting the model as a Cox PH model, this is only a "trick" and an SSF is *not* a Cox PH model. However, the residuals in mgcv are based on Cox PH assumptions and therefore do not apply to SSFs. Model checking is typically a challenge of SSF analyses, and this approach does not provide a solution.

Other model formulations: There are many many options for model formulations within mgcv. This tutorial only covers a select few special cases we believe to be of interest.

- A coded example of random slopes and hierarchical smooths can be found on my GitHub (github.com/NJKlappstein/smoothSSF).

- Tensor products can be used to model dependence in movement or habitat variables. Full details can be found in Arce Guillen et al. (2024; https://doi.org/10.1101/2024.06.27.600970). To get you started, the code to do so for the petrel data would be:

```
fit_tensor <- gam(cbind(dummy_times, step_id_) ~
                    te(sl_, ta_, bs = c("tp", "cc") ) +
                    depth,
                  data = data,
                  family = cox.ph,
                  knots = list(ta_ = c(-pi, pi)),
                  weights = case_)

gratia::draw(fit_tensor, rug = FALSE)
```


te(sl_,ta_)
ta_
sl_
Partial effect
Basis: Tensor product