Using MixedSSA to Update Movement Distribution Parameters from ISSA models

08-15-2023

Document Preamble

```
library(knitr)
library(ggplot2)
library(mixedSSA)
library(here)
library(dplyr)
library(tidyr)
library(cowplot)
library(stringr)
library(circular)
library(glmmTMB)

# Set knitr options
opts_chunk$set(fig.width=8,
               fig.height=6,
               warning=FALSE)
```

Intro

The purpose of this vignette is to demonstrate how to use the package `mixedSSA` to update movement distribution parameters using ISSA models fit using `glmmTMB`. Currently, `mixedSSA` supports updating gamma, exponential, half-normal, log-normal, and gamma distributions and the von Mises distribution, for step lengths and turn angles, respectively. This vignette will focus on two ISSA models where step lengths are assumed to come from a gamma distribution and turn angles are assumed to come from a von Mises. The models are fit to the accompanying paper's red snapper (Lutjanus campechanus) telemetry data. The data used throughout this vignette were re-sampled using `amt:track_resample` with a `rate` of 6 minutes and a tolerance of 1 minute.

Raw Data Visualization

Let's start by loading our data and plotting the step lengths and turn angles.

```
load_data <- function() {
  return(readRDS(here("vignette/Snapper_final_data_6min.rds")))
}
```

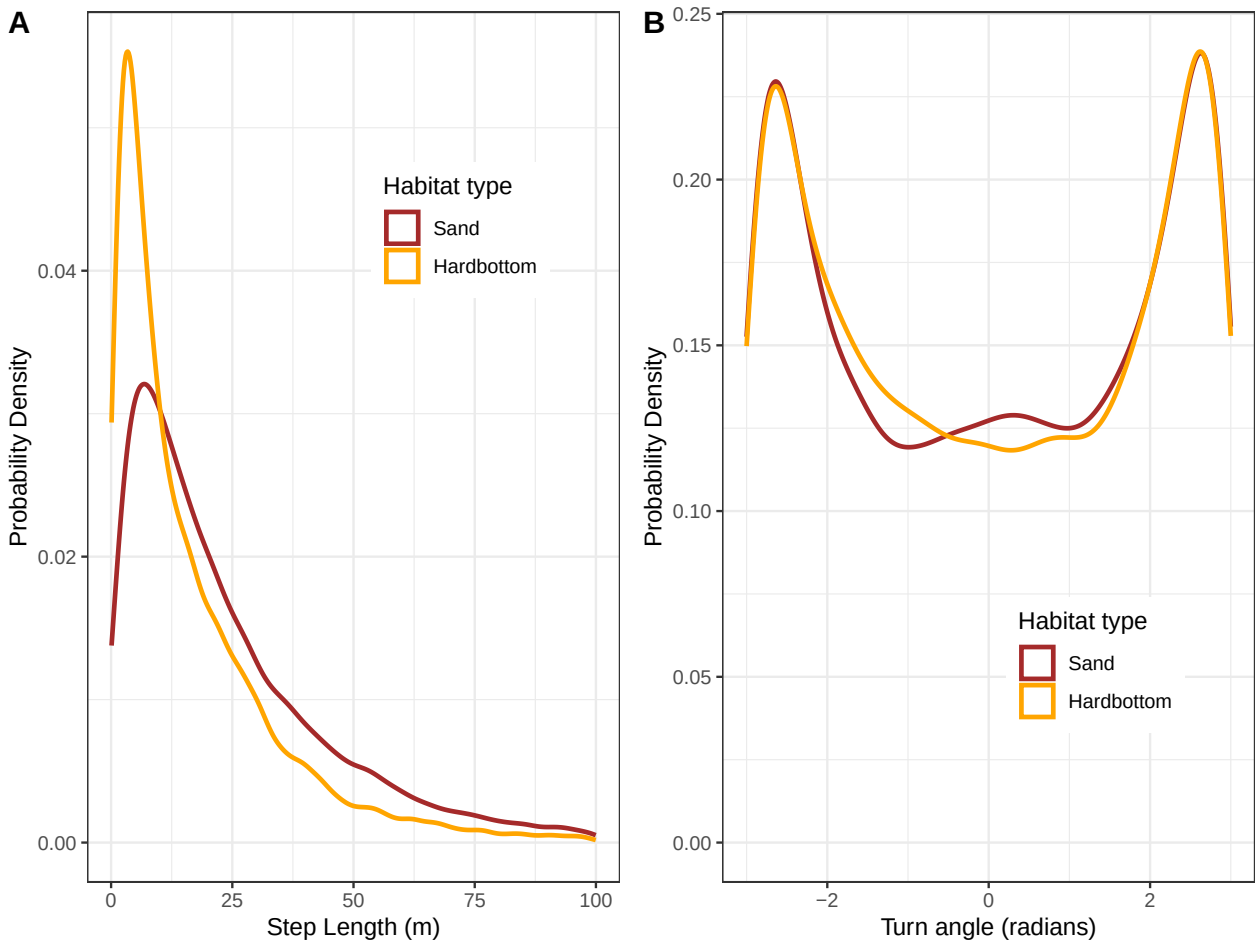
```
data <- load_data()
head(data)
```

```
## # A tibble: 6 x 16
##      id burst_    sl_    ta_ dt_      step_id_ case_ dist_edge reefStart reefEnd
## * <dbl>  <dbl>  <dbl>  <dbl> <drtn>      <int> <lgl>     <dbl>     <dbl>   <dbl>
## 1     2      5  9.93  -1.82  5.28333~        1 FALSE      4.47         1       1
## 2     2      5  0.345  1.04  5.28333~        1 FALSE      5            1       1
## 3     2      5 16.3   -0.465 5.28333~        1 FALSE      0            1       1
## 4     2      5 10.7   -1.31  5.28333~        1 FALSE      0            1       1
## 5     2      5  7.44   1.63  5.28333~        1 FALSE     11.2          1       1
## 6     2      5 44.1    0.719 5.28333~        1 FALSE      4.12         1       1
## # i 6 more variables: log_sl_ <dbl>, cos_ta_ <dbl>, step_id <chr>,
## #   reefS4 <dbl>, reefE4 <dbl>, newdist <dbl>
```

Let's plot the raw data to get a sense of the distribution of step lengths and turn angles

```
plot_movement_data <- function(data) {
  plot1 <- data %>%
    filter(case_ == TRUE) %>%
    drop_na(reefStart) %>%
    dplyr::select(sl_, reefStart) %>%
    ggplot(aes(sl_, col = factor(reefStart))) +
    geom_density(linewidth = 1) +
    scale_color_manual(
      name = "Habitat type",
      values = c("brown", "orange"),
      labels = c("Sand", "Hardbottom")
    ) +
    labs(x = "Step Length (m)", y = "Probability Density") +
    theme_bw() +
    theme(legend.position = c(0.7, 0.75))

  plot2 <- data %>%
    filter(case_ == TRUE) %>%
    drop_na(reefStart) %>%
    dplyr::select(ta_, reefStart) %>%
    ggplot(aes(ta_, col = factor(reefStart))) +
    geom_density(linewidth = 1) +
    scale_color_manual(
      name = "Habitat type",
      labels = c("Sand", "Hardbottom"),
      values = c("brown", "orange")
    ) +
    labs(x = "Turn angle (radians)", y = "Probability Density") +
    theme_bw() +
    theme(legend.position = c(0.7, 0.25))

  all_plots <- plot_grid(plot1, plot2,
    labels = c("A", "B"),
    ncol = 2, nrow = 1
  )
  return(all_plots)
```

```
}

plot_movement_data(data)
```



Fitting the Models

Here's a function that will let us fit the two types of models we will fit in this vignette

```
fit_model <- function(data, model_name, interaction_var_name, save_model = TRUE,
                      from_cache = FALSE) {
  # If you've already fit the model once, just pull it from the cache
  file_path <- here(str_interp("vignette/models/${model_name}.rds"))
  if (from_cache) {
    model <- readRDS(file_path)
    return(model)
  }


  formula <- as.formula(str_interp("case_ ~ sl_ + log_sl_ + cos(ta_) + reefE4 +
  newdist + ${interaction_var_name}:sl_ + ${interaction_var_name}:log_sl_ +
  ${interaction_var_name}:cos(ta_) +
```

```
  (1 | step_id) + (0 + sl_ + (log_sl_) + cos(ta_) | id) + (0 + newdist | id) +
                                  (0 + reefE4 | id)"))

  model_config <- glmmTMB(
    formula,
    REML = TRUE,
    family = poisson(), data = data,
    doFit = FALSE
  )

  model_config$parameters$theta[1] <- log(1e3)
  nthetas <- length(model_config$parameters$theta)
  model_config$mapArg <- list(theta = factor(c(1:nthetas)))
  model <- glmmTMB:::fitTMB(model_config)

  if (save_model) {
    saveRDS(model, file = file_path)
  }

  return(model)
}
```

Model 1, Interaction with Categorical Variable ReefS4

Fitting the Model

We will first fit an ISSA model with the categorical interaction variable, `ReefS4`. `ReefS4` is technically a dummy variable (i.e. 0 or 1), with 0 representing Sand and 1 representing Hard bottom (represented as `Hardbottom` in this vignette) habitat. We will have to do a little finagling later to handle the fact that we are representing a categorical variable numerically.

```
interaction_var_name <- "reefS4"
model1 <- fit_model(
  data = data,
  model_name = "model1",
  interaction_var_name = interaction_var_name,
  from_cache = TRUE # Change this to FALSE, if you haven't fit the model yet
)

summary(model1)
```

```
##  Family: poisson  ( log )
## Formula:
## case_ ~ sl_ + log_sl_ + cos(ta_) + reefE4 + newdist + reefS4:sl_ +
##     reefS4:log_sl_ + reefS4:cos(ta_) + (1 | step_id) + (0 + sl_ +
##     (log_sl_) + cos(ta_) | id) + (0 + newdist | id) + (0 + reefE4 |      id)
## Data: data
##
##      AIC      BIC   logLik deviance df.resid
## 182377.9 182576.8 -91170.9 182341.9   465479
##
## Random effects:
```

```
##
## Conditional model:
##  Groups  Name        Variance  Std.Dev.  Corr
##  step_id (Intercept) 1.266e-08 0.0001125
##  id      sl_         3.949e-04 0.0198722
##          log_sl_     3.062e-02 0.1749752 -0.95
##          cos(ta_)    3.309e-02 0.1818966  0.25 -0.19
##  id.1    newdist     4.556e-04 0.0213459
##  id.2    reefE4      4.330e-02 0.2080790
## Number of obs: 465488, groups:  step_id, 45915; id, 35
##
## Conditional model:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -3.116218   0.019086 -163.27  < 2e-16 ***
## sl_             -0.004719   0.003620   -1.30  0.19239
## log_sl_          0.107612   0.033393    3.22  0.00127 **
## cos(ta_)        -0.854549   0.036955  -23.12  < 2e-16 ***
## reefE4           0.246946   0.043437    5.69 1.31e-08 ***
## newdist         -0.013068   0.004001   -3.27  0.00109 **
## sl_:reefS4      -0.014503   0.001205  -12.04  < 2e-16 ***
## log_sl_:reefS4   0.052904   0.012030    4.40 1.09e-05 ***
## cos(ta_):reefS4  0.036143   0.020255    1.78  0.07435 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Updating the Step Length Distribution Parameters

We will use `mixedSSA::update_dist` to update the gamma distribution parameters for each habitat type, `Sand` or `Hardbottom`.

```r
update_sl_dist <- function(model, interaction_var_name,
                           random_effects_var_name = NULL, quantiles = NULL) {
  # The package `mixedSSA` determines how to update the parameters based on the
  # type of the interaction variable, so we have to make sure we set the
  # variable `reefS4` to a factor so that the package knows we are actually
  # dealing with a categorical variable
  if (interaction_var_name == "reefS4") {
    model$frame <- model$frame %>%
      mutate(
        reefS4 = as.factor(reefS4)
      )
  }

  # Use the package to get the updated gamma distribution parameter values
  updated_parameters <- mixedSSA::update_dist(
    model = model,
    dist_name = "gamma",
    beta_sl = "sl_", # the name of the step lengths coefficient in our model
    beta_log_sl = "log_sl_", # the name of the log(sl) coefficient in our model
    interaction_var_name = interaction_var_name, # reefS4 or newdist
    random_effects_var_name = random_effects_var_name, # for individual update
    quantiles = quantiles # for continuous interaction variables
  )
```

```
  return(updated_parameters)
}
```

`mixedSSA::update_dist` returns an instance of a class with multiple "slots" that can be accessed using "@". The updated distribution parameters can be accessed using @updated_parameters. But the object contains other data that were used to calculate the updated parameters, like `distribution_name`, `movement_data`, and `grouping`.

```
sl_params_obj <- update_sl_dist(
  model = model1,
  interaction_var_name = interaction_var_name
)
```

We can see we updated the parameters of the gamma distribution

```
sl_params_obj@distribution_name
```

```
## [1] "gamma"
```

And also take a look at the movement data (the raw step lengths or turn angles) that were used to fit the tentative distribution.

```
head(sl_params_obj@movement_data)
```

```
## [1] 22.722504  1.838960 25.189978  9.158110  3.838282 32.403371
```

Lastly, we can see our updated parameter values for each category (for categorical interaction variables) or quantile (for numerical interaction variables). In the case of a gamma distribution, the `updated_parameters` tibble also includes the values of the coefficients that were used to calculate the new parameter values.

```
(sl_params <- sl_params_obj@updated_parameters)
```

```
##    category   beta_sl beta_log_sl    shape    scale
## 1 tentative        NA          NA 1.090652 18.22081
## 2         0 -0.004719    0.107612 1.198264 16.77816
## 3         1 -0.019222    0.160516 1.251168 13.49457
```

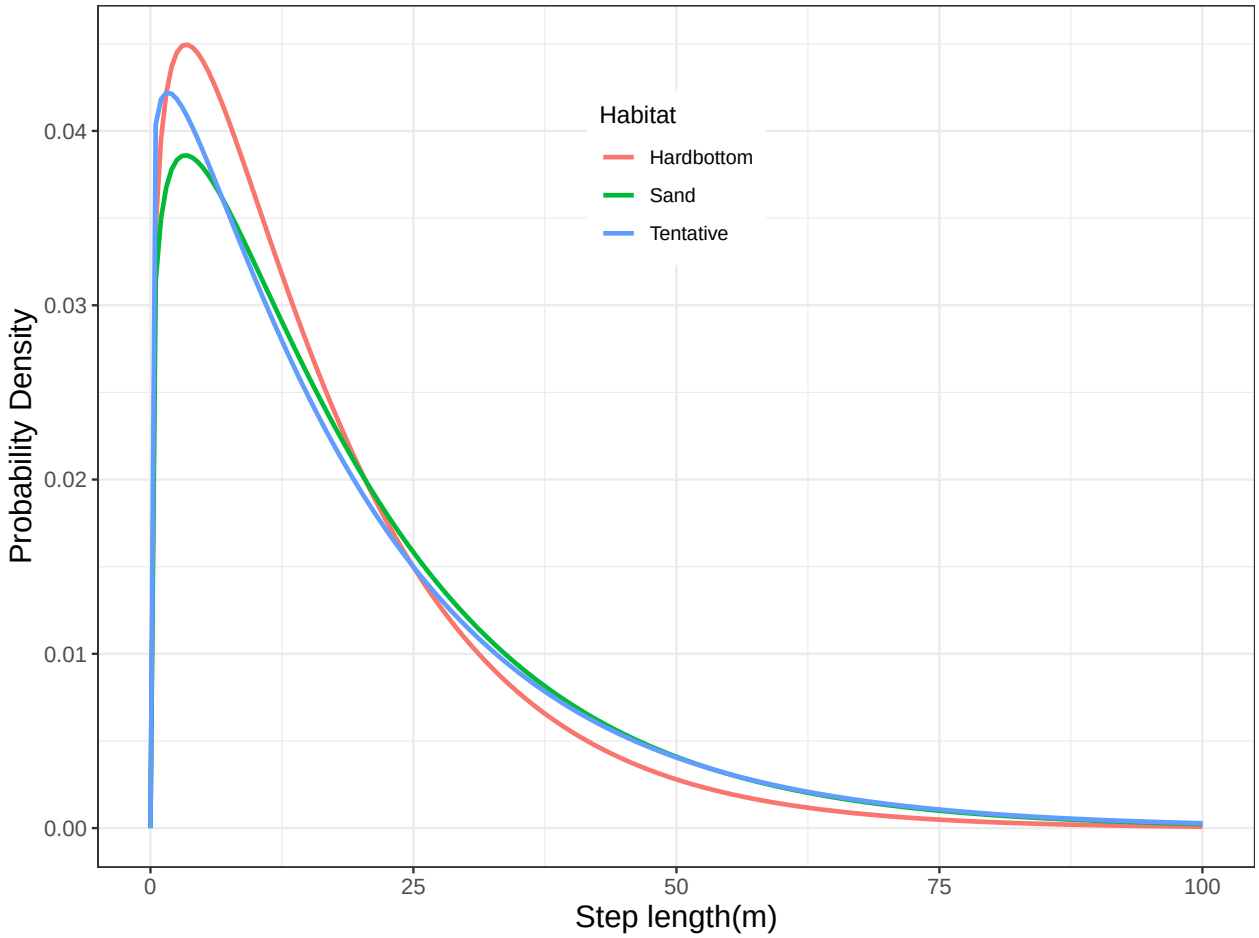
Plotting Updated Distributions

Let's actually take a look at the updated parameters and how they vary for our different habitat types

```
# a function for getting all the step length plot lines from the updated_parameters tibble
get_sl_plot_data <- function(updated_parameters) {
  plots_data <- list()

  for (i in 1:nrow(updated_parameters)) {
    plots_data[[i]] <- data.frame(x = rep(NA, 200))
    plots_data[[i]]$x <- seq(
      from = 0,
```

```
      to = 100,
      length.out = 200
    )

    # These will not be set if the column does not exist in the
    # updated_parameters tibble, which is fine
    plots_data[[i]]$category <- updated_parameters[i, "category"]
    plots_data[[i]]$quantile <- updated_parameters[i, "quantile"]
    plots_data[[i]]$id <- updated_parameters[i, "id"]

    plots_data[[i]]$y <- dgamma(
      x = plots_data[[i]]$x,
      shape = updated_parameters[i, "shape"],
      scale = abs(updated_parameters[i, "scale"])
    )
  }

  plots_data_all <- do.call(rbind, plots_data)
  return(plots_data_all)
}


# A generic plot function for plotting data generated from get_sl_plot_data or
# get_ta_plot_data
plot_updated_dist <- function(plots_data, color_col, vonmises = FALSE,
                              with_id = FALSE) {
  if (with_id) {
    lines <- geom_line(aes_string(
      x = "x", y = "y",
      group = "id", col = color_col,
      alpha = color_col
    ), linewidth = 1)
  } else {
    lines <- geom_line(aes_string(
      x = "x", y = "y",
      col = color_col
    ), linewidth = 1)
  }

  plot <- ggplot(data = plots_data) +
    lines +
    theme_bw() +
    theme(
      axis.title = element_text(size = 14),
      axis.text = element_text(size = 10),
      legend.position = c(0.5, 0.8)
    )

  if (vonmises) {
    plot <- plot +
      scale_x_continuous(
        breaks = c(-pi, -pi / 2, 0, pi / 2, pi),
        labels = c(expression(-pi, -pi / 2, 0, pi / 2, pi))
```

```
      ) +
      labs(x = "Relative Turn-angle (radians)", y = "Probability Density")
  } else {
    plot <- plot + labs(x = "Step length(m)", y = "Probability Density")
  }

  return(plot)
}


sl_plots_data <- get_sl_plot_data(sl_params)
sl_plots_data <- sl_plots_data %>%
  mutate(
    Habitat = case_when( # rename categories to the real meaning
      category == "tentative" ~ "Tentative",
      category == "0" ~ "Sand",
      category == "1" ~ "Hardbottom"
    )
  )

plot_updated_dist(sl_plots_data, with_id = FALSE, color_col = "Habitat")
```



Updating the Turn Angle Distribution Parameters

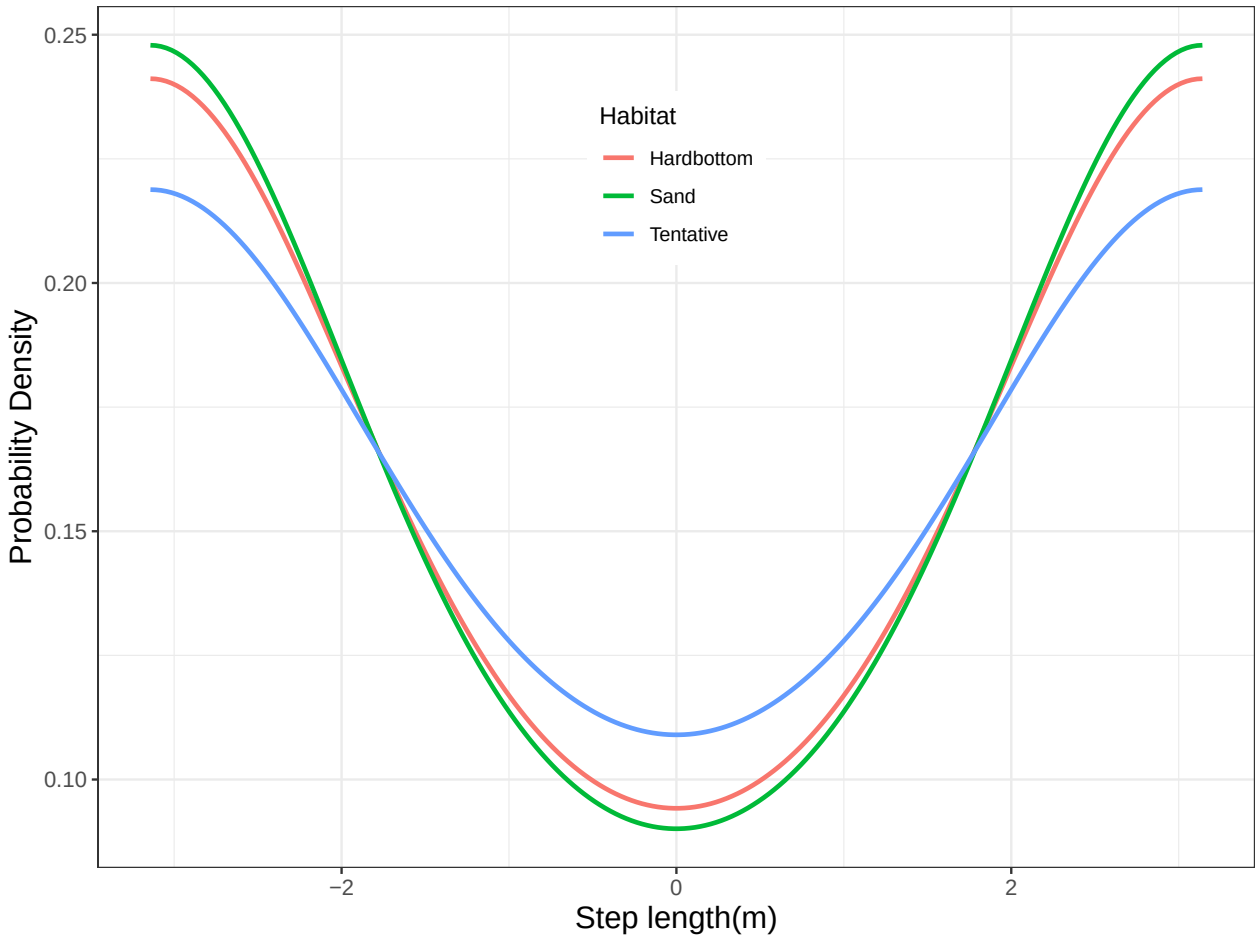
Now we can do the same for our turn angle distribution, the von Mises. When updating a von Mises distribution for turn angles, we must pass the tentative distribution. This is because the package `mixedSSA` is not able to back transform `cos(turn_angle)` and know the original sign (positive or negative) of the turn angle.

```
update_ta_dist <- function(model, data, interaction_var_name,
                           random_effects_var_name = NULL, quantiles = NULL) {
  # The package `mixedSSA` determines how to update the parameters
  # based on the type of the interaction variable, so we have to make sure
  # we set the variable `reefS4` to a factor so that the package knows we are
  # dealing with a categorical variable
  if (interaction_var_name == "reefS4") {
    model$frame <- model$frame %>%
      mutate(
        reefS4 = as.factor(reefS4)
      )
  }

  # Use the package to get the updated vonmises distribution parameter values
  updated_parameters <- mixedSSA:::update_dist(
    model = model,
    dist_name = "vonmises",
    beta_cos_ta = "cos(ta_)", # name of the cos(ta) coefficient in our model
    interaction_var_name = interaction_var_name,
    random_effects_var_name = random_effects_var_name,
    quantiles = quantiles,
    tentative_dist = amt::fit_distr(data$ta_[data$case_ == "TRUE"],
      "vonmises",
      na.rm = T
    )
  )

  return(updated_parameters)
}

ta_params <- update_ta_dist(
  model = model1,
  data = data,
  interaction_var_name = interaction_var_name
)@updated_parameters
```

Plotting Updated Distributions

```
# This function, like get_sl_plot_data, gets the plot line data using the
# von Mises density function
get_ta_plot_data <- function(updated_parameters, random_effects_var = FALSE) {
  plots_data <- list()

  for (i in 1:nrow(updated_parameters)) {
    plots_data[[i]] <- data.frame(x = rep(NA, 200))
```

```
    plots_data[[i]]$x <- seq(
      from = -pi,
      to = pi,
      length.out = 200
    )

    plots_data[[i]]$category <- updated_parameters[i, "category"]
    plots_data[[i]]$quantile <- updated_parameters[i, "quantile"]
    plots_data[[i]]$id <- updated_parameters[i, "id"]

    plots_data[[i]]$y <- circular::dvonmises(
      x = plots_data[[i]]$x,
      kappa = abs(updated_parameters[i, "kappa"]),
      mu = circular(pi)
    )
  }

  plots_data_all <- do.call(rbind, plots_data)

  return(plots_data_all)
}


ta_plots_data <- get_ta_plot_data(ta_params)
ta_plots_data <- ta_plots_data %>%
  mutate(
    Habitat = case_when(
      category == "tentative" ~ "Tentative",
      category == "0" ~ "Sand",
      category == "1" ~ "Hardbottom"
    )
  )

plot_updated_dist(ta_plots_data, with_id = FALSE, color_col = "Habitat")
```



Random Effects

Okay, we've demonstrated how you can use `mixedSSA` to calculate distribution parameters updated using fixed effects. But! You can also use `mixedSSA` to update movement distributions per individual animal (or any random effect you have fit in your model). Let's take a look at step lengths again. Here, we pass to our function the name of our random effect, in this case "id" of the snapper. This time, the `updated_parameters` tibble contains parameter values for every combo of individual and habitat type.

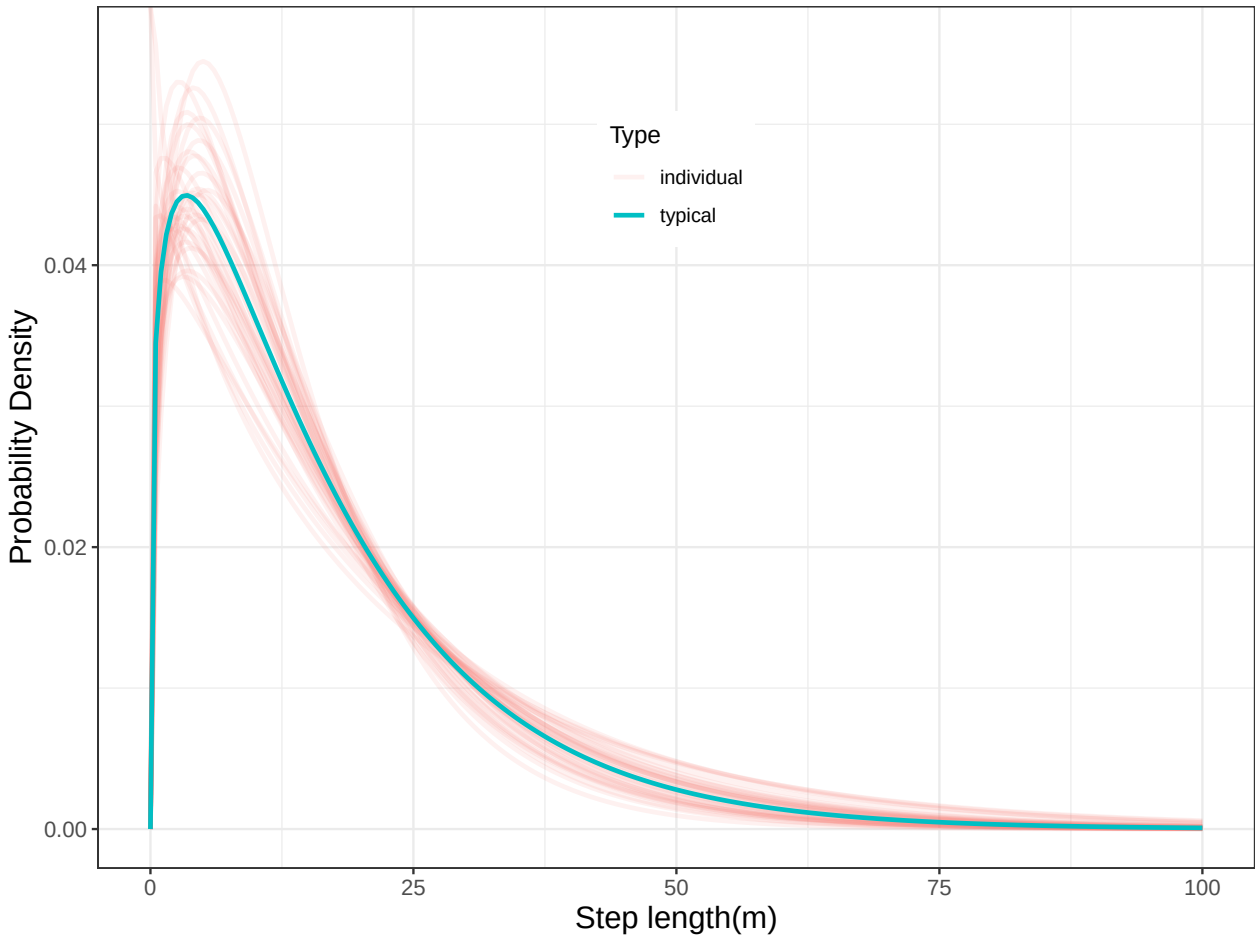
```
sl_params_ind <- update_sl_dist(
  model = model1,
  interaction_var_name = interaction_var_name,
  random_effects_var_name = "id"
)@updated_parameters

head(sl_params_ind)
```

```
##    category id   beta_sl beta_log_sl    shape    scale
## 1 tentative NA        NA          NA 1.090652 18.22081
## 2         0  2 -0.010396    0.172713 1.263365 15.31904
## 3         0  3  0.001717    0.031281 1.121933 18.80914
## 4         0  4 -0.033658    0.340183 1.430835 11.29431
## 5         0  5 -0.019132    0.157348 1.248000 13.51098
## 6         0  6  0.029611   -0.253008 0.837645 39.57016
```

Plotting Updated Individual Distributions

Let's take a look at how these individual parameters compare to the "typical" individual, where all random effects are set to 0.

```
# get the typical values from the non-individual updated parameters.
sl_plots_data <- get_sl_plot_data(sl_params) %>%
  filter(category != "tentative") %>%
  mutate(id = "typical")

# get plot data for the individual distributions
sl_plots_data_ind <- get_sl_plot_data(sl_params_ind)
sl_plots_data_ind <- rbind(sl_plots_data_ind, sl_plots_data) %>%
  filter(category == "1") %>% # only look at one habitat
  mutate(
    Type = case_when(
      id == "typical" ~ "typical",
      id != "typical" ~ "individual"
    )
  )

plot_updated_dist(sl_plots_data_ind, color_col = "Type", with_id = TRUE)
```



Model 2, Interaction with Numerical Variable `newdist` (i.e. distance to hard bottom, or edge habitat).

We've demonstrated how we can use `mixedSSA` with models fit with categorical interaction terms. Let's take a look at how it works with continuous variables.

Fitting the ISSA model

```
interaction_var_name <- "newdist"
model2 <- fit_model(
  data = data,
  model_name = "model2",
  interaction_var_name = interaction_var_name,
  from_cache = TRUE
)

summary(model2)
```

```
##  Family: poisson  ( log )
## Formula:
## case_ ~ sl_ + log_sl_ + cos(ta_) + reefE4 + newdist + newdist:sl_ +
##     newdist:log_sl_ + newdist:cos(ta_) + (1 | step_id) + (0 +
##     sl_ + (log_sl_) + cos(ta_) | id) + (0 + newdist | id) + (0 +
##     reefE4 | id)
## Data: data
##
##      AIC      BIC   logLik deviance df.resid
## 182695.3 182894.3 -91329.7 182659.3   465577
##
## Random effects:
##
## Conditional model:
##  Groups  Name        Variance  Std.Dev.  Corr
##  step_id (Intercept) 1.266e-08 0.0001125
##  id      sl_         4.269e-04 0.0206625
##          log_sl_     3.274e-02 0.1809387 -0.94
##          cos(ta_)    3.001e-02 0.1732364  0.32 -0.26
##  id.1    newdist     5.137e-04 0.0226652
##  id.2    reefE4      4.110e-02 0.2027392
## Number of obs: 465586, groups:  step_id, 45931; id, 35
##
## Conditional model:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -3.047e+00  2.132e-02 -142.93  < 2e-16 ***
## sl_              -7.333e-03  3.760e-03   -1.95   0.0511 .
## log_sl_           7.201e-02  3.466e-02    2.08   0.0378 *
## cos(ta_)         -8.926e-01  3.548e-02  -25.16  < 2e-16 ***
## reefE4            2.346e-01  4.241e-02    5.53 3.19e-08 ***
## newdist          -2.295e-02  4.543e-03   -5.05 4.37e-07 ***
## sl_:newdist      -2.521e-04  4.628e-05   -5.45 5.08e-08 ***
## log_sl_:newdist   6.403e-03  8.876e-04    7.21 5.41e-13 ***
## cos(ta_):newdist  5.594e-03  7.813e-04    7.16 8.11e-13 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Updating the Step Length Distribution Parameters

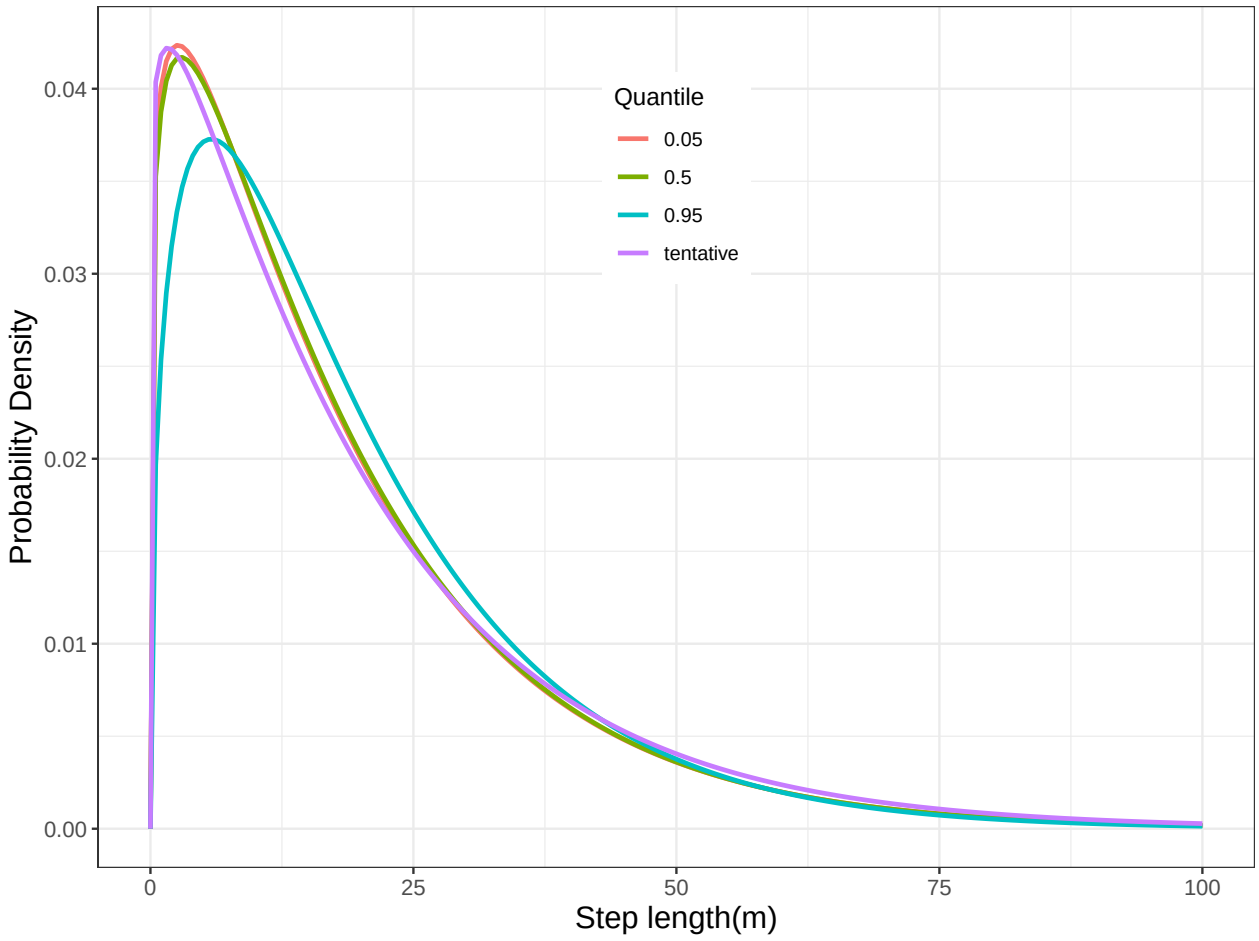
This time we pass `quantiles` to `mixedSSA::update_dist`. `mixedSSA` will use these quantiles of "newdist" to calculate scaled coefficient values to update the distribution parameters. `mixedSSA::update_dist` will return a tibble that contains the tentative parameter values and the parameter values for each quantile of `newdist`.

```
sl_params <- update_sl_dist(
  model = model2,
  interaction_var_name = interaction_var_name,
  quantiles = c(0.05, 0.5, 0.95)
)@updated_parameters

head(sl_params)
```

```
##    quantile   beta_sl beta_log_sl    shape    scale
## 1 tentative        NA          NA 1.090677 18.22123
## 2      0.05 -0.007333    0.072007 1.162684 16.07360
## 3       0.5 -0.008089    0.091217 1.181894 15.88052
## 4      0.95 -0.017374    0.327022 1.417699 13.83981
```

Plotting The Updated Distributions

```
sl_cont_plot_data <- get_sl_plot_data(sl_params)
sl_cont_plot_data <- sl_cont_plot_data %>%
  mutate(
    Quantile = as.factor(quantile)
  )

plot_updated_dist(sl_cont_plot_data, with_id = FALSE, color_col = "Quantile")
```

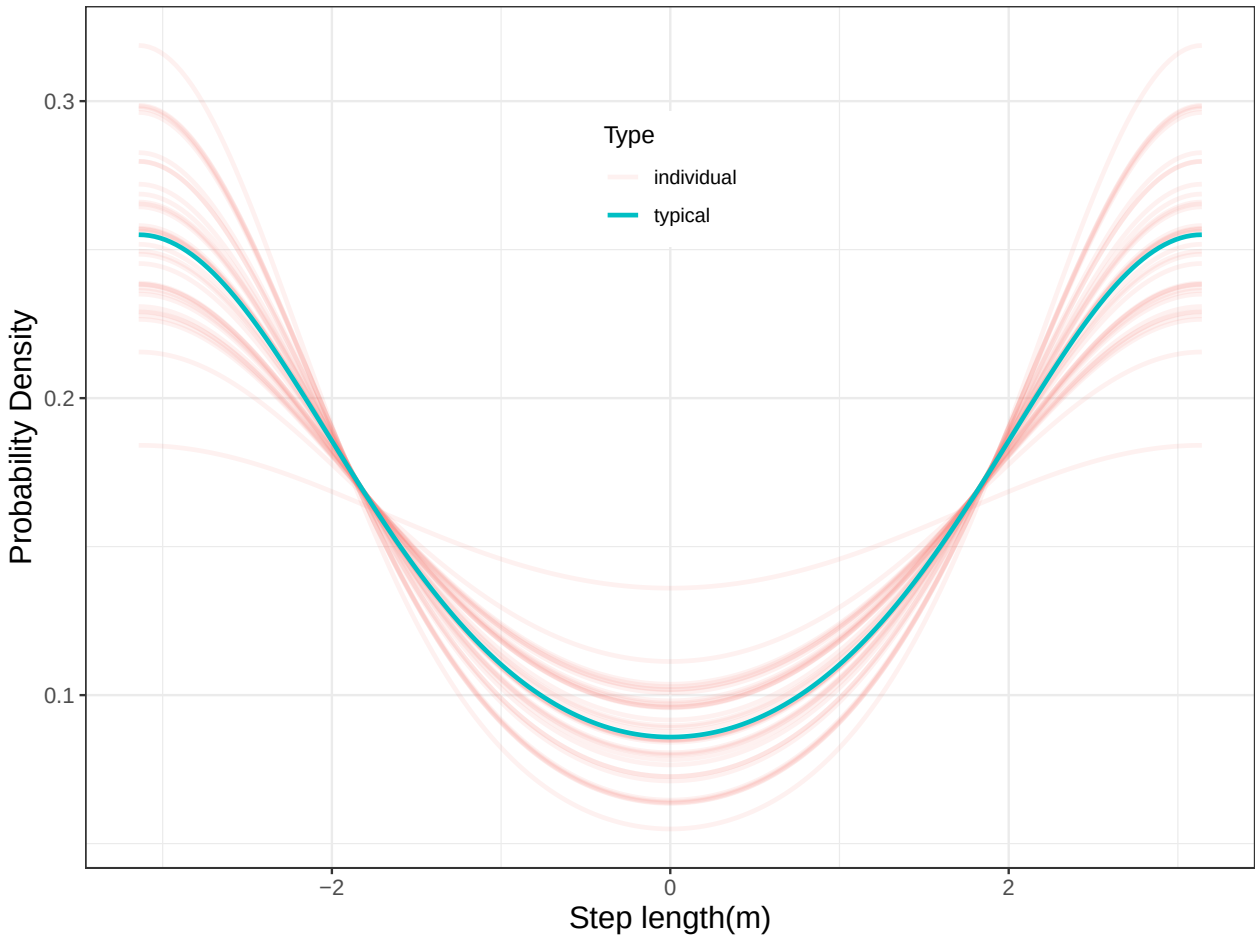


Random Effects

As with model 1, we can calculate the updated distribution parameters for each individual. This time, let's look at the turn angles, or von Mises distribution.

```
# First, get the "typical" individual (i.e. where random effects set to 0)
ta_params <- update_ta_dist(
  model = model2,
  data = data,
  interaction_var_name = interaction_var_name,
  quantiles = c(0.05, 0.5, 0.95)
)@updated_parameters

ta_params_ind <- update_ta_dist(
  model = model2,
  data = data,
  interaction_var_name = interaction_var_name,
  random_effects_var_name = "id",
  quantiles = c(0.05, 0.5, 0.95)
)@updated_parameters

# Get the "typical" plot data
ta_plots_data <- get_ta_plot_data(ta_params) %>%
```

```
  filter(quantile != "tentative") %>%
  mutate(id = "typical")

ta_plots_data_ind <- get_ta_plot_data(ta_params_ind, random_effects_var = TRUE)
ta_plots_data_ind <- rbind(ta_plots_data_ind, ta_plots_data) %>%
  filter(quantile == 0.05) %>% # Only plot the 5th percentile, to make things easier
  mutate(
    Type = case_when(
      id == "typical" ~ "typical",
      id != "typical" ~ "individual"
    ),
  )

plot_updated_dist(ta_plots_data_ind, color_col = "Type", with_id = TRUE)
```



Document footer

Session Information:

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] glmmTMB_1.1.7   circular_0.4-95 stringr_1.5.0   cowplot_1.1.1
##  [5] tidyr_1.3.0     dplyr_1.1.2     here_1.0.1      mixedSSA_1.0.0
##  [9] ggplot2_3.4.1   knitr_1.43
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-162               sf_1.0-13
##  [3] lubridate_1.9.2            assertive.models_0.0-2
##  [5] rprojroot_2.0.3            numDeriv_2016.8-1.1
##  [7] backports_1.4.1            assertive.datetimes_0.0-3
##  [9] tools_4.2.1                TMB_1.9.4
## [11] utf8_1.2.3                 R6_2.5.1
## [13] KernSmooth_2.23-20         DBI_1.1.3
## [15] colorspace_2.1-0           assertive.data_0.0-3
## [17] withr_2.5.0                assertive.reflection_0.0-5
## [19] tidyselect_1.2.0           emmeans_1.8.5
## [21] compiler_4.2.1             amt_0.2.1.0
## [23] cli_3.6.1                  assertive.properties_0.0-5
## [25] labeling_0.4.2             assertive.files_0.0-2
## [27] checkmate_2.2.0            scales_1.2.1
## [29] classInt_0.4-9             mvtnorm_1.1-3
## [31] proxy_0.4-27               digest_0.6.31
## [33] minqa_1.2.5                rmarkdown_2.20
## [35] assertive.numbers_0.0-2    pkgconfig_2.0.3
## [37] htmltools_0.5.5            lme4_1.1-33
## [39] fastmap_1.1.1              highr_0.10
## [41] rlang_1.1.1                rstudioapi_0.14
## [43] assertive_0.3-6            generics_0.1.3
## [45] farver_2.1.1               berryFunctions_1.22.0
## [47] magrittr_2.0.3             Matrix_1.5-3
## [49] Rcpp_1.0.10                munsell_0.5.0
## [51] fansi_1.0.4                abind_1.4-5
## [53] lifecycle_1.0.3            stringi_1.7.12
## [55] assertive.base_0.0-9       yaml_2.3.7
## [57] MASS_7.3-58.3              grid_4.2.1
## [59] lattice_0.20-45            assertive.code_0.0-4
## [61] splines_4.2.1              hash_2.2.6.2
## [63] pillar_1.9.0               boot_1.3-28.1
## [65] estimability_1.4.1         assertive.sets_0.0-3
## [67] codetools_0.2-19           glue_1.6.2
```

```
## [69] evaluate_0.21              vctrs_0.6.2
## [71] nloptr_2.0.3               Rdpack_2.4
## [73] gtable_0.3.3               purrr_1.0.1
## [75] assertive.strings_0.0-3    xfun_0.39
## [77] rbibutils_2.2.13           xtable_1.8-4
## [79] assertive.types_0.0-3      e1071_1.7-13
## [81] coda_0.19-4                assertive.data.uk_0.0-2
## [83] class_7.3-21               survival_3.5-5
## [85] tibble_3.2.1               units_0.8-2
## [87] assertive.matrices_0.0-2   timechange_0.2.0
## [89] fitdistrplus_1.1-11        assertive.data.us_0.0-2
```