Step selection model fitting using TwoStepCLogit

Scott Forrest

2024-02-29

Load packages

```
options(scipen=999)

library(tidyverse)
packages <- c("lubridate", "survival", "terra", "tictoc",
              "TwoStepCLogit", "ecospat", "beepr", "clogitL1",
              "ggpubr", "MASS", "patchwork", "glmmTMB", "scales",
              "formatR")
walk(packages, require, character.only = T)
```

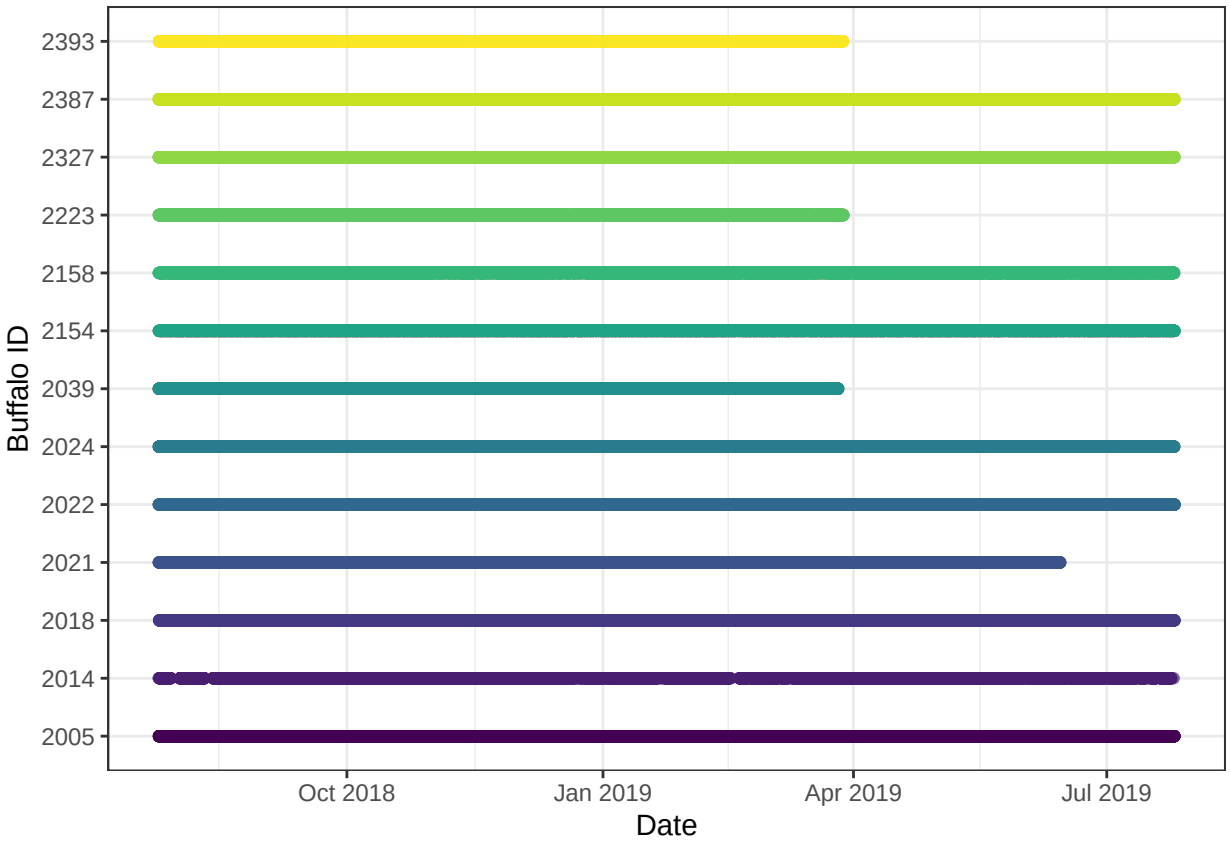
Importing buffalo data

```
buffalo_data_all <- read_csv(
  "outputs/buffalo_popn_GvM_covs_ST_KDEmem1000_allOPTIM_10rs_2024-02-05.csv")
```

```
## Rows: 1082697 Columns: 26
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (22): id, burst_, x1_, x2_, y1_, y2_, sl_, ta_, dt_, hour_t2, step_id_, y, ndvi_temporal, veg_w
## lgl   (1): case_
## dttm  (3): t1_, t2_, t2_rounded
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
buffalo_data_all <- buffalo_data_all %>%
  mutate(t1_ = lubridate::with_tz(buffalo_data_all$t1_, tzone = "Australia/Darwin"),
         t2_ = lubridate::with_tz(buffalo_data_all$t2_, tzone = "Australia/Darwin"))

buffalo_data_all <- buffalo_data_all %>%
  mutate(id_num = as.numeric(factor(id)),
         step_id = step_id_,
         x1 = x1_, x2 = x2_,
         y1 = y1_, y2 = y2_,
         t1 = t1_,
         t1_rounded = round_date(buffalo_data_all$t1_, "hour"),
         hour_t1 = hour(t1_rounded),
         t2 = t2_,
         t2_rounded = round_date(buffalo_data_all$t2_, "hour"),
         hour_t2 = hour(t2_rounded),
         hour = hour_t2,
         yday = yday(t1_),
         year = year(t1_),
```

```
        month = month(t1_),
        sl = sl_,
        log_sl = log(sl_),
        ta = ta_,
        cos_ta = cos(ta_),
        # scale canopy cover from 0 to 1
        canopy_01 = canopy_cover/100,
        # recover the natural scale of previous space use density
        kde_memory_density = exp(kde_memory_density_log),
        # here we create the harmonic terms for the hour of the day
        # for seasonal effects, change hour to day of the year (which is t),
        # and 24 to 365 (which is T)
        hour_s1 = sin(2*pi*hour/24),
        hour_s2 = sin(4*pi*hour/24),
        hour_s3 = sin(6*pi*hour/24),
        hour_c1 = cos(2*pi*hour/24),
        hour_c2 = cos(4*pi*hour/24),
        hour_c3 = cos(6*pi*hour/24))

# to just select a single year of data
buffalo_data_all <- buffalo_data_all %>% filter(t1 < "2019-07-25 09:32:42 ACST")
buffalo_ids <- unique(buffalo_data_all$id)

# Timeline of buffalo data
buffalo_data_all |> ggplot(aes(x = t1, y = factor(id), colour = factor(id))) +
  geom_point(alpha = 0.1) +
  scale_y_discrete("Buffalo ID") +
  scale_x_datetime("Date") +
  scale_colour_viridis_d() +
  theme_bw() +
  theme(legend.position = "none")
```



Fitting the model with no harmonics

Creating a data matrix

First we create a data matrix to be provided to the model, and then we scale and centre the full data matrix, with respect to each of the columns. That means that all variables are scaled and centred *after* the data has been split into wet and dry season data, and also after creating the quadratic and harmonic terms (when using them).

```
months_wet <- c(1:4, 11:12)
buffalo_ids <- unique(buffalo_data_all$id)

# comment and uncomment the relevant lines to get either wet or dry season data
# buffalo_data <- buffalo_data_all %>% filter(month %in% months_wet) # wet season
buffalo_data <- buffalo_data_all %>% filter(!month %in% months_wet) # dry season

buffalo_data_matrix_unscaled <- buffalo_data %>% transmute(

  ndvi = ndvi_temporal,
  ndvi_sq = ndvi_temporal ^ 2,
  canopy = canopy_01,
  canopy_sq = canopy_01 ^ 2,
  slope = slope,
  herby = veg_herby,
  spatial_memory = kde_memory_density,
  spatial_memory_sq = kde_memory_density ^ 2,
```

```
  step_l = sl,
  log_step_l = log_sl,
  cos_turn_a = cos_ta)

buffalo_data_matrix_scaled <- scale(buffalo_data_matrix_unscaled)

# save the scaling values to recover the natural scale of the coefficients
# which is required for the simulations
# (so then environmental variables don't need to be scaled)
mean_vals <- attr(buffalo_data_matrix_scaled, "scaled:center")
sd_vals <- attr(buffalo_data_matrix_scaled, "scaled:scale")
scaling_attributes <- data.frame(variable = names(buffalo_data_matrix_unscaled),
                                 mean = mean_vals, sd = sd_vals)

# add the id, step_id columns and presence/absence columns to
# the scaled data matrix for model fitting
buffalo_data_scaled_0p <- data.frame(id = buffalo_data$id,
                                     step_id = buffalo_data$step_id,
                                     y = buffalo_data$y,
                                     buffalo_data_matrix_scaled)
```

Writing the formula to pass to the model

Formula with no harmonics

```
formula_twostep <- y ~

  ndvi +
  ndvi_sq +
  canopy +
  canopy_sq +
  slope +
  herby +
  spatial_memory +
  spatial_memory_sq +
  step_l +
  log_step_l +
  cos_turn_a +

  strata(step_id) +
  cluster(id)
```

Fitting the model

As we have already fitted the model, we will load it here, but if the model_fit file doesn't exist, it will run the model fitting code. Be careful here that if you change the model formula, you will need to delete or rename the model_fit file to re-run the model fitting code, otherwise it will just load the previous model.

```
if(file.exists("outputs/model_twostep_0p_harms_dry.rds")) {

  model_twostep_0p_harms <- readRDS("outputs/model_twostep_0p_harms_dry.rds")

} else {
```

```
  tic()
  model_twostep_0p_harms <- Ts.estim(formula = formula_twostep,
           data = buffalo_data_scaled_0p,
           all.m.1 = TRUE,
           D = "UN(1)",
           itermax = 10000)
  toc()

  # save model object
  saveRDS(model_twostep_0p_harms, file = "outputs/model_twostep_0p_harms_dry.rds")

  beep(sound = 2)

}
```

Check the fitted model

```
model_twostep_0p_harms
```

```
## Call:
## Ts.estim(formula = formula_twostep, data = buffalo_data_scaled_0p,
##     all.m.1 = TRUE, D = "UN(1)", itermax = 10000)
##
## beta coefficients:
##                      estimate        se
## ndvi                 0.675552  0.141881
## ndvi_sq             -0.691487  0.163979
## canopy               0.192100  0.063591
## canopy_sq           -0.237176  0.068656
## slope               -0.176174  0.077423
## herby               -0.036103  0.035722
## spatial_memory       2.249319  0.242152
## spatial_memory_sq   -1.464462  0.301126
## step_l               0.136149  0.052802
## log_step_l           0.002606  0.029736
## cos_turn_a          -0.014018  0.024101
##
## D = estimate of the between-cluster variance-covariance matrix D,
##     for the random coefficients only:
##                          ndvi    ndvi_sq      canopy   canopy_sq       slope       herby  spatial_mem
## ndvi                0.2332849  0.0000000  0.00000000  0.00000000  0.00000000  0.00000000       0.0000
## ndvi_sq             0.0000000  0.3165945  0.00000000  0.00000000  0.00000000  0.00000000       0.0000
## canopy              0.0000000  0.0000000  0.04031609  0.00000000  0.00000000  0.00000000       0.0000
## canopy_sq           0.0000000  0.0000000  0.00000000  0.04829964  0.00000000  0.00000000       0.0000
## slope               0.0000000  0.0000000  0.00000000  0.00000000  0.07453139  0.00000000       0.0000
## herby               0.0000000  0.0000000  0.00000000  0.00000000  0.00000000  0.01557572       0.0000
## spatial_memory      0.0000000  0.0000000  0.00000000  0.00000000  0.00000000  0.00000000       0.7036
## spatial_memory_sq   0.0000000  0.0000000  0.00000000  0.00000000  0.00000000  0.00000000       0.0000
## step_l              0.0000000  0.0000000  0.00000000  0.00000000  0.00000000  0.00000000       0.0000
## log_step_l          0.0000000  0.0000000  0.00000000  0.00000000  0.00000000  0.00000000       0.0000
## cos_turn_a          0.0000000  0.0000000  0.00000000  0.00000000  0.00000000  0.00000000       0.0000
##                    spatial_memory_sq      step_l  log_step_l   cos_turn_a
## ndvi                        0.000000  0.00000000  0.00000000  0.000000000
```

```
## ndvi_sq                   0.000000  0.00000000  0.00000000  0.000000000
## canopy                    0.000000  0.00000000  0.00000000  0.000000000
## canopy_sq                 0.000000  0.00000000  0.00000000  0.000000000
## slope                     0.000000  0.00000000  0.00000000  0.000000000
## herby                     0.000000  0.00000000  0.00000000  0.000000000
## spatial_memory            0.000000  0.00000000  0.00000000  0.000000000
## spatial_memory_sq         1.057871  0.00000000  0.00000000  0.000000000
## step_l                    0.000000  0.03531749  0.00000000  0.000000000
## log_step_l                0.000000  0.00000000  0.01079844  0.000000000
## cos_turn_a                0.000000  0.00000000  0.00000000  0.007122307
```

```
# output model components
# model_twostep_0p_harms$beta
# model_twostep_0p_harms$se
# model_twostep_0p_harms$vcov
# diag(model_twostep_0p_harms$D) # between cluster variance
# model_twostep_0p_harms$r.effect # individual estimates

# create a dataframe of the coefficients and their scaling attributes
coefs_clr <- data.frame(coefs = names(model_twostep_0p_harms$beta),
                        value = model_twostep_0p_harms$beta)
coefs_clr$scale_sd <- scaling_attributes$sd
coefs_clr <- coefs_clr %>% mutate(value_nat = value / scale_sd)
head(coefs_clr)
```

```
##               coefs       value  scale_sd   value_nat
## ndvi           ndvi  0.67555225 0.1429054  4.72727048
## ndvi_sq     ndvi_sq -0.69148694 0.1019530 -6.78240682
## canopy       canopy  0.19210019 0.1947719  0.98628312
## canopy_sq canopy_sq -0.23717641 0.1572120 -1.50864038
## slope         slope -0.17617399 0.9580647 -0.18388527
## herby         herby -0.03610302 0.4783789 -0.07546951
```

Reconstructing the hourly coefficients with no harmonics. This step isn't necessary as we already have the coefficients, and have already rescaled them in the dataframe we created above. But as we are also fitting harmonic models and recover their coefficients across time, we have used the same approach here so then we can plot them together to illustrate the static/dynamic nature of the models. It also means that we can use the same simulation code (which indexes across the hour of the day), and just change the data frame of coefficients (as it will index across the coefficients of the static model but they won't change).

First we create a sequence of time to reconstruct the temporally dynamic coefficients. We will use 0.1 hourly increments (therefore every 6 minutes) to get smooth curves when we plot. When we are simulating steps to create trajectories however, we will use hourly increments as that matches the time between steps of our data and model fitting.

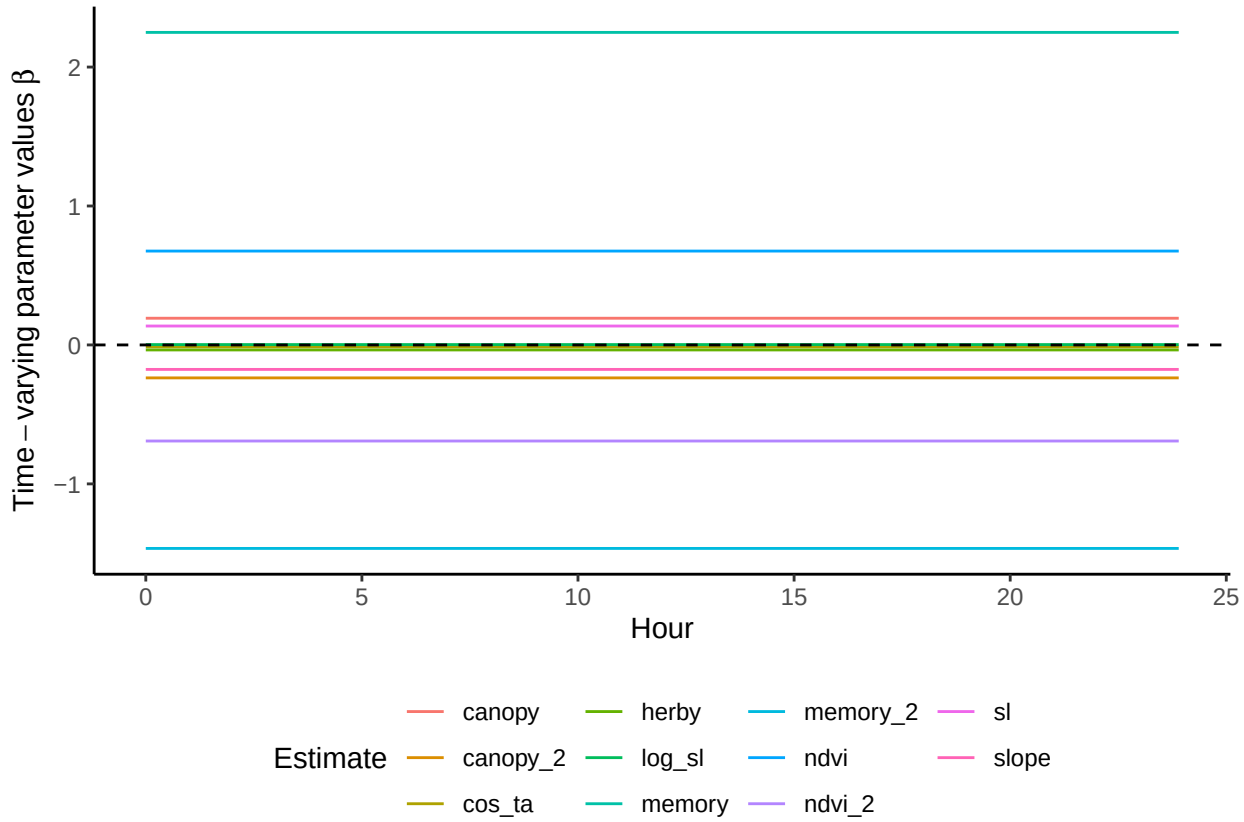
```
# hour <- seq(0,23,1)
hour <- seq(0,23.9,0.1)

hour_harmonics_df <- data.frame("linear_term" = rep(1, length(hour)))

harmonics_scaled_df_0p <- data.frame(
  "hour" = hour,
  "ndvi" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "ndvi_2" = as.numeric(
```

```
    coefs_clr %>% dplyr::filter(grepl("ndvi_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "slope" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("slope", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "herby" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("herby", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("step_l", coefs) & !grepl("log", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "log_sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("log_step_l", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "cos_ta" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("cos", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))))

harmonics_scaled_long_0p <- pivot_longer(harmonics_scaled_df_0p,
                                         cols = !1,
                                         names_to = "coef")
```

Plotting the results - scaled coefficients

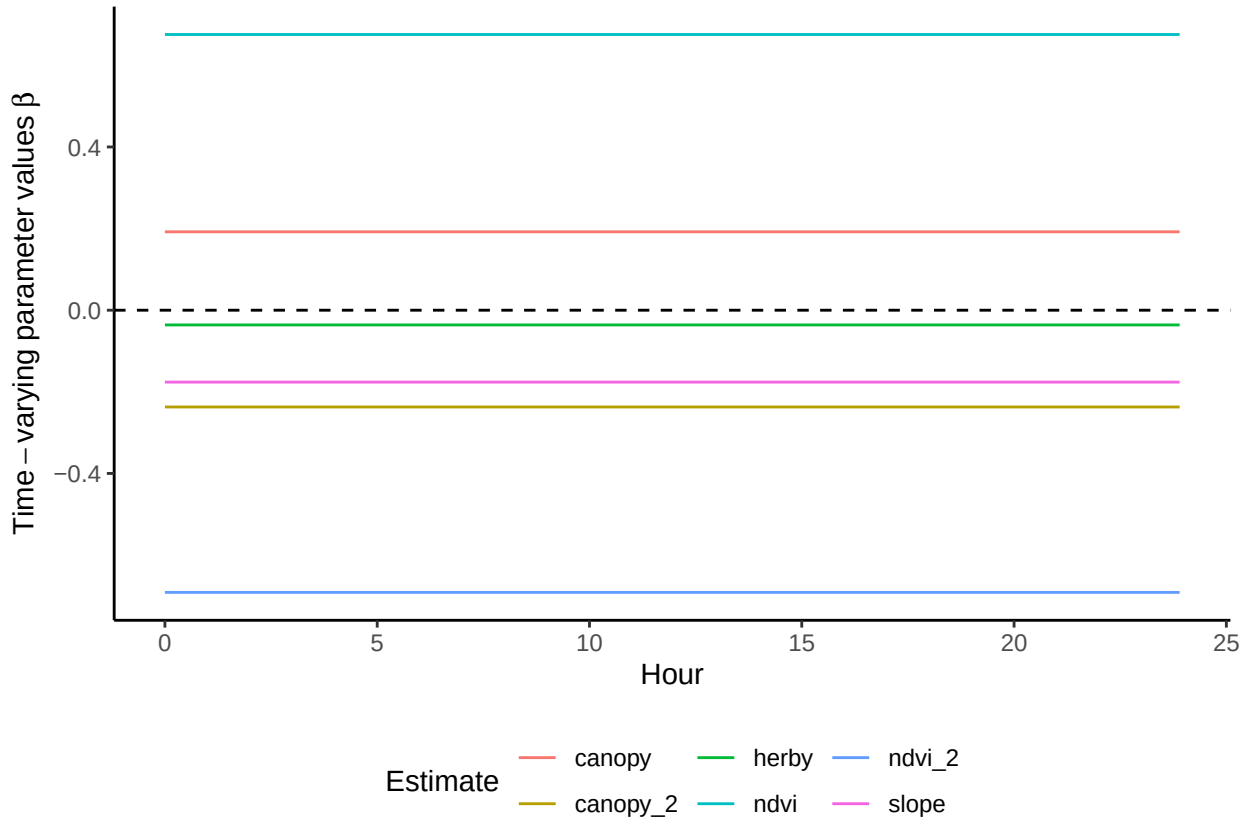
Here we are showing the coefficients across time, but as the model does not have temporal dynamics, the coefficients are static through time.

```
ggplot() +
    geom_path(data = harmonics_scaled_long_0p,
              aes(x = hour, y = value, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
    scale_x_continuous("Hour") +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



Plot just the habitat covariates

```
ggplot() +
    geom_path(data = harmonics_scaled_long_0p %>%
              filter(!coef %in% c("sl", "log_sl", "cos_ta", "memory", "memory_2")),
              aes(x = hour, y = value, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
    scale_x_continuous("Hour") +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



Reconstructing the natural scale parameters

```
harmonics_nat_df_0p <- data.frame(
  "hour" = hour,
  "ndvi" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "ndvi_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "slope" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("slope", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "herby" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("herby", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory", coefs)& !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
```

```
  "memory_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("step_l", coefs) & !grepl("log", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "log_sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("log_step_l", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "cos_ta" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("cos", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))))
```

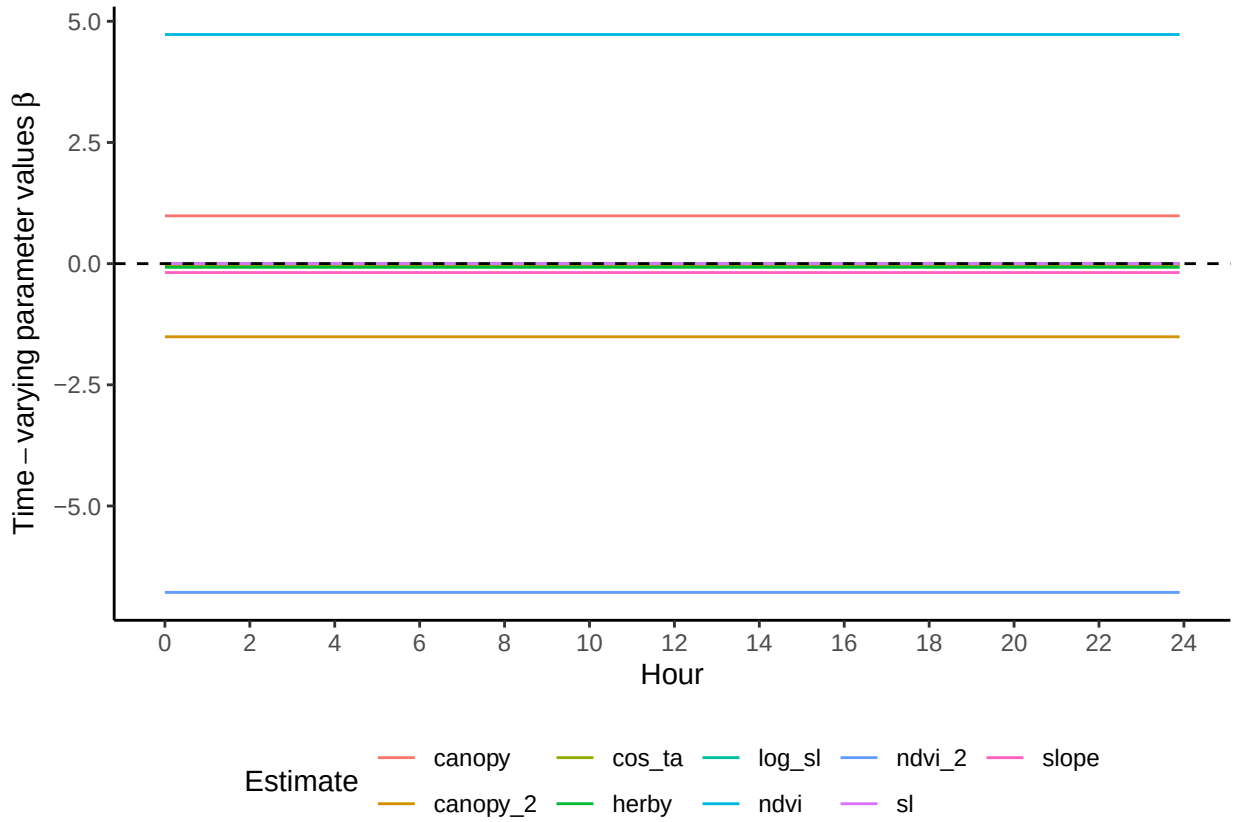
Updating the Gamma and von Mises distributions from the tentative distributions (from Fieberg et al 2021: Appendix C)

```
# from the step generation script
tentative_shape <- 0.438167
tentative_scale <- 534.3507
tentative_kappa <- 0.1848126

hour_coefs_nat_df_0p <- harmonics_nat_df_0p %>%
  mutate(shape = tentative_shape + log_sl,
         scale = 1/((1/tentative_scale) - sl),
         kappa = tentative_kappa + cos_ta)

# save the coefficients to use in the simulations
# write_csv(hour_coefs_nat_df_0p,
#           paste0("outputs/TwoStep_0pDaily_coefs_dry_",Sys.Date(), ".csv"))

# turning into a long data frame
hour_coefs_nat_long_0p <- pivot_longer(hour_coefs_nat_df_0p,
                                       cols = !1,
                                       names_to = "coef")
```

Plotting the temporally dynamic (in this case static) natural-scale external selection parameters

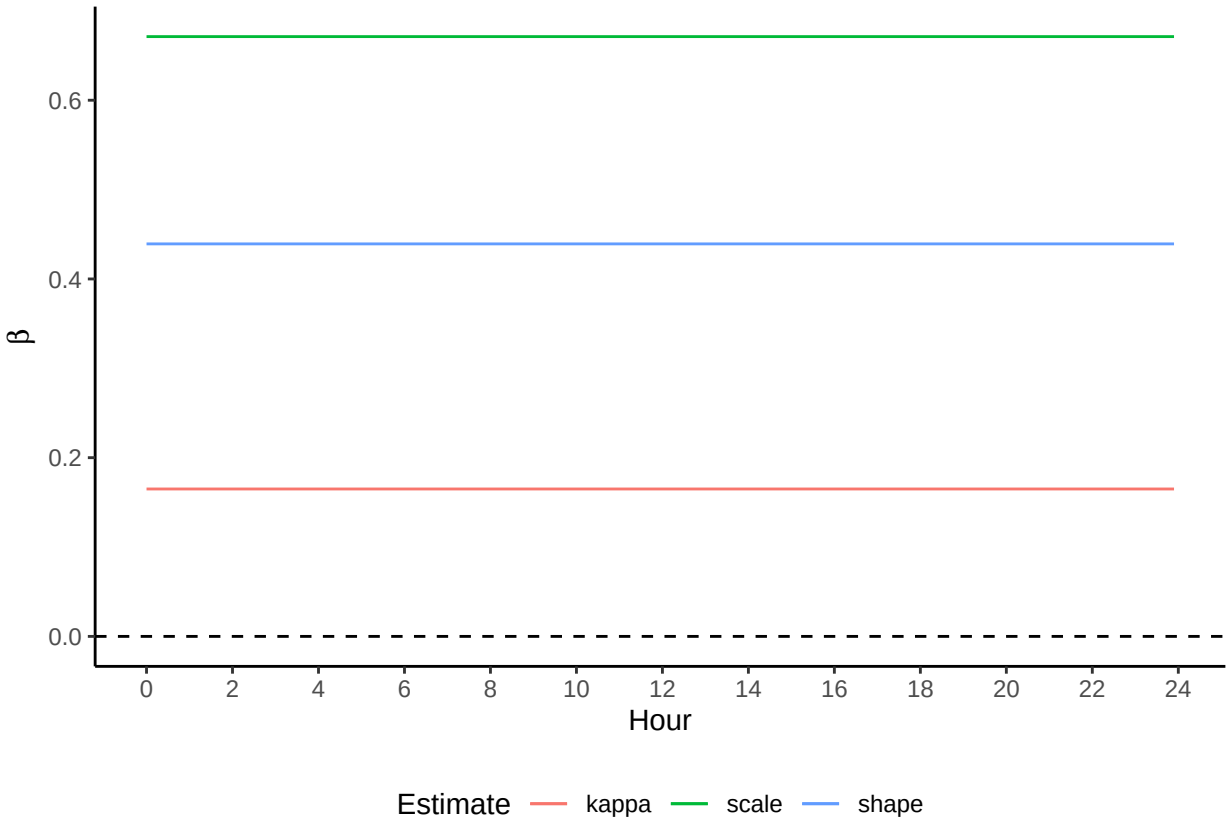
Now that the coefficients are in their natural scales, the memory parameters particularly will be huge (as the previous space use density is very small), so we are not plotting all coefficients.

```
ggplot() +
  geom_path(data = hour_coefs_nat_long_0p %>%
              filter(!coef %in% c("shape", "scale", "kappa", "memory", "memory_2")),
            aes(x = hour, y = value, colour = coef)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_color_discrete("Estimate") +
  theme_classic() +
  theme(legend.position = "bottom")
```



Plotting the natural-scale movement parameters

```
ggplot() +
    geom_path(data = hour_coefs_nat_long_0p %>%
                filter(coef %in% c("shape", "kappa")),
                aes(x = hour, y = value, colour = coef)) +
  geom_path(data = hour_coefs_nat_long_0p %>%
                filter(coef == "scale"),
                aes(x = hour, y = value/1000, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(beta)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



Sampling from movement kernel

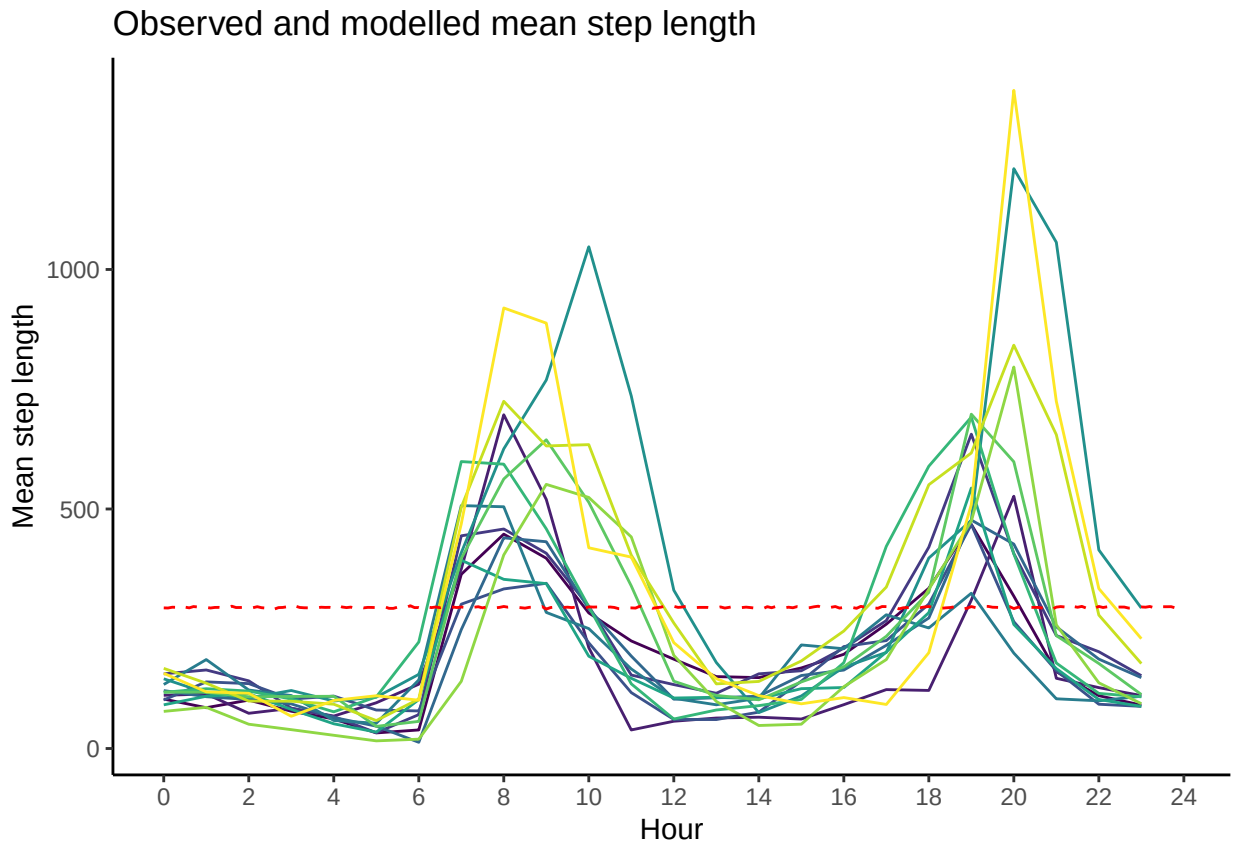
Here we sample from the movement kernel to generate a distribution of step lengths for each hour of the day, to assess how well it matches the observed step lengths.

```
# summarise the observed step lengths by hour
movement_summary <- buffalo_data %>% filter(y == 1) %>% group_by(id, hour) %>%
  summarise(mean_sl = mean(sl), median_sl = median(sl))
```

```
## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.
```

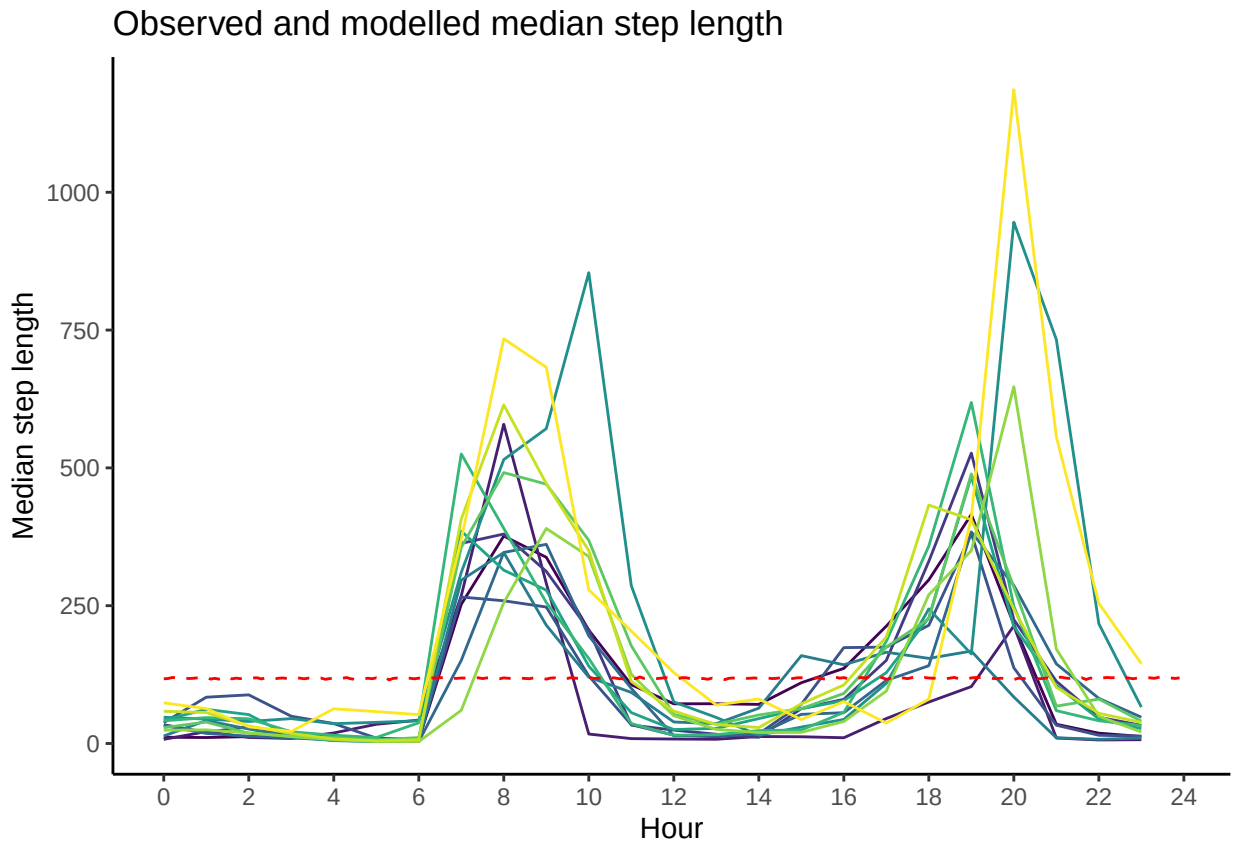
```
# number of samples at each hour (more = smoother plotting, but slower)
n <- 1e5

gamma_dist_list <- vector(mode = "list", length = nrow(hour_coefs_nat_df_0p))
gamma_mean <- c()
gamma_median <- c()
gamma_ratio <- c()

for(hour_no in 1:nrow(hour_coefs_nat_df_0p)) {

  gamma_dist_list[[hour_no]] <- rgamma(n, shape = hour_coefs_nat_df_0p$shape[hour_no],
                                       scale = hour_coefs_nat_df_0p$scale[hour_no])
  gamma_mean[hour_no] <- mean(gamma_dist_list[[hour_no]])
  gamma_median[hour_no] <- median(gamma_dist_list[[hour_no]])
  gamma_ratio[hour_no] <- gamma_mean[hour_no] / gamma_median[hour_no]
```

```
}

gamma_df_0p <- data.frame(model = "0p",
                          hour = hour_coefs_nat_df_0p$hour,
                          mean = gamma_mean,
                          median = gamma_median,
                          ratio = gamma_ratio)

mean_sl_0p <- ggplot() +
  geom_path(data = movement_summary,
            aes(x = hour, y = mean_sl, colour = factor(id))) +
  geom_path(data = gamma_df_0p,
            aes(x = hour, y = mean), colour = "red", linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Mean step length") +
  scale_colour_viridis_d("Buffalo") +
  ggtitle("Observed and modelled mean step length") + #,
          # subtitle = "Two pairs of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

mean_sl_0p
```




```
median_sl_0p <- ggplot() +
  geom_path(data = movement_summary,
            aes(x = hour, y = median_sl, colour = factor(id))) +
```

```
  geom_path(data = gamma_df_0p, aes(x = hour, y = median),
            colour = "red", linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Median step length") +
  scale_colour_viridis_d("Buffalo") +
  ggtitle("Observed and modelled median step length") + #,
          # subtitle = "Two pairs of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

median_sl_0p
```




```
# comparing the mean and median step lengths across all hours
# across the hours by individual buffalo
buffalo_data_all %>% filter(y == 1) %>%  group_by(id) %>%
  summarise(mean_sl = mean(sl),
            median_sl = median(sl),
            ratio = mean_sl/median_sl)
```

```
## # A tibble: 13 x 4
##       id mean_sl median_sl ratio
##    <dbl>   <dbl>     <dbl> <dbl>
##  1  2005    204.      98.0  2.08
##  2  2014    142.      15.9  8.94
##  3  2018    249.     103.   2.41
##  4  2021    183.      94.8  1.93
```

```
##  5  2022    216.      78.4  2.75
##  6  2024    229.      77.9  2.94
##  7  2039    357.     124.   2.87
##  8  2154    198.      95.6  2.07
##  9  2158    219.      84.8  2.58
## 10  2223    249.      80.2  3.10
## 11  2327    200.      51.3  3.91
## 12  2387    327.     111.   2.95
## 13  2393    322.     127.   2.53
```

```
# all buffalo
buffalo_data_all %>% filter(y == 1) %>%
  summarise(mean_sl = mean(sl),
            median_sl = median(sl),
            ratio = mean_sl/median_sl)
```

```
## # A tibble: 1 x 3
##   mean_sl median_sl ratio
##     <dbl>     <dbl> <dbl>
## 1    238.      86.9  2.74
```

```
# fitted model
gamma_df_0p |> summarise(mean_mean = mean(mean),
                         median_mean = mean(median),
                         ratio_mean = mean_mean/median_mean)
```

```
##   mean_mean median_mean ratio_mean
## 1  294.6372    118.4036   2.488413
```

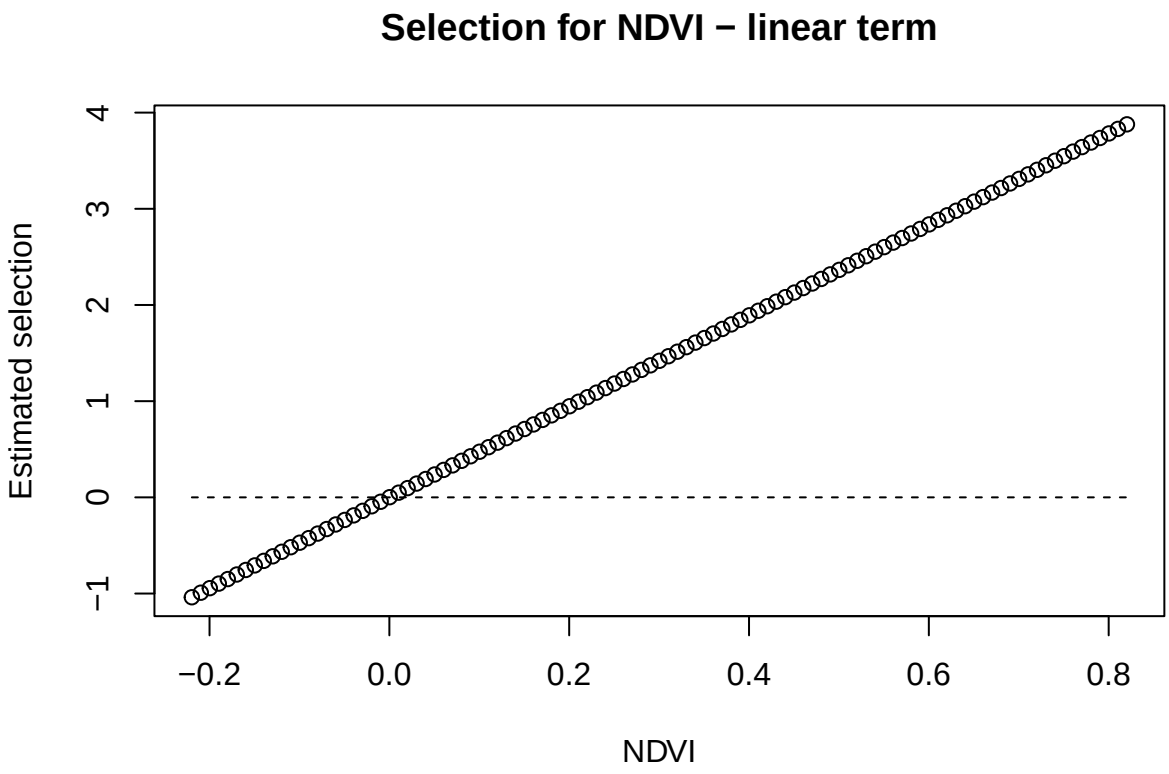
Creating selection surfaces

As we have both quadratic and harmonic terms in the model, we can reconstruct a 'selection surface' to visualise how the coefficient changes through time.

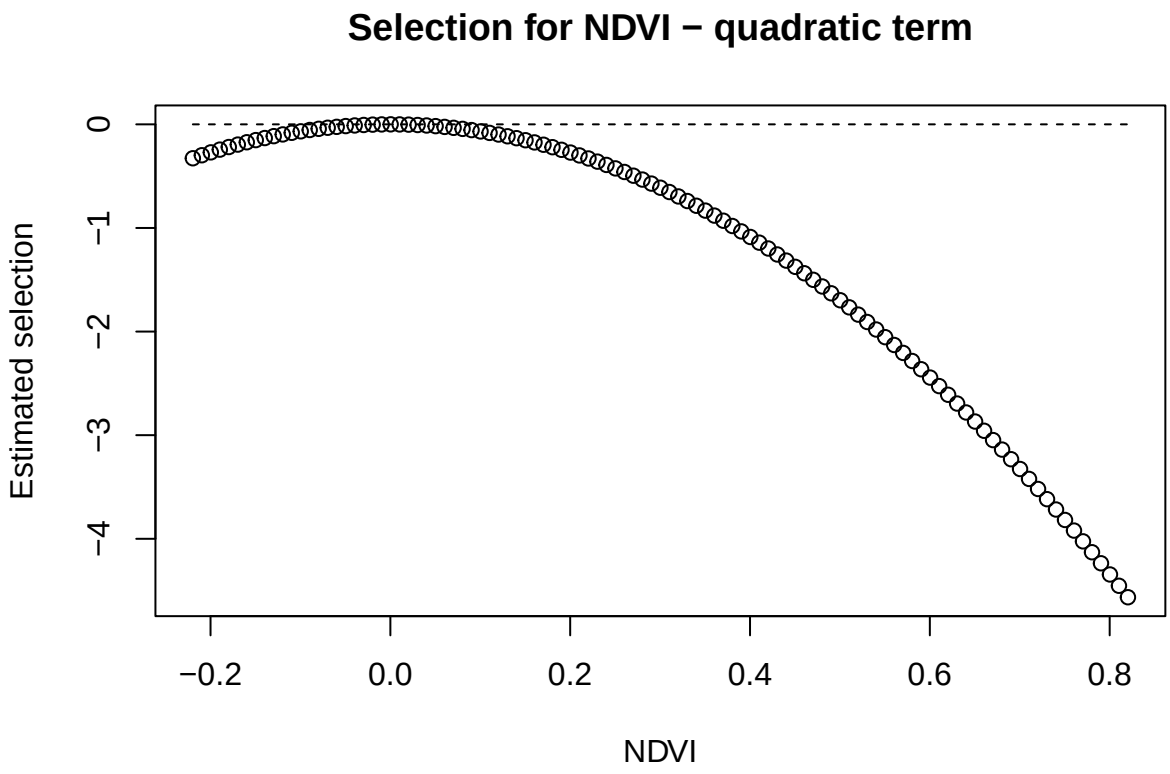
To illustrate, if we don't have temporal dynamics (as is the case for this model), then we have a coefficient for the linear term and a coefficient for the quadratic term. Using these, we can plot the selection curve at the scale of the environmental variable (in this case NDVI).

Using the natural scale coefficients from the model:

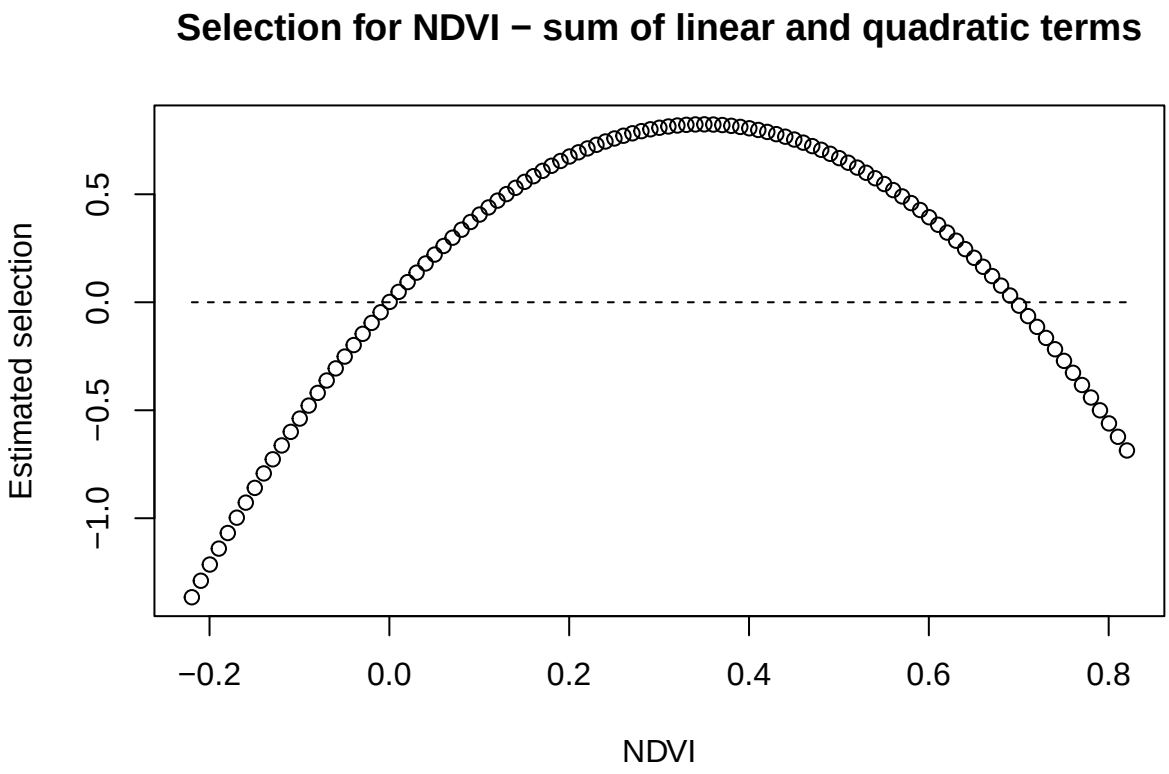
```
# first get a sequence of NDVI values,
# starting from the minimum observed in the data to the maximum
ndvi_min <- min(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_max <- max(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_seq <- seq(ndvi_min, ndvi_max, by = 0.01)

# take the coefficients from the model and calculation the selection value
# for every NDVI value in this sequence

# we can separate to the linear term
ndvi_linear_selection <- hour_coefs_nat_df_0p$ndvi[1] * ndvi_seq
plot(x = ndvi_seq, y = ndvi_linear_selection,
     main = "Selection for NDVI - linear term",
     xlab = "NDVI", ylab = "Estimated selection")
lines(ndvi_seq, rep(0,length(ndvi_seq)), lty = "dashed")
```




```
# and the quadratic term
ndvi_quadratic_selection <- (hour_coefs_nat_df_0p$ndvi_2[1] * (ndvi_seq ^ 2))
plot(x = ndvi_seq, y = ndvi_quadratic_selection,
     main = "Selection for NDVI - quadratic term",
     xlab = "NDVI", ylab = "Estimated selection")
lines(ndvi_seq, rep(0,length(ndvi_seq)), lty = "dashed")
```




```
# and the sum of both
ndvi_sum_selection <- ndvi_linear_selection + ndvi_quadratic_selection
plot(x = ndvi_seq, y = ndvi_sum_selection,
     main = "Selection for NDVI - sum of linear and quadratic terms",
     xlab = "NDVI", ylab = "Estimated selection")
lines(ndvi_seq, rep(0,length(ndvi_seq)), lty = "dashed")
```



When there are not temporal dynamics, then this quadratic curve will be the same throughout the day, but when we have temporally dynamic coefficients for both the linear term and the quadratic term, then we will have a different curve for each hour of the day, which we can visualise as a selection surface. An example of that is shown for the model with 1 pair of harmonic terms.

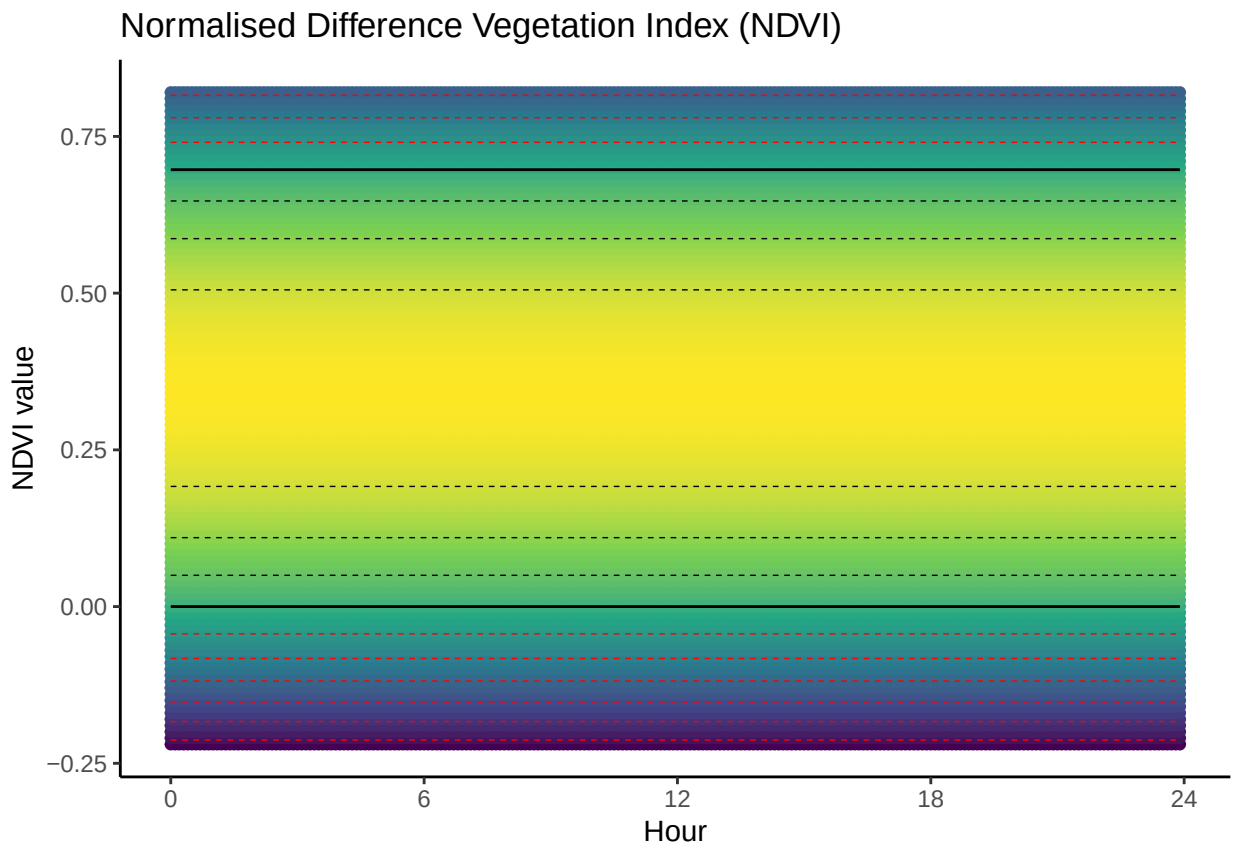
However, we will still construct the selection surfaces for the model with no temporal dynamics, just to illustrate the concept, and then to compare against the plots with dynamic coefficients from the models with harmonic terms.

Contour plot for NDVI

We use the same process as above, but now we index across the linear and quadratic terms for every time point, which in this case are 0.1 hours for smooth plots.

```
ndvi_min <- min(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_max <- max(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_seq <- seq(ndvi_min, ndvi_max, by = 0.01)

# Create empty data frame
ndvi_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_0p),
                                       nrow = length(ndvi_seq)))
for(i in 1:nrow(hour_coefs_nat_df_0p)) {
  # Assign the vector as a column to the dataframe
  ndvi_fresponse_df[,i] <- (hour_coefs_nat_df_0p$ndvi[i] * ndvi_seq) +
    (hour_coefs_nat_df_0p$ndvi_2[i] * (ndvi_seq ^ 2))
}
```

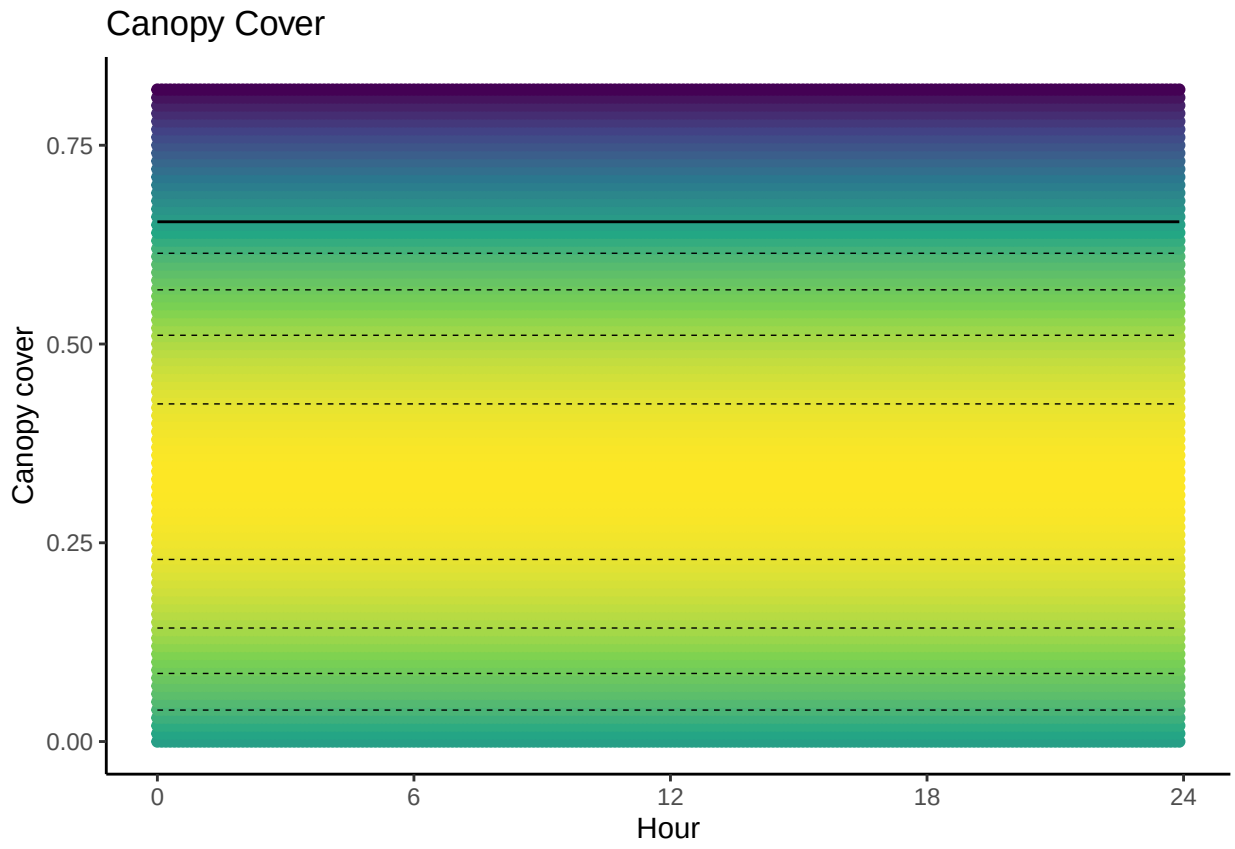
```r
ndvi_fresponse_df <- data.frame(ndvi_seq, ndvi_fresponse_df)
colnames(ndvi_fresponse_df) <- c("ndvi", hour)
ndvi_fresponse_long <- pivot_longer(ndvi_fresponse_df,
                                    cols = !1, names_to = "hour")

ndvi_contour_max <- max(ndvi_fresponse_long$value) # 0.7890195
ndvi_contour_min <- min(ndvi_fresponse_long$value) # -0.7945691
ndvi_contour_increment <- (ndvi_contour_max-ndvi_contour_min)/10

ndvi_quad_0p <- ggplot(data = ndvi_fresponse_long,
                       aes(x = as.numeric(hour), y = ndvi)) +
  geom_point(aes(colour = value)) + # colour = "white"
  geom_contour(aes(z = value),
               breaks = seq(ndvi_contour_increment,
                            ndvi_contour_max,
                            ndvi_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-ndvi_contour_increment,
                            ndvi_contour_min,
                            -ndvi_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("NDVI value", breaks = seq(-1, 1, 0.25)) +
  scale_colour_viridis_c("Selection") +
  ggtitle("Normalised Difference Vegetation Index (NDVI)") +
  theme_classic() +
  theme(legend.position = "none")

ndvi_quad_0p
```



Canopy cover

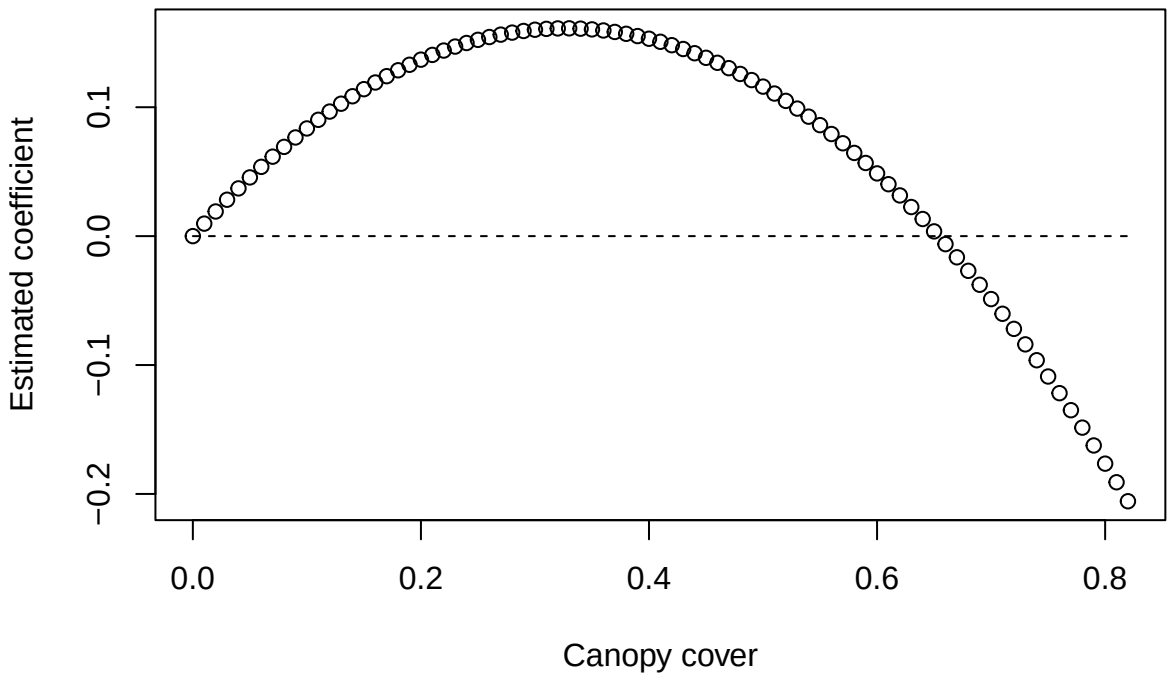
```
canopy_min <- min(buffalo_data$canopy_01, na.rm = TRUE)
canopy_max <- max(buffalo_data$canopy_01, na.rm = TRUE)
canopy_seq <- seq(canopy_min, canopy_max, by = 0.01)

# Create empty data frame
canopy_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_0p),
                                         nrow = length(canopy_seq)))
for(i in 1:nrow(hour_coefs_nat_df_0p)) {
  # Assign the vector as a column to the dataframe
  canopy_fresponse_df[,i] <- (hour_coefs_nat_df_0p$canopy[i] * canopy_seq) +
    (hour_coefs_nat_df_0p$canopy_2[i] * (canopy_seq ^ 2))
}

canopy_fresponse_df <- data.frame(canopy_seq, canopy_fresponse_df)
colnames(canopy_fresponse_df) <- c("canopy", hour)
canopy_fresponse_long <- pivot_longer(canopy_fresponse_df,
                                      cols = !1,
                                      names_to = "hour")

canopy_contour_min <- min(canopy_fresponse_long$value) # 0
canopy_contour_max <- max(canopy_fresponse_long$value) # 2.181749
canopy_contour_increment <- (canopy_contour_max-canopy_contour_min)/10
```

```
canopy_quad_0p <- ggplot(data = canopy_fresponse_long, aes(x = as.numeric(hour),
                                                           y = canopy)) +
  geom_point(aes(colour = value)) +
  geom_contour(aes(z = value),
               breaks = seq(canopy_contour_increment, canopy_contour_max,
                            canopy_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  # geom_contour(aes(z = value),
               # breaks = seq(-canopy_contour_increment, canopy_contour_min,
               #              -canopy_contour_increment),
  #              colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("Canopy cover", breaks = seq(0, 1, 0.25)) +
  scale_colour_viridis_c("Selection") +
  ggtitle("Canopy Cover") +
  theme_classic() +
  theme(legend.position = "none")

canopy_quad_0p
```



We can also just plot the quadratic curve in two dimensions.

```
plot(canopy_fresponse_df[,1], canopy_fresponse_df[,2],
     xlab = "Canopy cover", ylab = "Estimated coefficient")
lines(canopy_fresponse_df[,1], rep(0,length(canopy_fresponse_df[,1])), lty = "dashed")
```

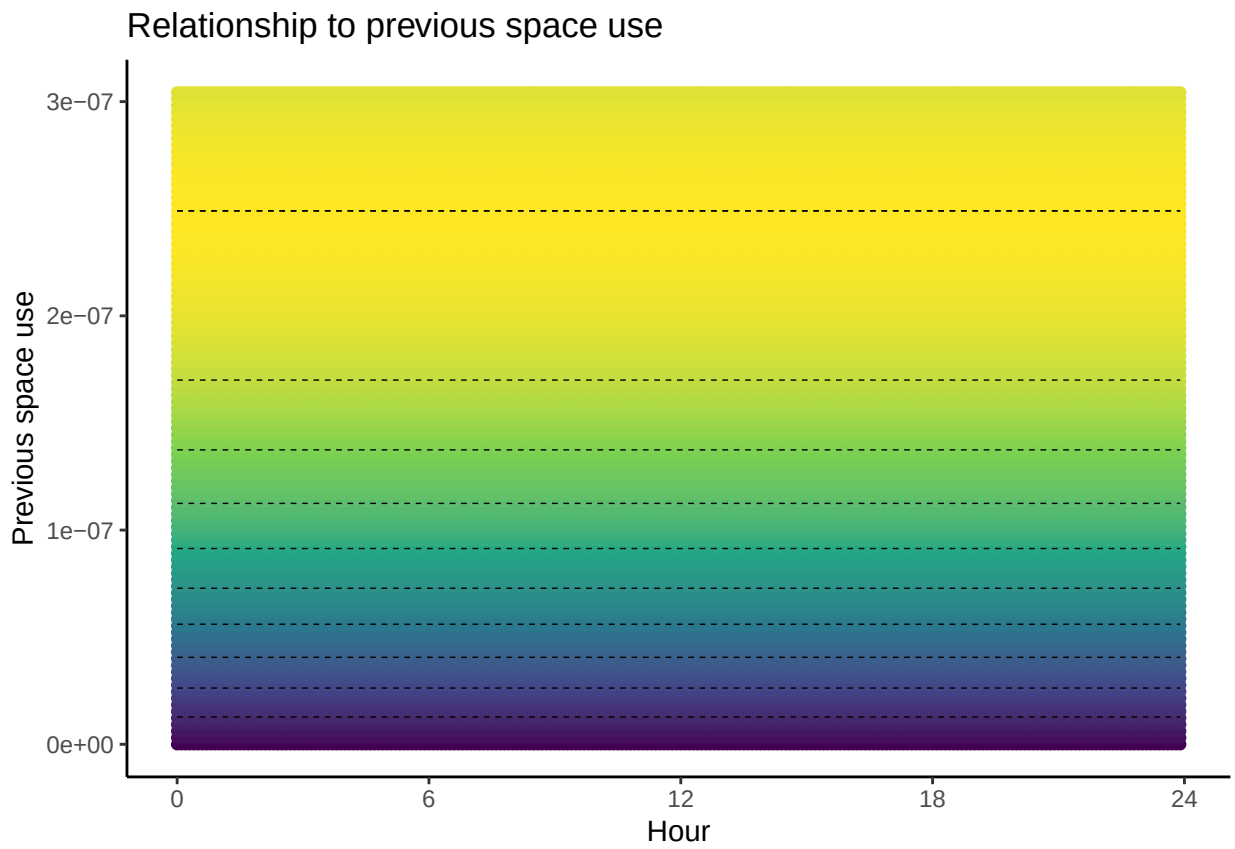


Previous space use density

```
memory_min <- min(buffalo_data$kde_memory_density, na.rm = TRUE)
# memory_max <- max(buffalo_data$kde_memory_density, na.rm = TRUE)
memory_max <- quantile(buffalo_data |> filter(y == 1) |>
                         pull(kde_memory_density), probs = 0.95, na.rm = TRUE)
memory_seq <- seq(memory_min, memory_max, length.out = 100)

# Create empty data frame
memory_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_0p),
                                         nrow = length(memory_seq)))
for(i in 1:nrow(hour_coefs_nat_df_0p)) {
  # Assign the vector as a column to the dataframe
  memory_fresponse_df[,i] <- (hour_coefs_nat_df_0p$memory[i] * memory_seq) +
    (hour_coefs_nat_df_0p$memory_2[i] * (memory_seq ^ 2))
}

memory_fresponse_df <- data.frame(memory_seq, memory_fresponse_df)
colnames(memory_fresponse_df) <- c("memory", hour)
memory_fresponse_long <- pivot_longer(memory_fresponse_df,
                                      cols = !1,
                                      names_to = "hour")

memory_contour_min <- min(memory_fresponse_long$value) # 0
memory_contour_max <- max(memory_fresponse_long$value) # 2.181749
```

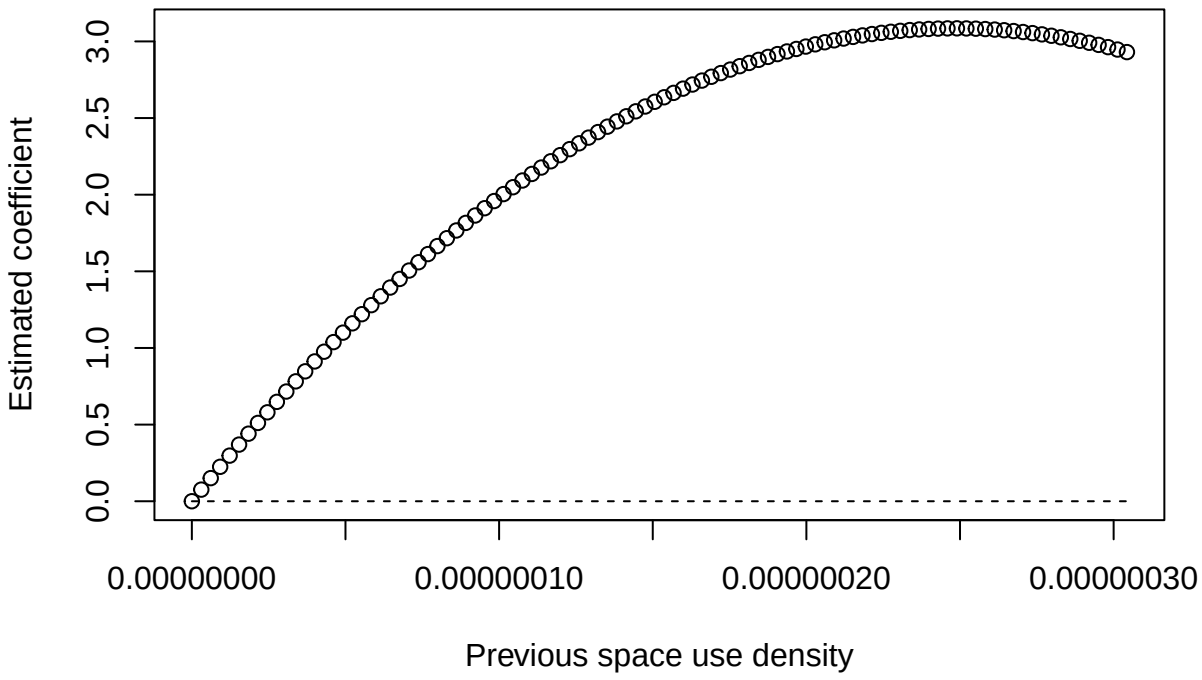
```r
memory_contour_increment <- (memory_contour_max-memory_contour_min)/10

memory_quad_0p <- ggplot(data = memory_fresponse_long,
                         aes(x = as.numeric(hour), y = memory)) +
  geom_point(aes(colour = value)) +
  geom_contour(aes(z = value),
               breaks = seq(memory_contour_increment, memory_contour_max,
                            memory_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-memory_contour_increment,
                            min(-memory_contour_increment,
                                memory_contour_min), -memory_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("Previous space use", labels = scientific) +
  scale_colour_viridis_c("Selection") +
  ggtitle("Relationship to previous space use") +
  theme_classic() +
  theme(legend.position = "none")

memory_quad_0p
```

```
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```



We can also just plot the quadratic curve in two dimensions

```
plot(memory_fresponse_df[,1], memory_fresponse_df[,2],
     xlab = "Previous space use density", ylab = "Estimated coefficient")
lines(memory_fresponse_df[,1], rep(0,length(memory_fresponse_df[,1])), lty = "dashed")
```



Model fitting with a single pair of harmonics

Creating the data matrix

Now we start to add the harmonic terms. As we have already created the harmonic terms for the hour of the day (s1, c1, s2, etc), we just interact (multiply) these with each of the covariates, including the quadratic terms. As before, we create the data matrix with all quadratic and harmonic terms, and *then* scale the matrix by each column and store the scaling and centering variables to reconstruct the natural scale coefficients.

```
months_wet <- c(1:4, 11:12)
buffalo_ids <- unique(buffalo_data_all$id)

# buffalo_data <- buffalo_data_all %>% filter(month %in% months_wet) # wet season
buffalo_data <- buffalo_data_all %>% filter(!month %in% months_wet) # dry season

buffalo_data_matrix_unscaled <- buffalo_data %>% transmute(

  # the 'linear' term
  ndvi = ndvi_temporal,
  # interact with the harmonic terms
  ndvi_s1 = ndvi_temporal * hour_s1,
  ndvi_c1 = ndvi_temporal * hour_c1,

  ndvi_sq = ndvi_temporal ^ 2,
  ndvi_sq_s1 = (ndvi_temporal ^ 2) * hour_s1,
  ndvi_sq_c1 = (ndvi_temporal ^ 2) * hour_c1,
```

```
  canopy = canopy_01,
  canopy_s1 = canopy_01 * hour_s1,
  canopy_c1 = canopy_01 * hour_c1,

  canopy_sq = canopy_01 ^ 2,
  canopy_sq_s1 = (canopy_01 ^ 2) * hour_s1,
  canopy_sq_c1 = (canopy_01 ^ 2) * hour_c1,

  slope = slope,
  slope_s1 = slope * hour_s1,
  slope_c1 = slope * hour_c1,

  herby = veg_herby,
  herby_s1 = veg_herby * hour_s1,
  herby_c1 = veg_herby * hour_c1,

  spatial_memory = kde_memory_density,
  spatial_memory_s1 = kde_memory_density * hour_s1,
  spatial_memory_c1 = kde_memory_density * hour_c1,

  spatial_memory_sq = kde_memory_density ^ 2,
  spatial_memory_sq_s1 = (kde_memory_density ^ 2) * hour_s1,
  spatial_memory_sq_c1 = (kde_memory_density ^ 2) * hour_c1,

  step_l = sl,
  step_l_s1 = sl * hour_s1,
  step_l_c1 = sl * hour_c1,

  log_step_l = log_sl,
  log_step_l_s1 = log_sl * hour_s1,
  log_step_l_c1 = log_sl * hour_c1,

  cos_turn_a = cos_ta,
  cos_turn_a_s1 = cos_ta * hour_s1,
  cos_turn_a_c1 = cos_ta * hour_c1)

buffalo_data_matrix_scaled <- scale(buffalo_data_matrix_unscaled)

mean_vals <- attr(buffalo_data_matrix_scaled, "scaled:center")
sd_vals <- attr(buffalo_data_matrix_scaled, "scaled:scale")
scaling_attributes <- data.frame(variable = names(buffalo_data_matrix_unscaled),
                                 mean = mean_vals, sd = sd_vals)

buffalo_data_scaled_1p <- data.frame(id = buffalo_data$id,
                                     step_id = buffalo_data$step_id,
                                     y = buffalo_data$y,
                                     buffalo_data_matrix_scaled)
```

Formula with one pair of harmonics

```
formula_twostep <- y ~
```

```
  ndvi +
  ndvi_s1 +
  ndvi_c1 +

  ndvi_sq +
  ndvi_sq_s1 +
  ndvi_sq_c1 +

  canopy +
  canopy_s1 +
  canopy_c1 +

  canopy_sq +
  canopy_sq_s1 +
  canopy_sq_c1 +

  slope +
  slope_s1 +
  slope_c1 +

  herby +
  herby_s1 +
  herby_c1 +

  spatial_memory +
  spatial_memory_s1 +
  spatial_memory_c1 +

  spatial_memory_sq +
  spatial_memory_sq_s1 +
  spatial_memory_sq_c1 +

  step_l +
  step_l_s1 +
  step_l_c1 +

  log_step_l +
  log_step_l_s1 +
  log_step_l_c1 +

  cos_turn_a +
  cos_turn_a_s1 +
  cos_turn_a_c1 +

  strata(step_id) +
  cluster(id)
```

Fitting the model

As we have already fitted the model, we will load it here, but if the model_fit file doesn't exist, it will run the model fitting code. Be careful here that if you change the model formula, you will need to delete or rename the model_fit file to re-run the model fitting code, otherwise it will just load the previous model.

```
if(file.exists("outputs/model_twostep_1p_harms_dry.rds")) {

  model_twostep_1p_harms <- readRDS("outputs/model_twostep_1p_harms_dry.rds")

} else {

  tic()
  model_twostep_1p_harms <- Ts.estim(formula = formula_twostep,
           data = buffalo_data_scaled_1p,
           all.m.1 = TRUE,
           D = "UN(1)",
           itermax = 10000)
  toc()

  # save model object
  saveRDS(model_twostep_1p_harms, file = "outputs/model_twostep_1p_harms_dry.rds")

  beep(sound = 2)

}
```

Reading in saved model fits

```
# model_twostep_1p_harms

# model_twostep_1p_harms
# model_twostep_1p_harms$beta
# model_twostep_1p_harms$se
# model_twostep_1p_harms$vcov
# diag(model_twostep_1p_harms$D) # between cluster variance
# model_twostep_1p_harms$r.effect # individual estimates

coefs_clr <- data.frame(coefs = names(model_twostep_1p_harms$beta),
                        value = model_twostep_1p_harms$beta)
coefs_clr$scale_sd <- scaling_attributes$sd
coefs_clr <- coefs_clr %>% mutate(value_nat = value / scale_sd)
head(coefs_clr)
```

```
##                  coefs       value  scale_sd   value_nat
## ndvi              ndvi  1.00300292 0.1429054   7.0186519
## ndvi_s1        ndvi_s1  0.40450985 0.2371431   1.7057629
## ndvi_c1        ndvi_c1 -0.81286409 0.2426054  -3.3505602
## ndvi_sq        ndvi_sq -1.12407329 0.1019530 -11.0254031
## ndvi_sq_s1  ndvi_sq_s1 -0.24661277 0.1065570  -2.3143737
## ndvi_sq_c1  ndvi_sq_c1  0.03730274 0.1105880   0.3373127
```

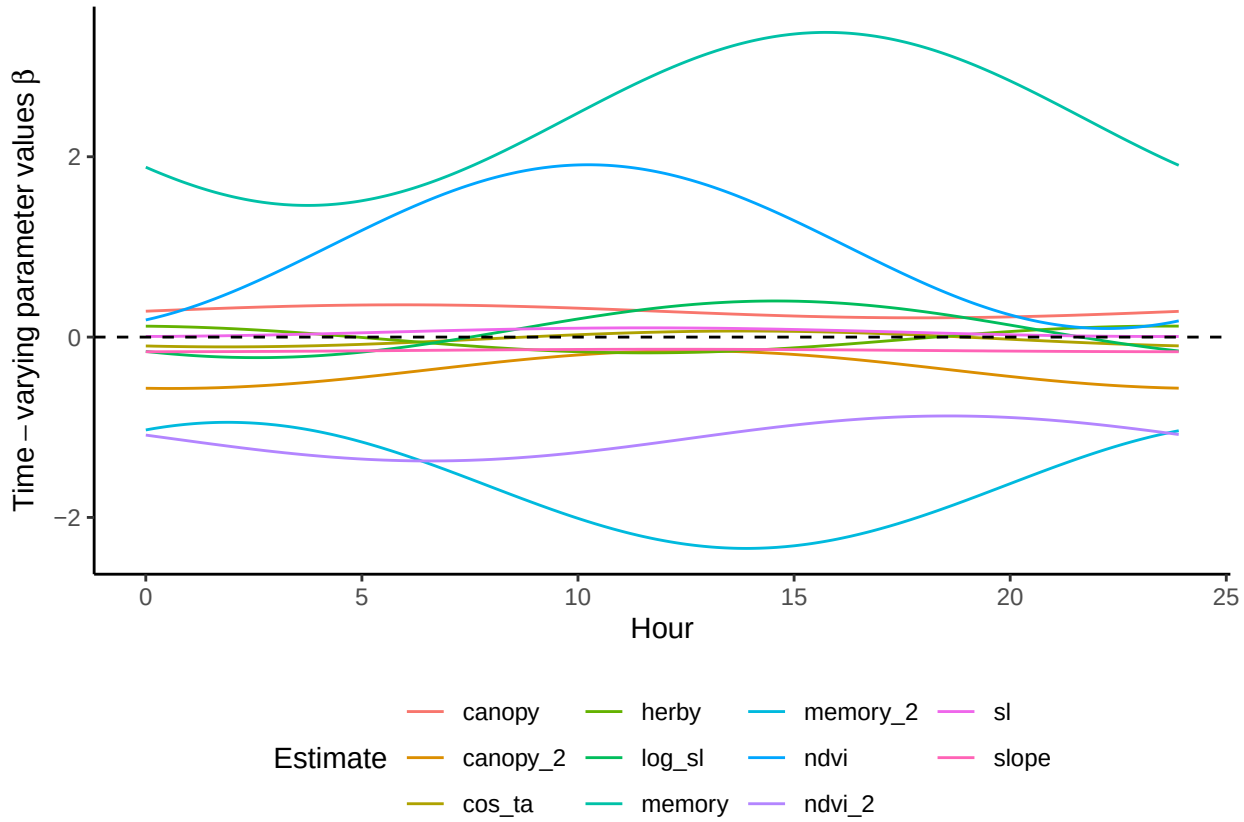
Reconstructing coefficients with two pairs of harmonics, with quadratic terms

```
# hour <- seq(0,23,1)
hour <- seq(0,23.9,0.1)

hour_harmonics_df <- data.frame("linear_term" = rep(1, length(hour)),
                                "hour_s1" = sin(2*pi*hour/24),
                                "hour_c1" = cos(2*pi*hour/24))

harmonics_scaled_df_1p <- data.frame(
```

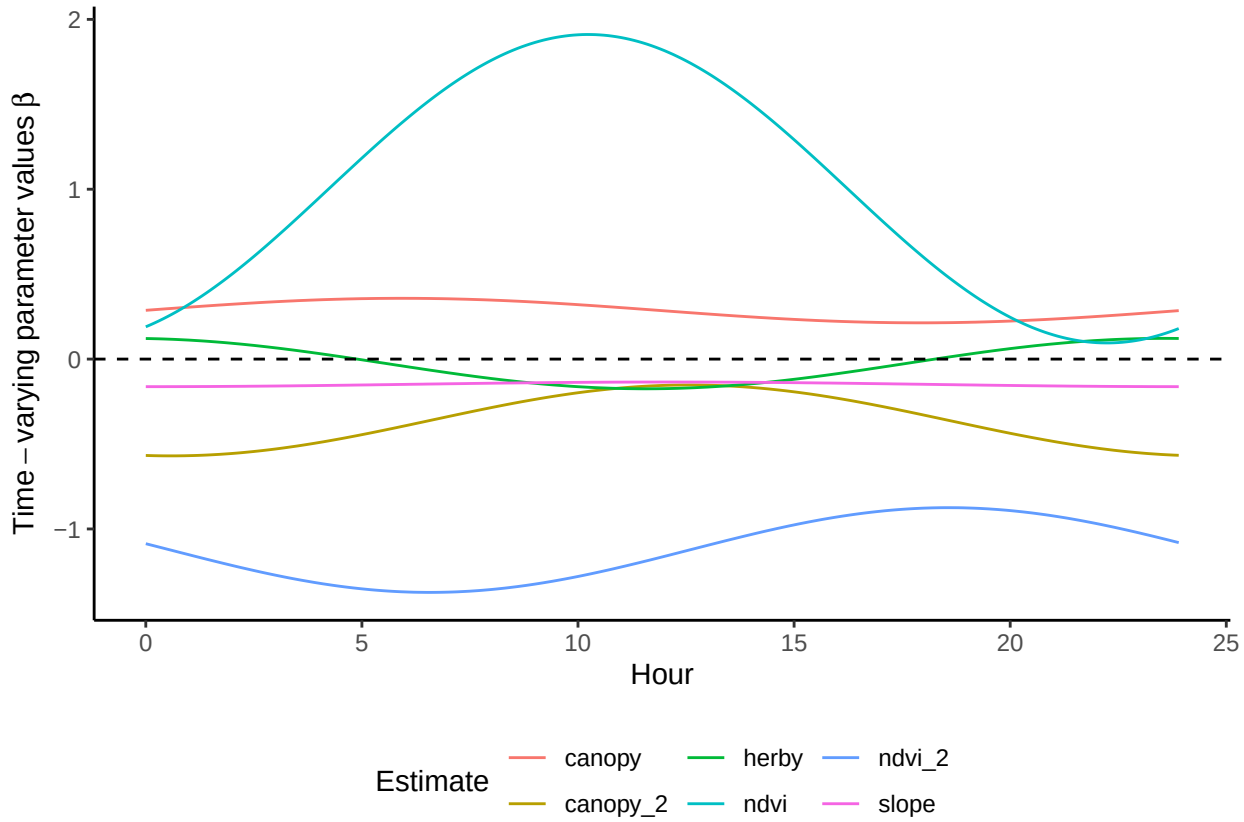
```
  "hour" = hour,
  "ndvi" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "ndvi_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "slope" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("slope", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "herby" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("herby", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("step_l", coefs) & !grepl("log", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "log_sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("log_step_l", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "cos_ta" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("cos", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))))

harmonics_scaled_long_1p <- pivot_longer(harmonics_scaled_df_1p,
                                         cols = !1,
                                         names_to = "coef")
```

Temporally dynamic scaled external selection parameters

```
ggplot() +
    geom_path(data = harmonics_scaled_long_1p,
              aes(x = hour, y = value, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
    scale_x_continuous("Hour") +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```




```
ggplot() +
    geom_path(data = harmonics_scaled_long_1p %>%
              filter(!coef %in% c("sl", "log_sl", "cos_ta", "memory", "memory_2")),
              aes(x = hour, y = value, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
    scale_x_continuous("Hour") +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



For the natural scale parameters

```
harmonics_nat_df_1p <- data.frame(
  "hour" = hour,
  "ndvi" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "ndvi_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "slope" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("slope", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "herby" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("herby", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory", coefs)& !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory_sq", coefs)) %>%
```

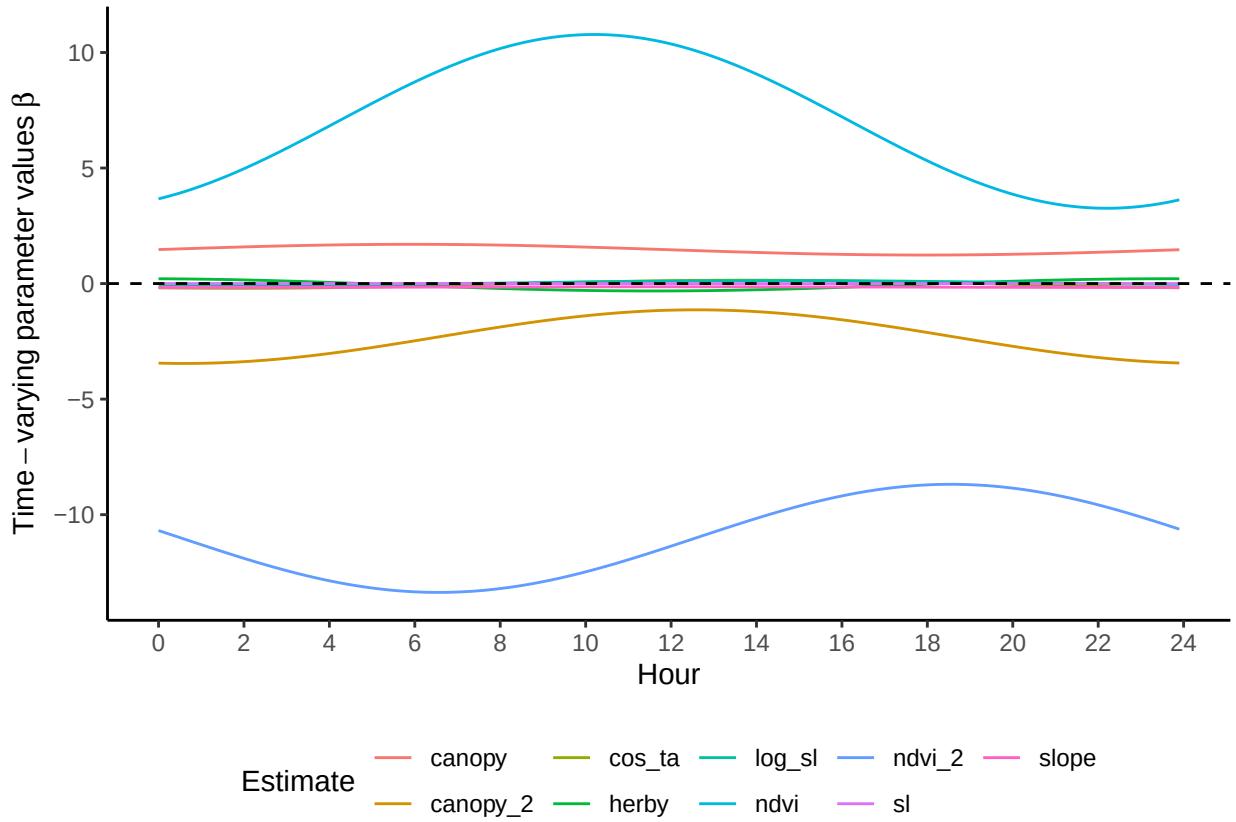
```
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("step_l", coefs) & !grepl("log", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "log_sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("log_step_l", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "cos_ta" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("cos", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))))
```

Reconstructing the Gamma and von Mises distributions from the tentative distributions (from Fieberg et al 2021: Appendix C)

```
tentative_shape <- 0.438167
tentative_scale <- 534.3507
tentative_kappa <- 0.1848126

hour_coefs_nat_df_1p <- harmonics_nat_df_1p %>%
  mutate(shape = tentative_shape + log_sl,
         scale = 1/((1/tentative_scale) - sl),
         kappa = tentative_kappa + cos_ta)

# save the coefficients to use in the simulations
# write_csv(hour_coefs_nat_df_1p,
#           paste0("outputs/TwoStep_1pDaily_coefs_dry_",Sys.Date(), ".csv"))

# turning into a long data frame
hour_coefs_nat_long_1p <- pivot_longer(hour_coefs_nat_df_1p,
                                       cols = !1, names_to = "coef")
```

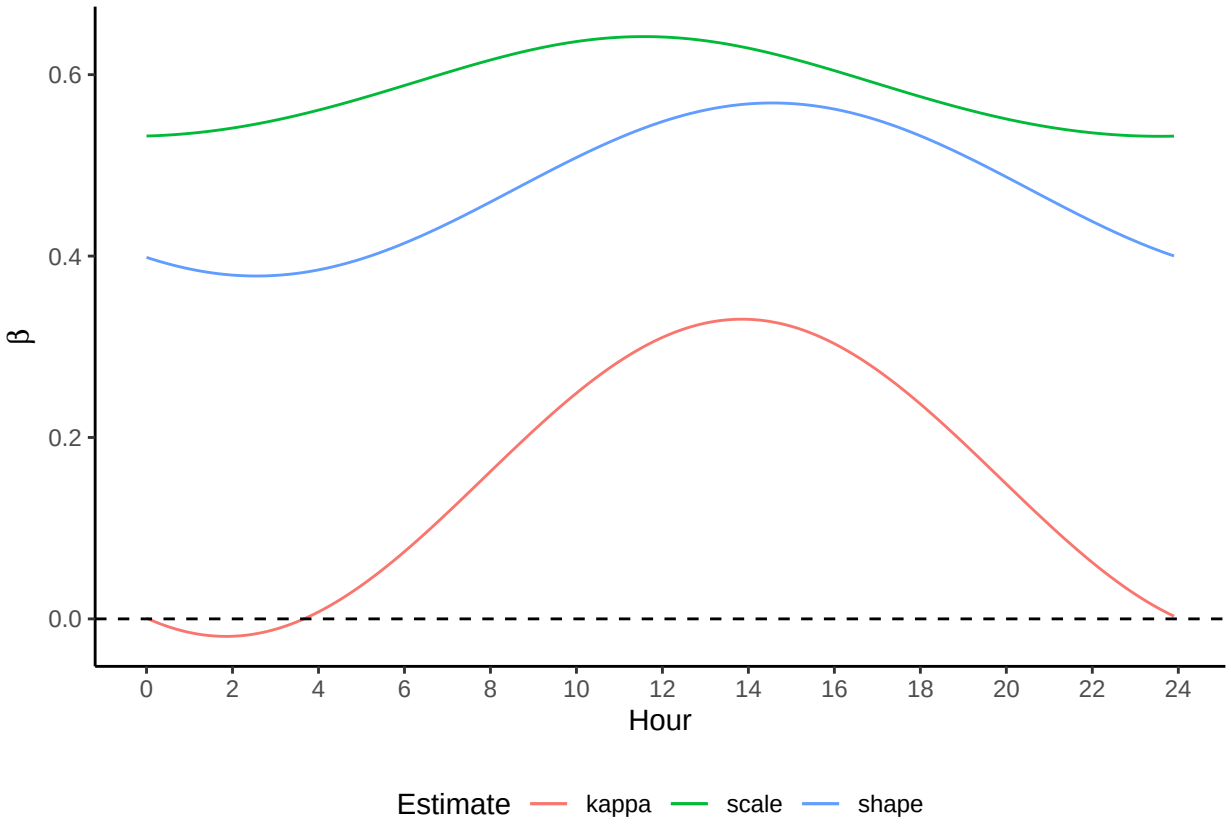
Temporally dynamic natural-scale external selection parameters

```
ggplot() +
  geom_path(data = hour_coefs_nat_long_1p %>%
              filter(!coef %in% c("shape", "scale", "kappa", "memory", "memory_2")),
            aes(x = hour, y = value, colour = coef)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_color_discrete("Estimate") +
  theme_classic() +
  theme(legend.position = "bottom")
```



Temporally dynamic natural-scale movement parameters

```
ggplot() +
    geom_path(data = hour_coefs_nat_long_1p %>%
                filter(coef %in% c("shape", "kappa")),
                aes(x = hour, y = value, colour = coef)) +
  geom_path(data = hour_coefs_nat_long_1p %>%
                filter(coef == "scale"),
                aes(x = hour, y = value/1000, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(beta)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



Sampling from movement kernel

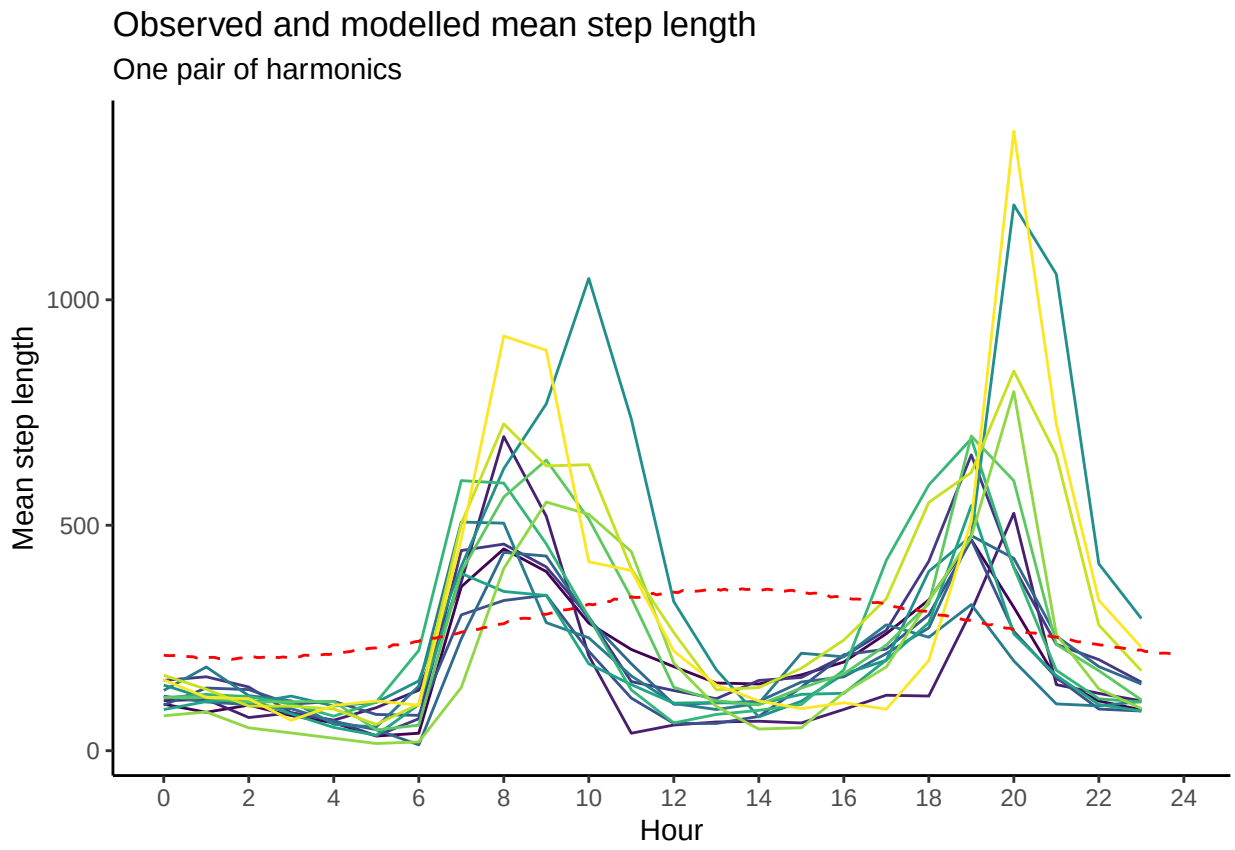
Here we sample from the movement kernel to generate a distribution of step lengths for each hour of the day, to assess how well it matches the observed step lengths.

```
movement_summary <- buffalo_data %>% filter(y == 1) %>%
  group_by(id, hour) %>%
  summarise(mean_sl = mean(sl), median_sl = median(sl))
```

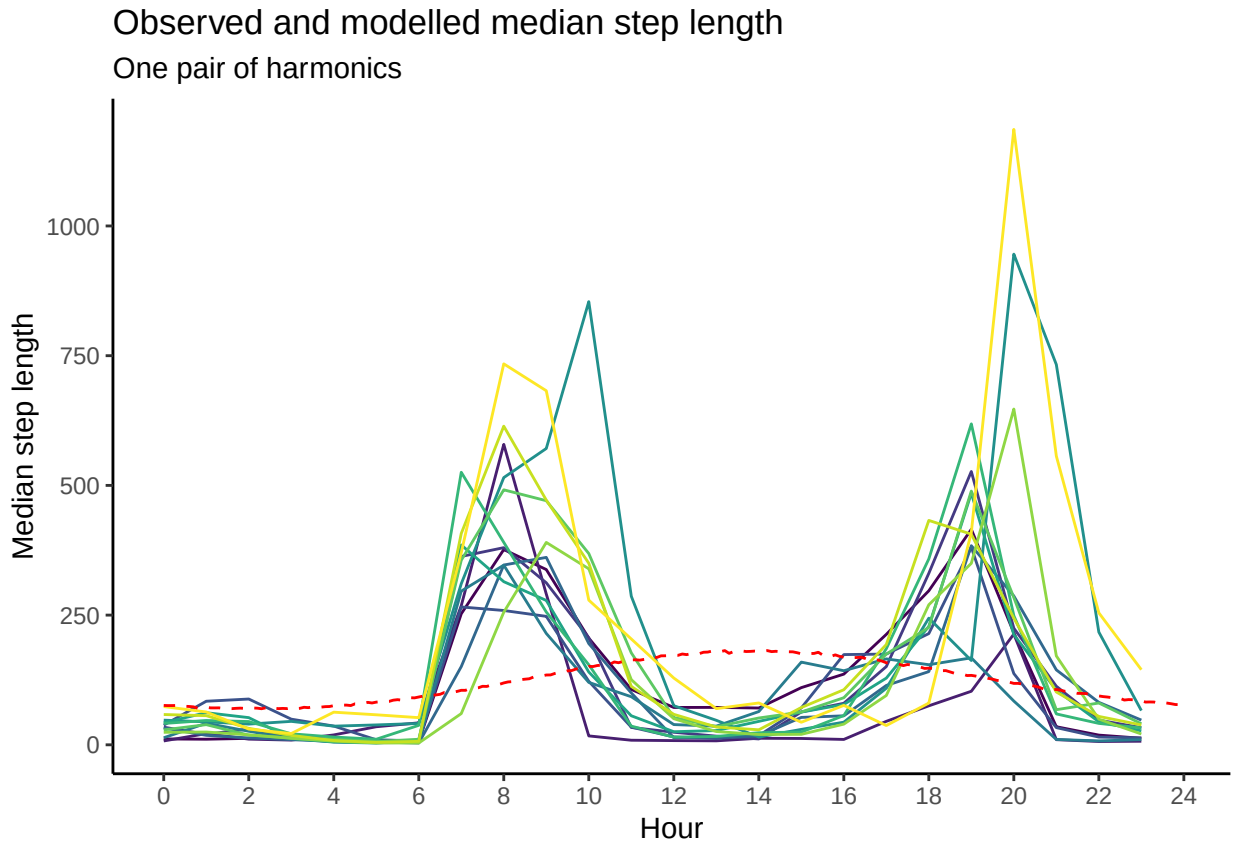
```
## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.
```

```
hour_no <- 1
n <- 1e5

gamma_dist_list <- vector(mode = "list", length = nrow(hour_coefs_nat_df_1p))
gamma_mean <- c()
gamma_median <- c()
gamma_ratio <- c()

for(hour_no in 1:nrow(hour_coefs_nat_df_1p)) {

  gamma_dist_list[[hour_no]] <- rgamma(n,
                                       shape = hour_coefs_nat_df_1p$shape[hour_no],
                                       scale = hour_coefs_nat_df_1p$scale[hour_no])

  gamma_mean[hour_no] <- mean(gamma_dist_list[[hour_no]])
  gamma_median[hour_no] <- median(gamma_dist_list[[hour_no]])
```

```
  gamma_ratio[hour_no] <- gamma_mean[hour_no] / gamma_median[hour_no]

}

gamma_df_1p <- data.frame(model = "1p",
                          hour = hour_coefs_nat_df_1p$hour,
                          mean = gamma_mean,
                          median = gamma_median,
                          ratio = gamma_ratio)

mean_sl_1p <- ggplot() +
  geom_path(data = movement_summary,
            aes(x = hour, y = mean_sl, colour = factor(id))) +
  geom_path(data = gamma_df_1p,
            aes(x = hour, y = mean),
            colour = "red", linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Mean step length") +
  scale_colour_viridis_d("Buffalo") +
  ggtitle("Observed and modelled mean step length",
          subtitle = "One pair of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

mean_sl_1p
```



```
median_sl_1p <- ggplot() +
  geom_path(data = movement_summary,
            aes(x = hour, y = median_sl, colour = factor(id))) +
  geom_path(data = gamma_df_1p,
            aes(x = hour, y = median),
            colour = "red", linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Median step length") +
  scale_colour_viridis_d("Buffalo") +
  ggtitle("Observed and modelled median step length",
          subtitle = "One pair of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

median_sl_1p
```




```
# across the hours
buffalo_data_all %>% filter(y == 1) %>%  group_by(id) %>%
  summarise(mean_sl = mean(sl),
            median_sl = median(sl),
            ratio = mean_sl/median_sl)
```

```
## # A tibble: 13 x 4
##       id mean_sl median_sl ratio
##    <dbl>   <dbl>     <dbl> <dbl>
##  1  2005    204.      98.0  2.08
```

```
##  2  2014    142.      15.9  8.94
##  3  2018    249.     103.   2.41
##  4  2021    183.      94.8  1.93
##  5  2022    216.      78.4  2.75
##  6  2024    229.      77.9  2.94
##  7  2039    357.     124.   2.87
##  8  2154    198.      95.6  2.07
##  9  2158    219.      84.8  2.58
## 10  2223    249.      80.2  3.10
## 11  2327    200.      51.3  3.91
## 12  2387    327.     111.   2.95
## 13  2393    322.     127.   2.53
```

```
buffalo_data_all %>% filter(y == 1) %>%
  summarise(mean_sl = mean(sl),
            median_sl = median(sl),
            ratio = mean_sl/median_sl)
```

```
## # A tibble: 1 x 3
##   mean_sl median_sl ratio
##     <dbl>     <dbl> <dbl>
## 1    238.      86.9  2.74
```

```
gamma_df_1p |> summarise(mean_mean = mean(mean),
                         median_mean = mean(median),
                         ratio_mean = mean_mean/median_mean)
```

```
##   mean_mean median_mean ratio_mean
## 1    278.39    122.1832   2.278463
```

Creating selection surfaces

As we have both quadratic and harmonic terms in the model, we can reconstruct a 'selection surface' to visualise how the coefficient changes through time.

To illustrate, we have a coefficient for the linear term and a coefficient for the quadratic term, for every hour of the day (or for every time point that we used to reconstruct the temporal dynamics (0.1 hour in this case). Using these coefficients, we can plot the selection curve at the scale of the environmental variable (in this case NDVI).

Using the natural scale coefficients from the model, let's take the coefficient from hour 0 and plot the quadratic curve

```
# first get a sequence of NDVI values,
# starting from the minimum observed in the data to the maximum
ndvi_min <- min(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_max <- max(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_seq <- seq(ndvi_min, ndvi_max, by = 0.01)
```

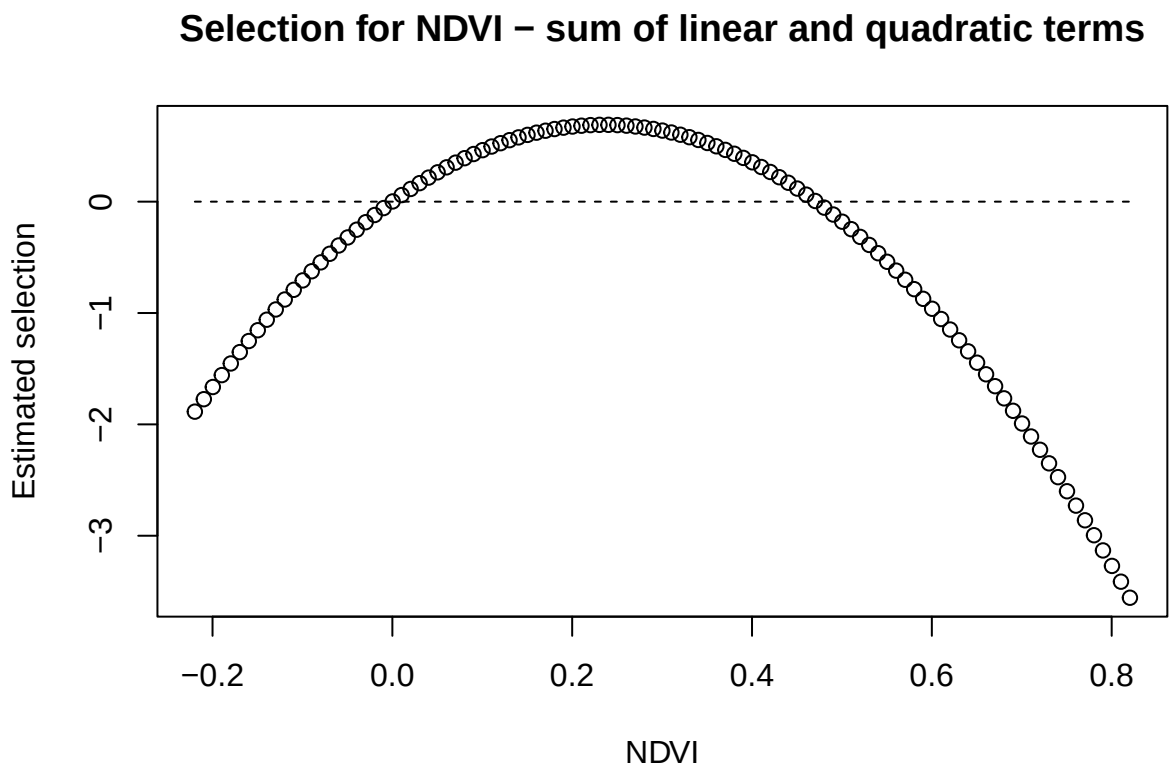
Take the coefficients from the model and calculation the selection value for every NDVI value in this sequence.

For brevity we won't plot the linear and quadratic terms separetely, but we can do so if needed.

First for **Hour 3**

```
hour_no <- 3

# we can separate to the linear term
ndvi_linear_selection <-
```

```
  hour_coefs_nat_df_1p$ndvi[which(hour_coefs_nat_df_1p$hour == hour_no)] * ndvi_seq
# plot(x = ndvi_seq, y = ndvi_linear_selection,
#      main = "Selection for NDVI - linear term",
#      xlab = "NDVI", ylab = "Estimated selection")

# and the quadratic term
ndvi_quadratic_selection <-
  (hour_coefs_nat_df_1p$ndvi_2[which(hour_coefs_nat_df_1p$hour == hour_no)] * (ndvi_seq ^ 2))
# plot(x = ndvi_seq, y = ndvi_quadratic_selection,
#      main = "Selection for NDVI - quadratic term",
#      xlab = "NDVI", ylab = "Estimated selection")

# and the sum of both
ndvi_sum_selection <- ndvi_linear_selection + ndvi_quadratic_selection
plot(x = ndvi_seq, y = ndvi_sum_selection,
     main = "Selection for NDVI - sum of linear and quadratic terms",
     xlab = "NDVI", ylab = "Estimated selection")
lines(ndvi_seq, rep(0,length(ndvi_seq)), lty = "dashed")
```

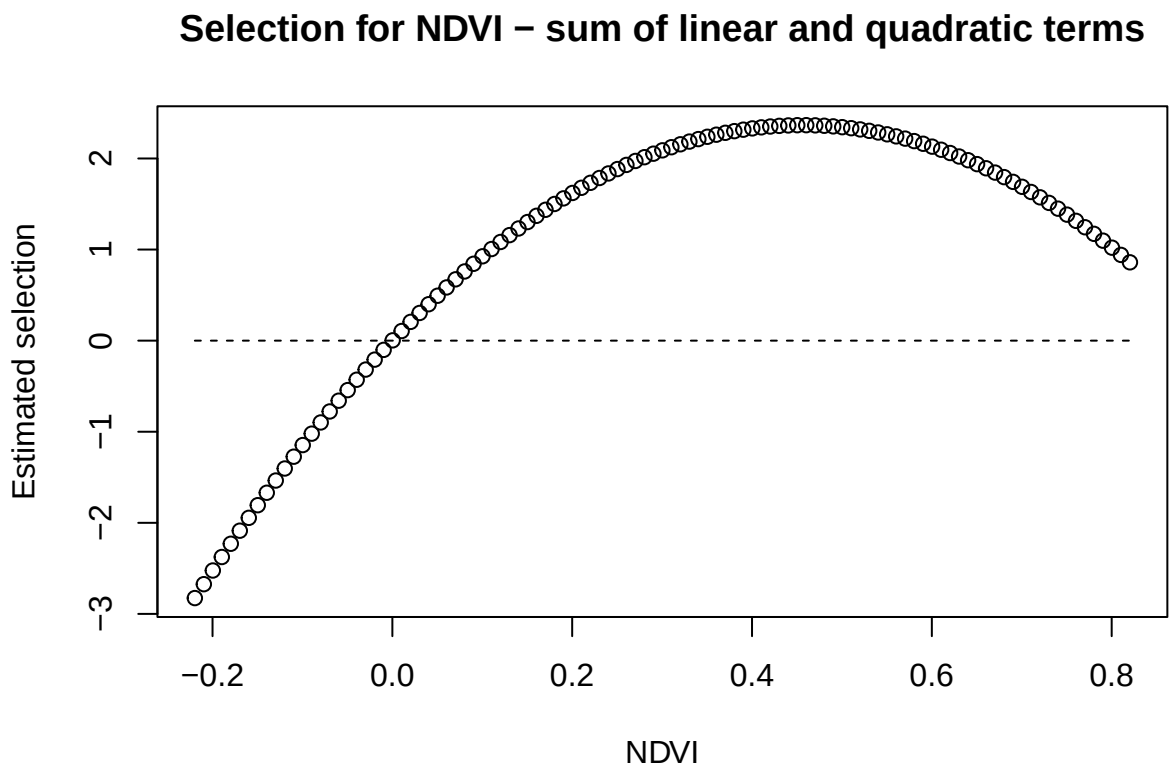


We can see that the coefficient at hour 3 shows highest selection for NDVI values slightly above 0.2, and the coefficient is mostly negative.

Secondly for **Hour 12**

```
hour_no <- 12

# we can separate to the linear term
```

```r
ndvi_linear_selection <-
  hour_coefs_nat_df_1p$ndvi[which(hour_coefs_nat_df_1p$hour == hour_no)] * ndvi_seq
# plot(x = ndvi_seq, y = ndvi_linear_selection,
#      main = "Selection for NDVI - linear term",
#      xlab = "NDVI", ylab = "Estimated selection")

# and the quadratic term
ndvi_quadratic_selection <-
  (hour_coefs_nat_df_1p$ndvi_2[which(hour_coefs_nat_df_1p$hour == hour_no)] * (ndvi_seq ^ 2))
# plot(x = ndvi_seq, y = ndvi_quadratic_selection,
#      main = "Selection for NDVI - quadratic term",
#      xlab = "NDVI", ylab = "Estimated selection")

# and the sum of both
ndvi_sum_selection <- ndvi_linear_selection + ndvi_quadratic_selection
plot(x = ndvi_seq, y = ndvi_sum_selection,
     main = "Selection for NDVI - sum of linear and quadratic terms",
     xlab = "NDVI", ylab = "Estimated selection")
lines(ndvi_seq, rep(0,length(ndvi_seq)), lty = "dashed")
```



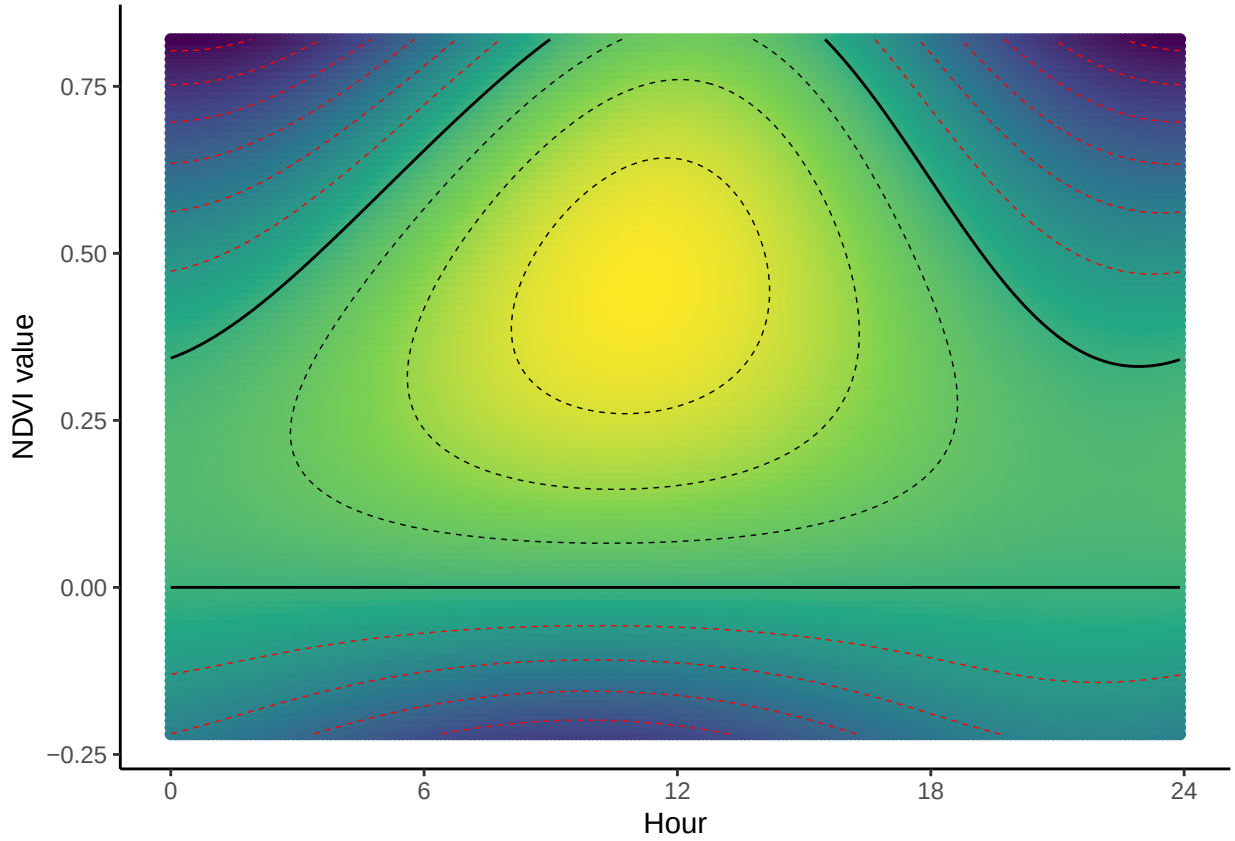
Whereas for hour 12, the coefficient shows highest selection for NDVI values slightly above 0.4, and the coefficient is positive for NDVI values above 0.

We can imagine viewing these plots for every hour of the day, where each hour has a different quadratic curve, but this would be a lot of plots. We can also see it as a 3D surface, where the x-axis is the hour of the day, the y-axis is the NDVI value, and the z-axis (colour) is the coefficient value.

As with the static model, we index over the linear and quadratic terms and calculate the coefficient values at every time point.

Selection surface for NDVI

```
ndvi_min <- min(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_max <- max(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_seq <- seq(ndvi_min, ndvi_max, by = 0.01)

# Create empty data frame
ndvi_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_1p),
                                       nrow = length(ndvi_seq)))
for(i in 1:nrow(hour_coefs_nat_df_1p)) {
  # Assign the vector as a column to the dataframe
  ndvi_fresponse_df[,i] <- (hour_coefs_nat_df_1p$ndvi[i] * ndvi_seq) +
    (hour_coefs_nat_df_1p$ndvi_2[i] * (ndvi_seq ^ 2))
}

ndvi_fresponse_df <- data.frame(ndvi_seq, ndvi_fresponse_df)
colnames(ndvi_fresponse_df) <- c("ndvi", hour)
ndvi_fresponse_long <- pivot_longer(ndvi_fresponse_df, cols = !1, names_to = "hour")

ndvi_contour_max <- max(ndvi_fresponse_long$value) # 0.7890195
ndvi_contour_min <- min(ndvi_fresponse_long$value) # -0.7945691
ndvi_contour_increment <- (ndvi_contour_max-ndvi_contour_min)/10

ndvi_quad_1p <- ggplot(data = ndvi_fresponse_long,
                       aes(x = as.numeric(hour), y = ndvi)) +
  geom_point(aes(colour = value)) + # colour = "white"
  geom_contour(aes(z = value),
               breaks = seq(ndvi_contour_increment,
                            ndvi_contour_max,
                            ndvi_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-ndvi_contour_increment,
                            ndvi_contour_min,
                            -ndvi_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("NDVI value", breaks = seq(-1, 1, 0.25)) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Normalised Difference Vegetation Index (NDVI)") +
  theme_classic() +
  theme(legend.position = "none")

ndvi_quad_1p
```

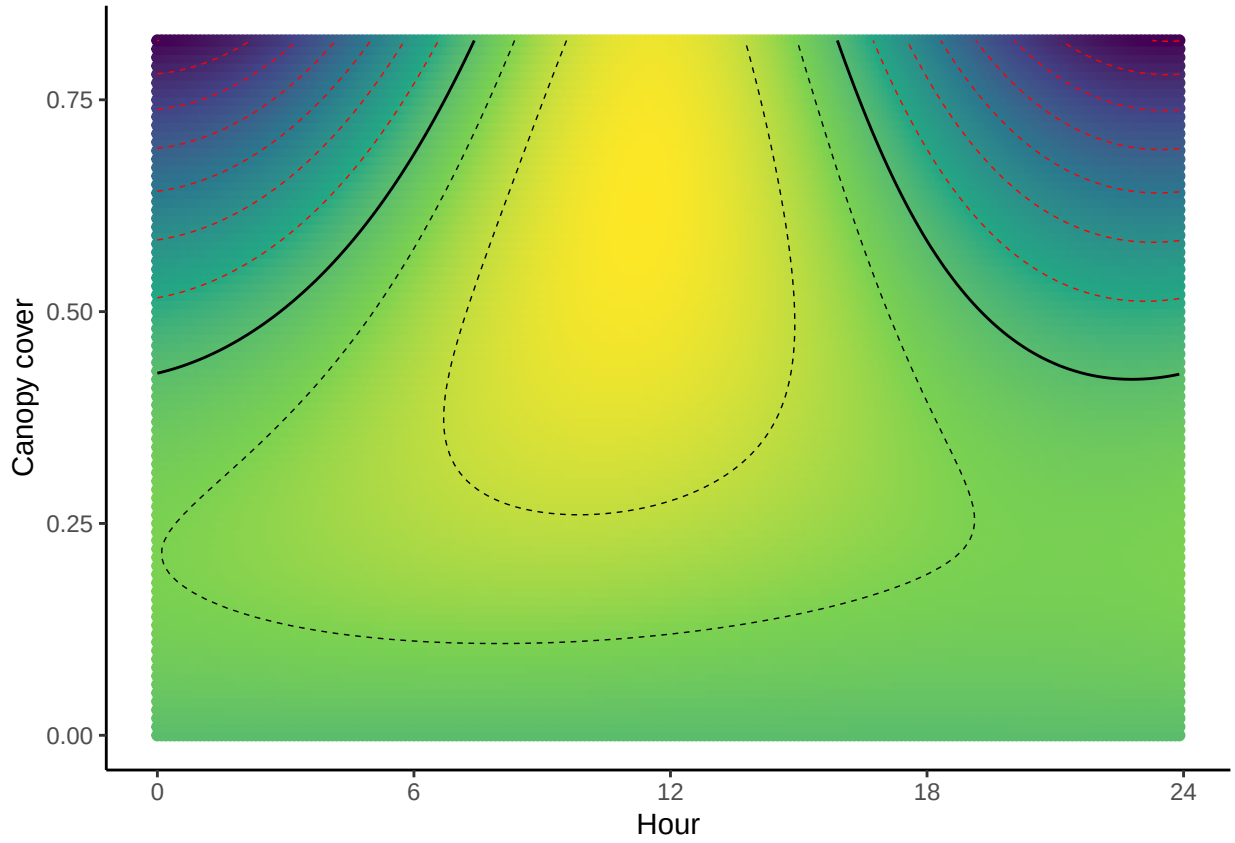


Canopy cover

```
canopy_min <- min(buffalo_data$canopy_01, na.rm = TRUE)
canopy_max <- max(buffalo_data$canopy_01, na.rm = TRUE)
canopy_seq <- seq(canopy_min, canopy_max, by = 0.01)

# Create empty data frame
canopy_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_1p),
                                         nrow = length(canopy_seq)))
for(i in 1:nrow(hour_coefs_nat_df_1p)) {
  # Assign the vector as a column to the dataframe
  canopy_fresponse_df[,i] <- (hour_coefs_nat_df_1p$canopy[i] * canopy_seq) +
    (hour_coefs_nat_df_1p$canopy_2[i] * (canopy_seq ^ 2))
}

canopy_fresponse_df <- data.frame(canopy_seq, canopy_fresponse_df)
colnames(canopy_fresponse_df) <- c("canopy", hour)
canopy_fresponse_long <- pivot_longer(canopy_fresponse_df, cols = !1,
                                      names_to = "hour")

canopy_contour_min <- min(canopy_fresponse_long$value) # 0
canopy_contour_max <- max(canopy_fresponse_long$value) # 2.181749
canopy_contour_increment <- (canopy_contour_max-canopy_contour_min)/10

canopy_quad_1p <- ggplot(data = canopy_fresponse_long,
```

```
                        aes(x = as.numeric(hour), y = canopy)) +
  geom_point(aes(colour = value)) +
  geom_contour(aes(z = value),
               breaks = seq(canopy_contour_increment,
                            canopy_contour_max,
                            canopy_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-canopy_contour_increment,
                            canopy_contour_min,
                            -canopy_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("Canopy cover", breaks = seq(0, 1, 0.25)) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Canopy Cover") +
  theme_classic() +
  theme(legend.position = "none")
```

```
canopy_quad_1p
```

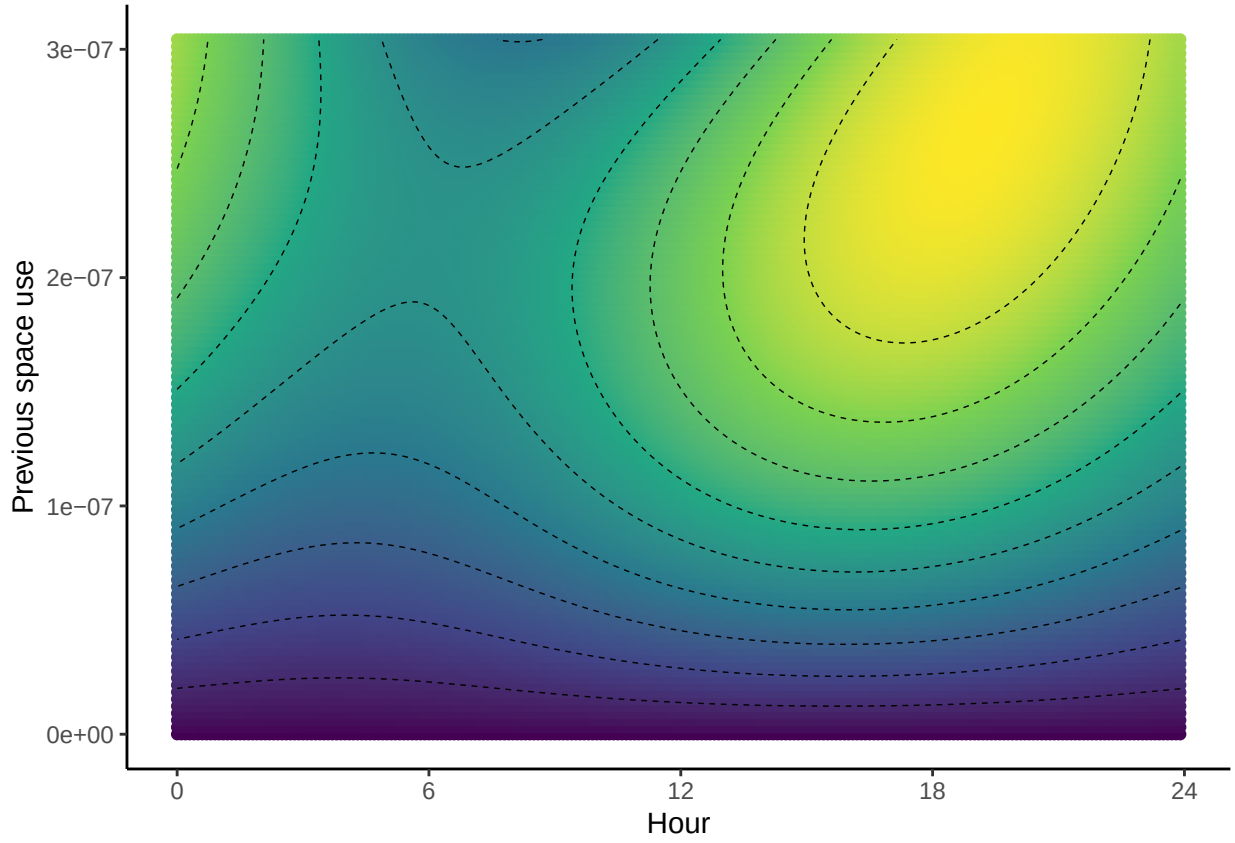


Previous space use density

```r
memory_min <- min(buffalo_data$kde_memory_density, na.rm = TRUE)
# memory_max <- max(buffalo_data$kde_memory_density, na.rm = TRUE)
memory_max <- quantile(buffalo_data |> filter(y == 1) |>
                         pull(kde_memory_density), probs = 0.95, na.rm = TRUE)
memory_seq <- seq(memory_min, memory_max, length.out = 100)

# Create empty data frame
memory_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_1p),
                                         nrow = length(memory_seq)))
for(i in 1:nrow(hour_coefs_nat_df_1p)) {
  # Assign the vector as a column to the dataframe
  memory_fresponse_df[,i] <- (hour_coefs_nat_df_1p$memory[i] * memory_seq) +
    (hour_coefs_nat_df_1p$memory_2[i] * (memory_seq ^ 2))
}

memory_fresponse_df <- data.frame(memory_seq, memory_fresponse_df)
colnames(memory_fresponse_df) <- c("memory", hour)
memory_fresponse_long <- pivot_longer(memory_fresponse_df, cols = !1,
                                      names_to = "hour")

memory_contour_min <- min(memory_fresponse_long$value) # 0
memory_contour_max <- max(memory_fresponse_long$value) # 2.181749
memory_contour_increment <- (memory_contour_max-memory_contour_min)/10

memory_quad_1p <- ggplot(data = memory_fresponse_long,
                         aes(x = as.numeric(hour), y = memory)) +
  geom_point(aes(colour = value)) +
  geom_contour(aes(z = value),
               breaks = seq(memory_contour_increment,
                            memory_contour_max,
                            memory_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-memory_contour_increment,
                            min(-memory_contour_increment, memory_contour_min),
                            -memory_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("Previous space use", labels = scientific) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Relationship to previous space use") +
  theme_classic() +
  theme(legend.position = "none")

memory_quad_1p
```

```
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x): no non-missing arguments to max; returning -Inf
```



Two pairs of harmonics

```
months_wet <- c(1:4, 11:12)
buffalo_ids <- unique(buffalo_data_all$id)

# buffalo_data <- buffalo_data_all %>% filter(month %in% months_wet) # wet season
buffalo_data <- buffalo_data_all %>% filter(!month %in% months_wet) # dry season

buffalo_data_matrix_unscaled <- buffalo_data %>% transmute(

  ndvi = ndvi_temporal,
  ndvi_s1 = ndvi_temporal * hour_s1,
  ndvi_s2 = ndvi_temporal * hour_s2,
  ndvi_c1 = ndvi_temporal * hour_c1,
  ndvi_c2 = ndvi_temporal * hour_c2,

  ndvi_sq = ndvi_temporal ^ 2,
  ndvi_sq_s1 = (ndvi_temporal ^ 2) * hour_s1,
  ndvi_sq_s2 = (ndvi_temporal ^ 2) * hour_s2,
  ndvi_sq_c1 = (ndvi_temporal ^ 2) * hour_c1,
  ndvi_sq_c2 = (ndvi_temporal ^ 2) * hour_c2,

  canopy = canopy_01,
```

```
canopy_s1 = canopy_01 * hour_s1,
canopy_s2 = canopy_01 * hour_s2,
canopy_c1 = canopy_01 * hour_c1,
canopy_c2 = canopy_01 * hour_c2,

canopy_sq = canopy_01 ^ 2,
canopy_sq_s1 = (canopy_01 ^ 2) * hour_s1,
canopy_sq_s2 = (canopy_01 ^ 2) * hour_s2,
canopy_sq_c1 = (canopy_01 ^ 2) * hour_c1,
canopy_sq_c2 = (canopy_01 ^ 2) * hour_c2,

slope = slope,
slope_s1 = slope * hour_s1,
slope_s2 = slope * hour_s2,
slope_c1 = slope * hour_c1,
slope_c2 = slope * hour_c2,

herby = veg_herby,
herby_s1 = veg_herby * hour_s1,
herby_s2 = veg_herby * hour_s2,
herby_c1 = veg_herby * hour_c1,
herby_c2 = veg_herby * hour_c2,

spatial_memory = kde_memory_density,
spatial_memory_s1 = kde_memory_density * hour_s1,
spatial_memory_s2 = kde_memory_density * hour_s2,
spatial_memory_c1 = kde_memory_density * hour_c1,
spatial_memory_c2 = kde_memory_density * hour_c2,

spatial_memory_sq = kde_memory_density ^ 2,
spatial_memory_sq_s1 = (kde_memory_density ^ 2) * hour_s1,
spatial_memory_sq_s2 = (kde_memory_density ^ 2) * hour_s2,
spatial_memory_sq_c1 = (kde_memory_density ^ 2) * hour_c1,
spatial_memory_sq_c2 = (kde_memory_density ^ 2) * hour_c2,

step_l = sl,
step_l_s1 = sl * hour_s1,
step_l_s2 = sl * hour_s2,
step_l_c1 = sl * hour_c1,
step_l_c2 = sl * hour_c2,

log_step_l = log_sl,
log_step_l_s1 = log_sl * hour_s1,
log_step_l_s2 = log_sl * hour_s2,
log_step_l_c1 = log_sl * hour_c1,
log_step_l_c2 = log_sl * hour_c2,

cos_turn_a = cos_ta,
cos_turn_a_s1 = cos_ta * hour_s1,
cos_turn_a_s2 = cos_ta * hour_s2,
cos_turn_a_c1 = cos_ta * hour_c1,
cos_turn_a_c2 = cos_ta * hour_c2)
```

```r
buffalo_data_matrix_scaled <- scale(buffalo_data_matrix_unscaled)

mean_vals <- attr(buffalo_data_matrix_scaled, "scaled:center")
sd_vals <- attr(buffalo_data_matrix_scaled, "scaled:scale")
scaling_attributes <- data.frame(variable = names(buffalo_data_matrix_unscaled),
                                 mean = mean_vals, sd = sd_vals)

buffalo_data_scaled_2p <- data.frame(id = buffalo_data$id,
                                     step_id = buffalo_data$step_id,
                                     y = buffalo_data$y,
                                     buffalo_data_matrix_scaled)
```

Formula with two pairs of harmonics

```r
formula_twostep <- y ~

  ndvi +
  ndvi_s1 +
  ndvi_s2 +
  ndvi_c1 +
  ndvi_c2 +

  ndvi_sq +
  ndvi_sq_s1 +
  ndvi_sq_s2 +
  ndvi_sq_c1 +
  ndvi_sq_c2 +

  canopy +
  canopy_s1 +
  canopy_s2 +
  canopy_c1 +
  canopy_c2 +

  canopy_sq +
  canopy_sq_s1 +
  canopy_sq_s2 +
  canopy_sq_c1 +
  canopy_sq_c2 +

  slope +
  slope_s1 +
  slope_s2 +
  slope_c1 +
  slope_c2 +

  herby +
  herby_s1 +
  herby_s2 +
  herby_c1 +
  herby_c2 +

  spatial_memory +
  spatial_memory_s1 +
```

```
  spatial_memory_s2 +
  spatial_memory_c1 +
  spatial_memory_c2 +

  spatial_memory_sq +
  spatial_memory_sq_s1 +
  spatial_memory_sq_s2 +
  spatial_memory_sq_c1 +
  spatial_memory_sq_c2 +

  step_l +
  step_l_s1 +
  step_l_s2 +
  step_l_c1 +
  step_l_c2 +

  log_step_l +
  log_step_l_s1 +
  log_step_l_s2 +
  log_step_l_c1 +
  log_step_l_c2 +

  cos_turn_a +
  cos_turn_a_s1 +
  cos_turn_a_s2 +
  cos_turn_a_c1 +
  cos_turn_a_c2 +

  strata(step_id) +
  cluster(id)
```

Fitting the model

As we have already fitted the model, we will load it here, but if the model_fit file doesn't exist, it will run the model fitting code. Be careful here that if you change the model formula, you will need to delete or rename the model_fit file to re-run the model fitting code, otherwise it will just load the previous model.

```
if(file.exists("outputs/model_twostep_2p_harms_dry.rds")) {

  model_twostep_2p_harms <- readRDS("outputs/model_twostep_2p_harms_dry.rds")

} else {

  tic()
  model_twostep_2p_harms <- Ts.estim(formula = formula_twostep,
            data = buffalo_data_scaled_2p,
            all.m.1 = TRUE,
            D = "UN(1)",
            itermax = 10000)
  toc()

  # save model object
  saveRDS(model_twostep_2p_harms, file = "outputs/model_twostep_2p_harms_dry.rds")
```

```
  beep(sound = 2)

}
```

Read saved model object

```
# model_twostep_2p_harms

# model_twostep_2p_harms
# model_twostep_2p_harms$beta
# model_twostep_2p_harms$se
# model_twostep_2p_harms$vcov
# diag(model_twostep_2p_harms$D) # between cluster variance
# model_twostep_2p_harms$r.effect # individual estimates

# creating data frame of model coefficients
coefs_clr <- data.frame(coefs = names(model_twostep_2p_harms$beta),
                        value = model_twostep_2p_harms$beta)
coefs_clr$scale_sd <- scaling_attributes$sd
coefs_clr <- coefs_clr %>% mutate(value_nat = value / scale_sd)
head(coefs_clr)
```

```
##           coefs       value  scale_sd    value_nat
## ndvi       ndvi  1.18745765 0.1429054   8.30939941
## ndvi_s1 ndvi_s1  0.39442696 0.2371431   1.66324473
## ndvi_s2 ndvi_s2  0.01121123 0.2372765   0.04724962
## ndvi_c1 ndvi_c1 -0.95521148 0.2426054  -3.93730465
## ndvi_c2 ndvi_c2  1.25104461 0.2425239   5.15843931
## ndvi_sq ndvi_sq -1.33022674 0.1019530 -13.04744654
```

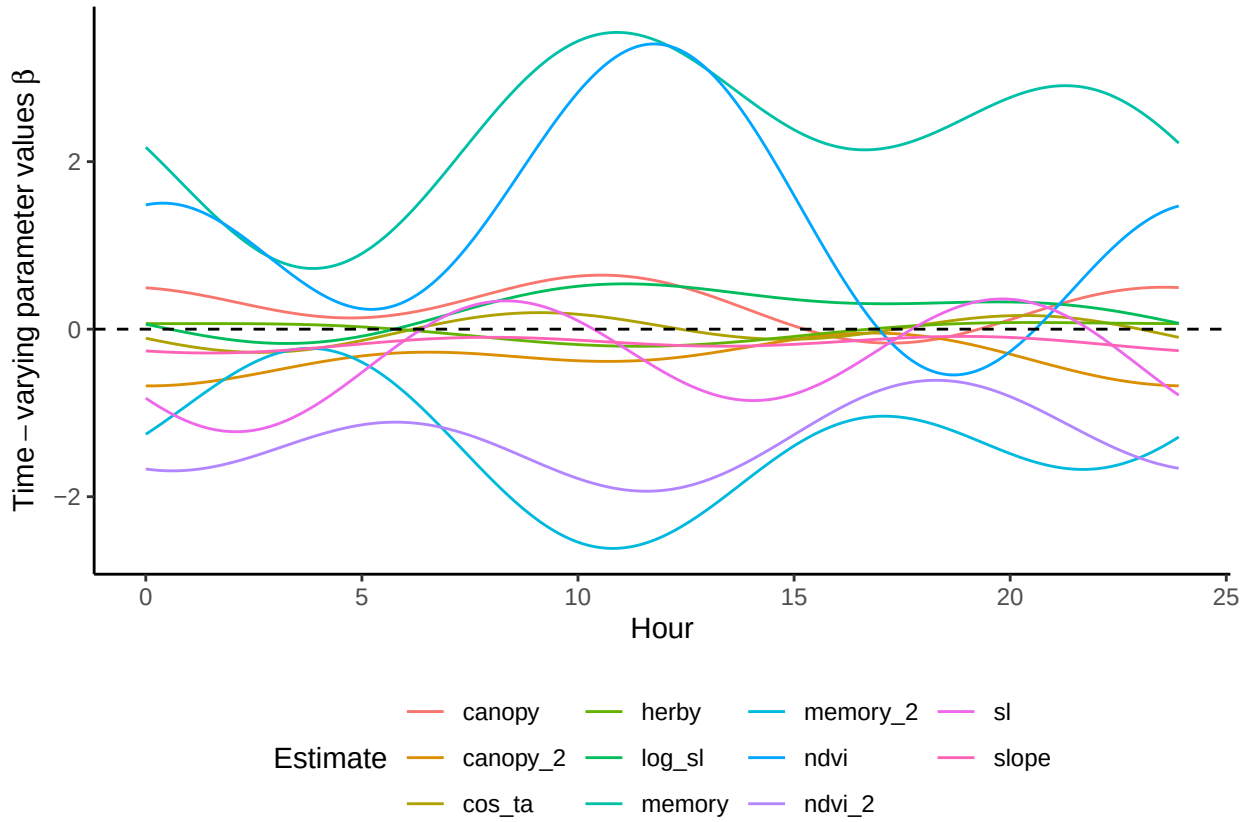
Reconstructing coefficients with two pairs of harmonics, with quadratic terms

```
# hour <- seq(0,23,1)
hour <- seq(0,23.9,0.1)

hour_harmonics_df <- data.frame("linear_term" = rep(1, length(hour)),
                                "hour_s1" = sin(2*pi*hour/24),
                                "hour_s2" = sin(4*pi*hour/24),
                                "hour_c1" = cos(2*pi*hour/24),
                                "hour_c2" = cos(4*pi*hour/24))

harmonics_scaled_df_2p <- data.frame(
  "hour" = hour,
  "ndvi" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "ndvi_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "slope" = as.numeric(
```

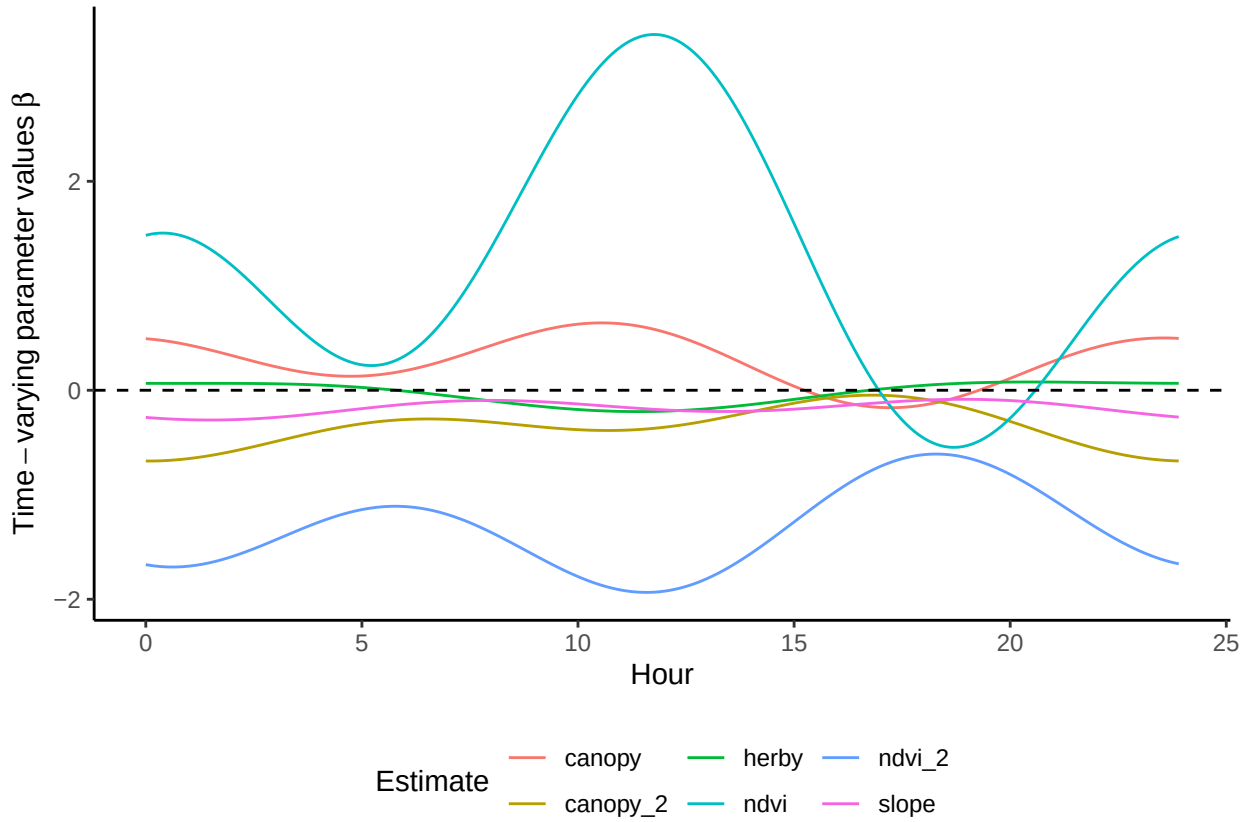
```
    coefs_clr %>% dplyr::filter(grepl("slope", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "herby" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("herby", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("step_l", coefs) & !grepl("log", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "log_sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("log_step_l", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "cos_ta" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("cos", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))))

harmonics_scaled_long_2p <- pivot_longer(harmonics_scaled_df_2p, cols = !1,
                                         names_to = "coef")
```

Temporally dynamic scaled external selection parameters

```
ggplot() +
    geom_path(data = harmonics_scaled_long_2p,
              aes(x = hour, y = value, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
    scale_x_continuous("Hour") +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```




```
ggplot() +
    geom_path(data = harmonics_scaled_long_2p %>%
              filter(!coef %in% c("sl", "log_sl", "cos_ta", "memory", "memory_2")),
              aes(x = hour, y = value, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
    scale_x_continuous("Hour") +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



For the natural scale parameters

```
harmonics_nat_df_2p <- data.frame(
  "hour" = hour,
  "ndvi" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "ndvi_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "slope" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("slope", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "herby" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("herby", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory", coefs)& !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory_sq", coefs)) %>%
```

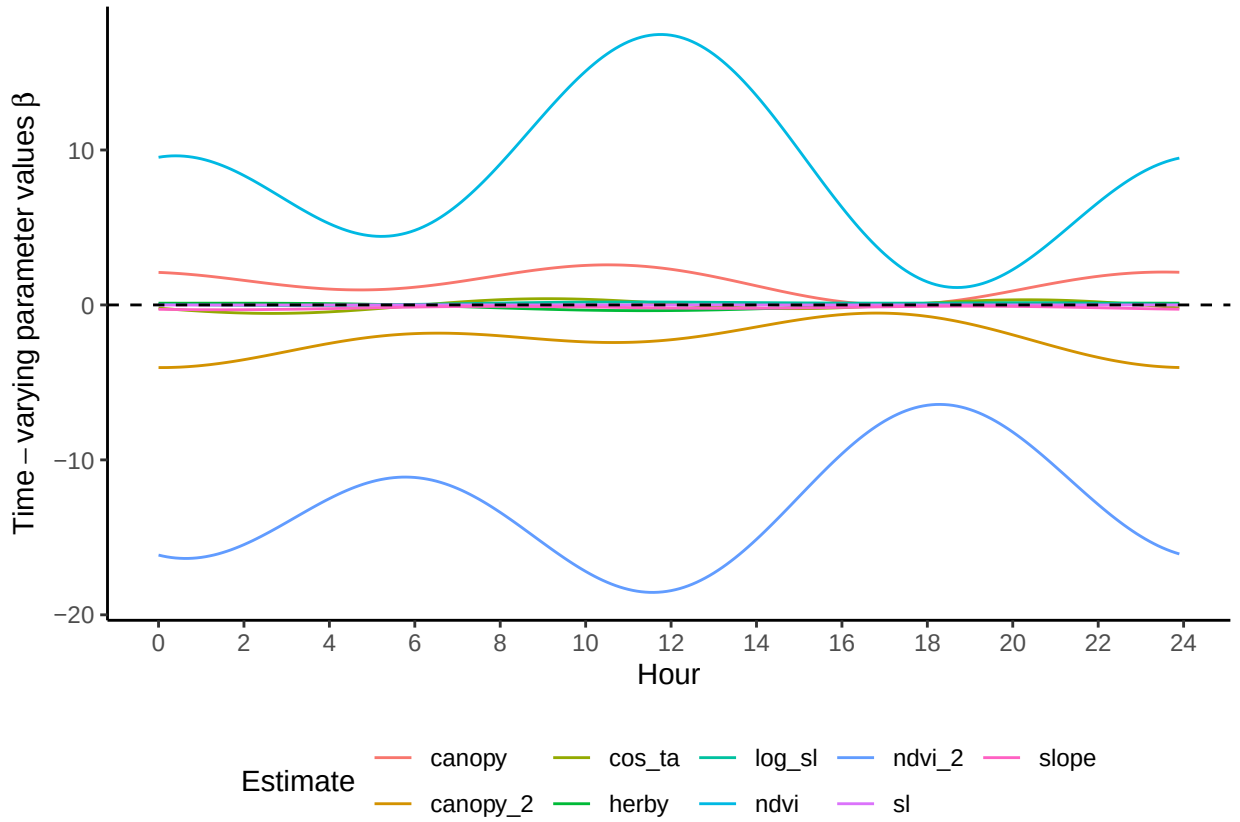
```
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("step_l", coefs) & !grepl("log", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "log_sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("log_step_l", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "cos_ta" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("cos", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))))
```

Reconstructing the Gamma and von Mises distributions from the tentative distributions (from Fieberg et al 2021: Appendix C)

```
tentative_shape <- 0.438167
tentative_scale <- 534.3507
tentative_kappa <- 0.1848126

hour_coefs_nat_df_2p <- harmonics_nat_df_2p %>%
  mutate(shape = tentative_shape + log_sl,
         scale = 1/((1/tentative_scale) - sl),
         kappa = tentative_kappa + cos_ta)

# save the coefficients to use in the simulations
# write_csv(hour_coefs_nat_df_2p,
#           paste0("outputs/TwoStep_2pDaily_coefs_dry_",Sys.Date(), ".csv"))

# turning into a long data frame
hour_coefs_nat_long_2p <- pivot_longer(hour_coefs_nat_df_2p, cols = !1,
                                       names_to = "coef")
```

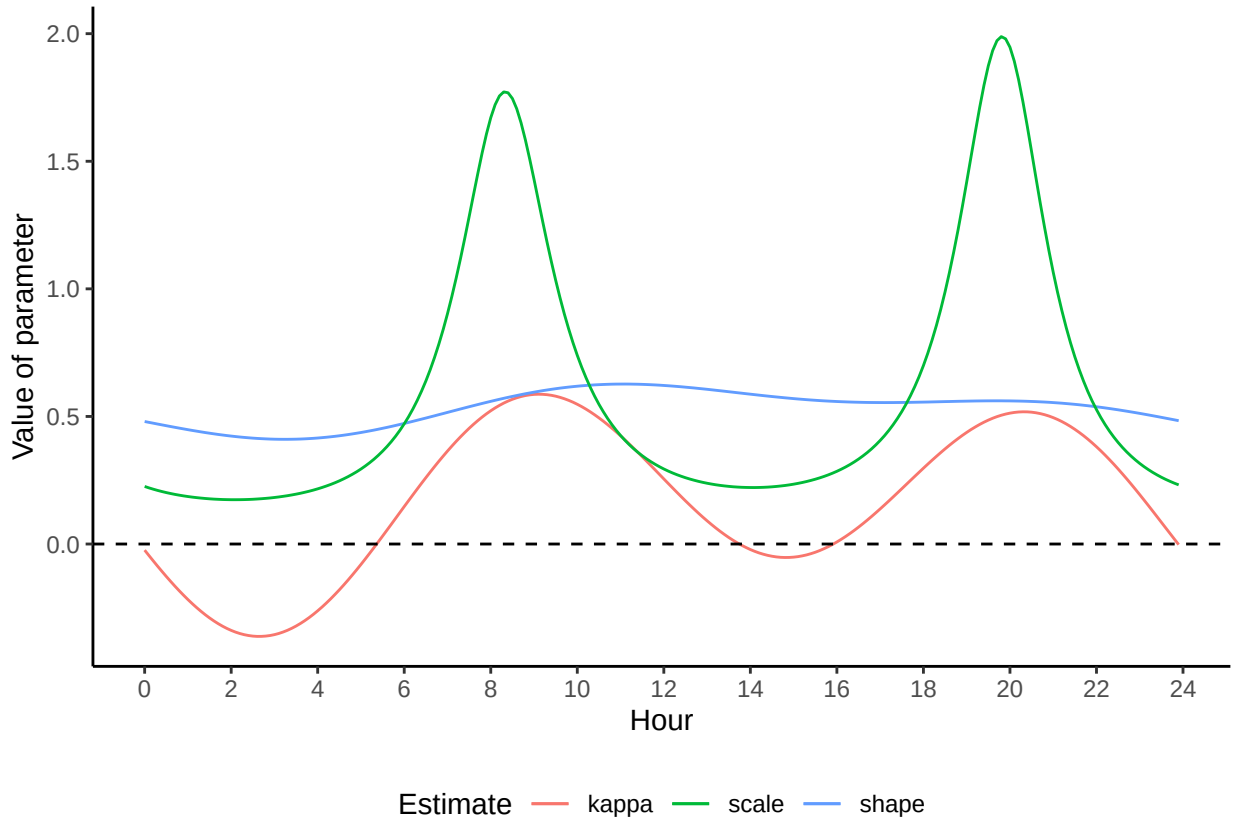
Temporally dynamic natural-scale external selection parameters

```
ggplot() +
  geom_path(data = hour_coefs_nat_long_2p %>%
              filter(!coef %in% c("shape", "scale", "kappa", "memory", "memory_2")),
            aes(x = hour, y = value, colour = coef)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_color_discrete("Estimate") +
  theme_classic() +
  theme(legend.position = "bottom")
```



Temporally dynamic natural-scale movement parameters

```
ggplot() +
    geom_path(data = hour_coefs_nat_long_2p %>%
                filter(coef %in% c("shape", "kappa")),
                aes(x = hour, y = value, colour = coef)) +
  geom_path(data = hour_coefs_nat_long_2p %>%
                filter(coef == "scale"),
                aes(x = hour, y = value/1000, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous("Value of parameter") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



Sampling from movement kernel

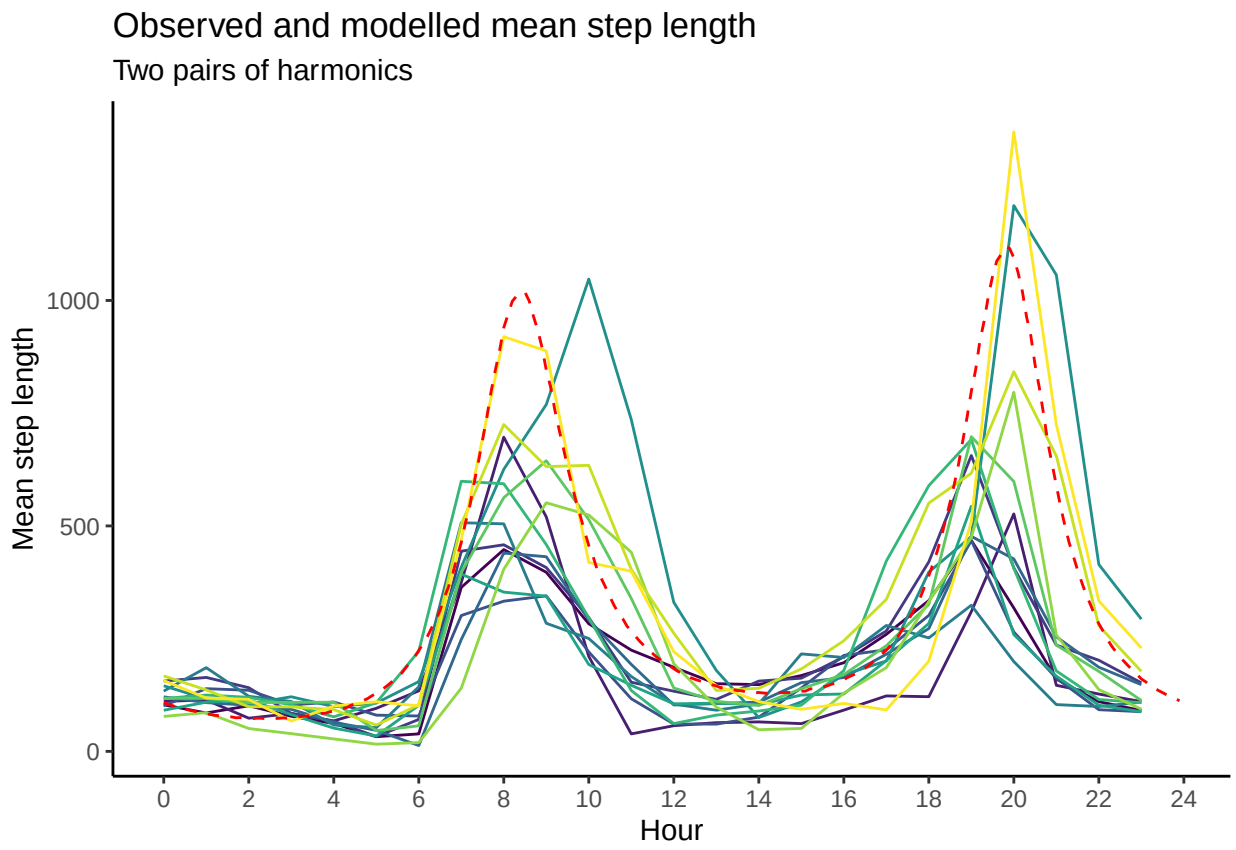
Here we sample from the movement kernel to generate a distribution of step lengths for each hour of the day, to assess how well it matches the observed step lengths.

```
movement_summary <- buffalo_data %>% filter(y == 1) %>%  group_by(id, hour) %>%
  summarise(mean_sl = mean(sl), median_sl = median(sl))
```

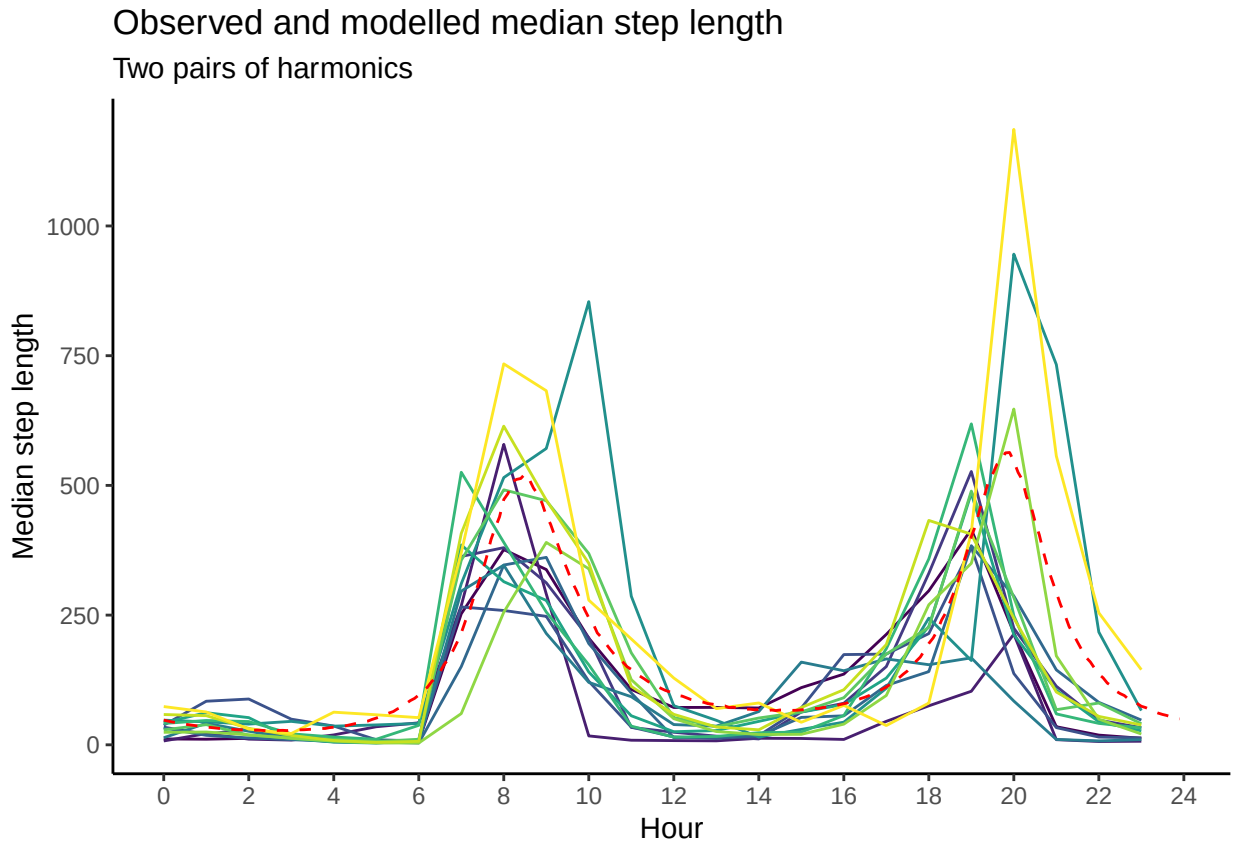
```
## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.
```

```
hour_no <- 1
n <- 1e5

gamma_dist_list <- vector(mode = "list", length = nrow(hour_coefs_nat_df_2p))
gamma_mean <- c()
gamma_median <- c()
gamma_ratio <- c()

for(hour_no in 1:nrow(hour_coefs_nat_df_2p)) {

gamma_dist_list[[hour_no]] <- rgamma(n,
                                     shape = hour_coefs_nat_df_2p$shape[hour_no],
                                     scale = hour_coefs_nat_df_2p$scale[hour_no])

gamma_mean[hour_no] <- mean(gamma_dist_list[[hour_no]])
gamma_median[hour_no] <- median(gamma_dist_list[[hour_no]])
gamma_ratio[hour_no] <- gamma_mean[hour_no] / gamma_median[hour_no]
```

```
}

gamma_df_2p <- data.frame(model = "2p",
                          hour = hour_coefs_nat_df_2p$hour,
                          mean = gamma_mean,
                          median = gamma_median,
                          ratio = gamma_ratio)

mean_sl_2p <- ggplot() +
  geom_path(data = movement_summary,
            aes(x = hour, y = mean_sl, colour = factor(id))) +
  geom_path(data = gamma_df_2p,
            aes(x = hour, y = mean),
            colour = "red", linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Mean step length") +
  scale_colour_viridis_d("Buffalo") +
  ggtitle("Observed and modelled mean step length",
          subtitle = "Two pairs of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

mean_sl_2p
```



```
median_sl_2p <- ggplot() +
  geom_path(data = movement_summary,
            aes(x = hour, y = median_sl, colour = factor(id))) +
  geom_path(data = gamma_df_2p,
            aes(x = hour, y = median),
            colour = "red", linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Median step length") +
  scale_colour_viridis_d("Buffalo") +
  ggtitle("Observed and modelled median step length",
          subtitle = "Two pairs of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

median_sl_2p
```




```
# across the hours
buffalo_data_all %>% filter(y == 1) %>%  group_by(id) %>%
  summarise(mean_sl = mean(sl),
            median_sl = median(sl),
            ratio = mean_sl/median_sl)
```

```
## # A tibble: 13 x 4
##       id mean_sl median_sl ratio
##    <dbl>   <dbl>     <dbl> <dbl>
##  1  2005    204.      98.0  2.08
```

```
##  2  2014    142.      15.9  8.94
##  3  2018    249.     103.   2.41
##  4  2021    183.      94.8  1.93
##  5  2022    216.      78.4  2.75
##  6  2024    229.      77.9  2.94
##  7  2039    357.     124.   2.87
##  8  2154    198.      95.6  2.07
##  9  2158    219.      84.8  2.58
## 10  2223    249.      80.2  3.10
## 11  2327    200.      51.3  3.91
## 12  2387    327.     111.   2.95
## 13  2393    322.     127.   2.53
```

```
buffalo_data_all %>% filter(y == 1) %>%
  summarise(mean_sl = mean(sl),
            median_sl = median(sl),
            ratio = mean_sl/median_sl)
```

```
## # A tibble: 1 x 3
##   mean_sl median_sl ratio
##     <dbl>     <dbl> <dbl>
## 1    238.      86.9  2.74
```

```
gamma_df_2p |> summarise(mean_mean = mean(mean),
                         median_mean = mean(median),
                         ratio_mean = mean_mean/median_mean)
```

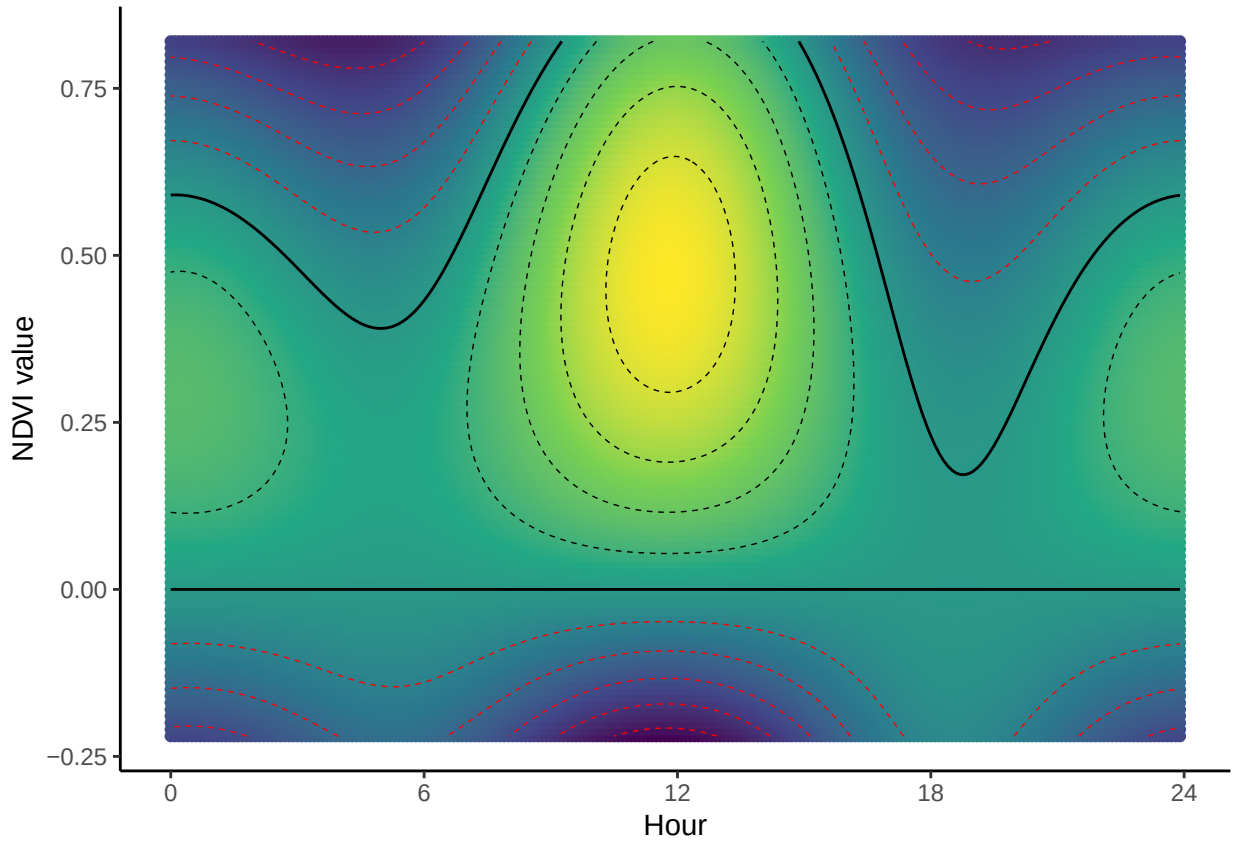
```
##   mean_mean median_mean ratio_mean
## 1   335.187     166.037   2.018749
```

Creating selection surfaces

Selection surface for NDVI

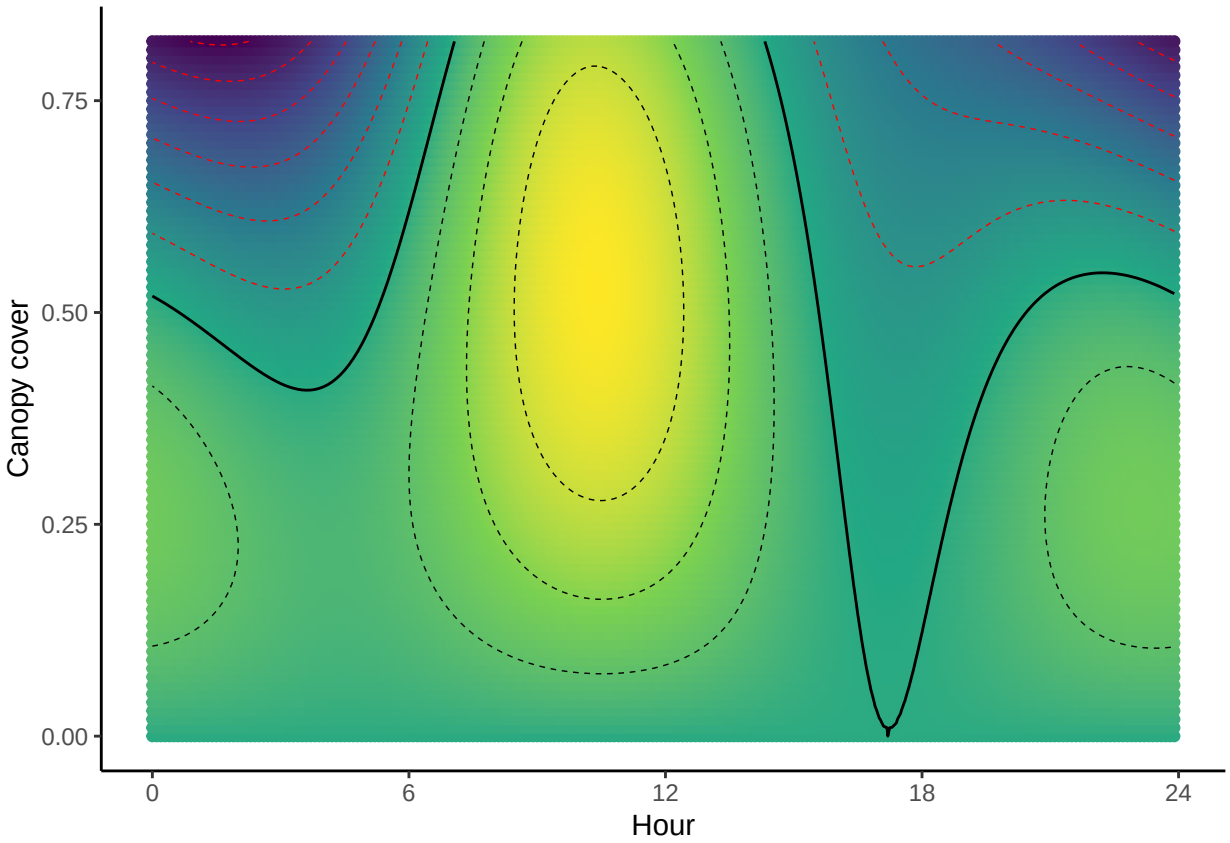
```
ndvi_min <- min(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_max <- max(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_seq <- seq(ndvi_min, ndvi_max, by = 0.01)

# Create empty data frame
ndvi_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_2p),
                                       nrow = length(ndvi_seq)))
for(i in 1:nrow(hour_coefs_nat_df_2p)) {
  # Assign the vector as a column to the dataframe
  ndvi_fresponse_df[,i] <- (hour_coefs_nat_df_2p$ndvi[i] * ndvi_seq) +
    (hour_coefs_nat_df_2p$ndvi_2[i] * (ndvi_seq ^ 2))
}

ndvi_fresponse_df <- data.frame(ndvi_seq, ndvi_fresponse_df)
colnames(ndvi_fresponse_df) <- c("ndvi", hour)
ndvi_fresponse_long <- pivot_longer(ndvi_fresponse_df, cols = !1,
                                    names_to = "hour")

ndvi_contour_max <- max(ndvi_fresponse_long$value) # 0.7890195
ndvi_contour_min <- min(ndvi_fresponse_long$value) # -0.7945691
ndvi_contour_increment <- (ndvi_contour_max-ndvi_contour_min)/10
```

```r
ndvi_quad_2p <- ggplot(data = ndvi_fresponse_long,
                       aes(x = as.numeric(hour), y = ndvi)) +
  geom_point(aes(colour = value)) + # colour = "white"
  geom_contour(aes(z = value),
               breaks = seq(ndvi_contour_increment,
                            ndvi_contour_max,
                            ndvi_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-ndvi_contour_increment,
                            ndvi_contour_min,
                            -ndvi_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("NDVI value", breaks = seq(-1, 1, 0.25)) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Normalised Difference Vegetation Index (NDVI)") +
  theme_classic() +
  theme(legend.position = "none")

ndvi_quad_2p
```



Canopy cover

```
canopy_min <- min(buffalo_data$canopy_01, na.rm = TRUE)
canopy_max <- max(buffalo_data$canopy_01, na.rm = TRUE)
canopy_seq <- seq(canopy_min, canopy_max, by = 0.01)

# Create empty data frame
canopy_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_2p),
                                         nrow = length(canopy_seq)))
for(i in 1:nrow(hour_coefs_nat_df_2p)) {
  # Assign the vector as a column to the dataframe
  canopy_fresponse_df[,i] <- (hour_coefs_nat_df_2p$canopy[i] * canopy_seq) +
    (hour_coefs_nat_df_2p$canopy_2[i] * (canopy_seq ^ 2))
}

canopy_fresponse_df <- data.frame(canopy_seq, canopy_fresponse_df)
colnames(canopy_fresponse_df) <- c("canopy", hour)
canopy_fresponse_long <- pivot_longer(canopy_fresponse_df, cols = !1,
                                      names_to = "hour")

canopy_contour_min <- min(canopy_fresponse_long$value) # 0
canopy_contour_max <- max(canopy_fresponse_long$value) # 2.181749
canopy_contour_increment <- (canopy_contour_max-canopy_contour_min)/10

canopy_quad_2p <- ggplot(data = canopy_fresponse_long,
                         aes(x = as.numeric(hour), y = canopy)) +
  geom_point(aes(colour = value)) +
  geom_contour(aes(z = value),
               breaks = seq(canopy_contour_increment,
                            canopy_contour_max,
                            canopy_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-canopy_contour_increment,
                            canopy_contour_min,
                            -canopy_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("Canopy cover", breaks = seq(0, 1, 0.25)) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Canopy Cover") +
  theme_classic() +
  theme(legend.position = "none")

canopy_quad_2p
```

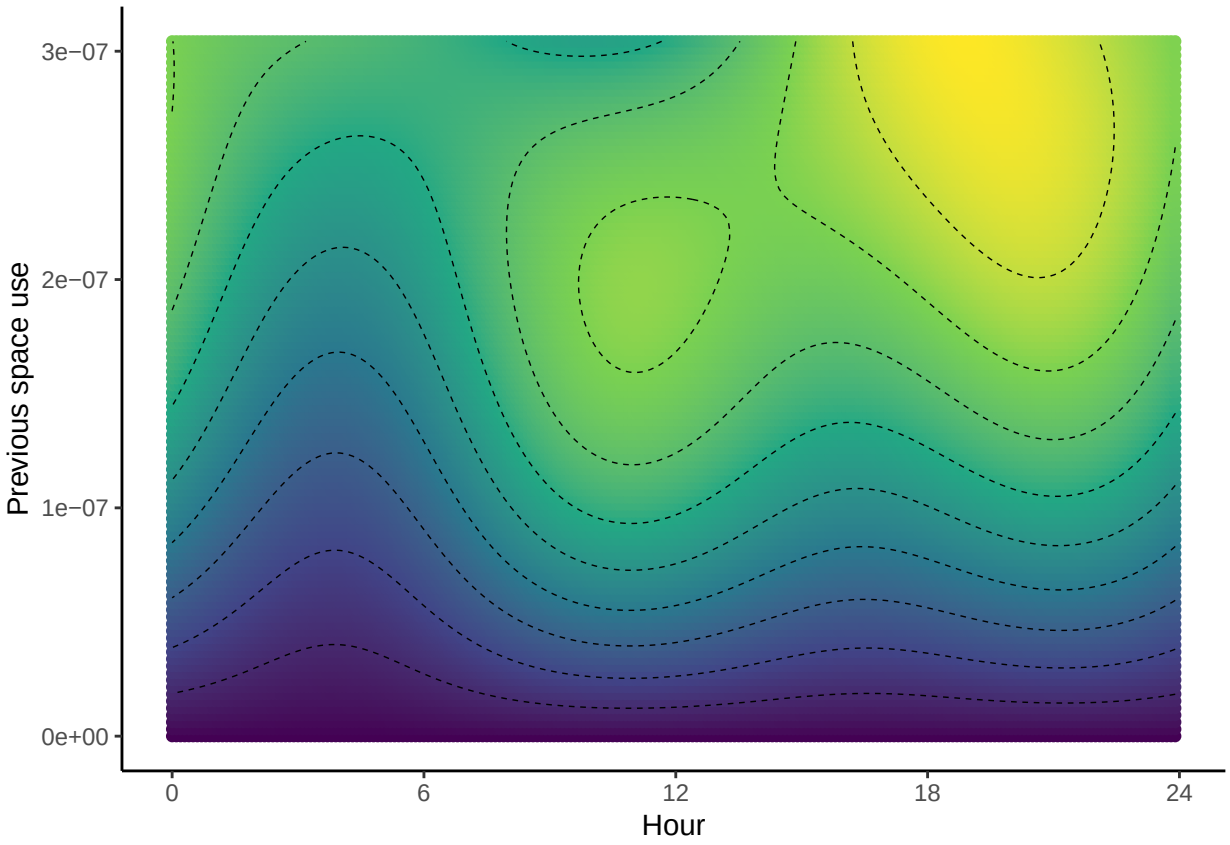


Previous space use density

```
memory_min <- min(buffalo_data$kde_memory_density, na.rm = TRUE)
# memory_max <- max(buffalo_data$kde_memory_density, na.rm = TRUE)
memory_max <- quantile(buffalo_data |> filter(y == 1) |>
                         pull(kde_memory_density), probs = 0.95, na.rm = TRUE)
memory_seq <- seq(memory_min, memory_max, length.out = 100)

# Create empty data frame
memory_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_2p),
                                         nrow = length(memory_seq)))
for(i in 1:nrow(hour_coefs_nat_df_2p)) {
  # Assign the vector as a column to the dataframe
  memory_fresponse_df[,i] <- (hour_coefs_nat_df_2p$memory[i] * memory_seq) +
    (hour_coefs_nat_df_2p$memory_2[i] * (memory_seq ^ 2))
}

memory_fresponse_df <- data.frame(memory_seq, memory_fresponse_df)
colnames(memory_fresponse_df) <- c("memory", hour)
memory_fresponse_long <- pivot_longer(memory_fresponse_df, cols = !1,
                                      names_to = "hour")

memory_contour_min <- min(memory_fresponse_long$value) # 0
memory_contour_max <- max(memory_fresponse_long$value) # 2.181749
memory_contour_increment <- (memory_contour_max-memory_contour_min)/10
```

```
memory_quad_2p <- ggplot(data = memory_fresponse_long,
                         aes(x = as.numeric(hour), y = memory)) +
  geom_point(aes(colour = value)) +
  geom_contour(aes(z = value),
               breaks = seq(memory_contour_increment,
                            memory_contour_max,
                            memory_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-memory_contour_increment,
                            min(-memory_contour_increment, memory_contour_min),
                            -memory_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("Previous space use", labels = scientific) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Relationship to previous space use") +
  theme_classic() +
  theme(legend.position = "none")

memory_quad_2p
```

```
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```



Three pairs of harmonics

```
months_wet <- c(1:4, 11:12)
buffalo_ids <- unique(buffalo_data_all$id)

# buffalo_data <- buffalo_data_all %>% filter(month %in% months_wet) # wet season
buffalo_data <- buffalo_data_all %>% filter(!month %in% months_wet) # dry season

buffalo_data_matrix_unscaled <- buffalo_data %>% transmute(

  ndvi = ndvi_temporal,
  ndvi_s1 = ndvi_temporal * hour_s1,
  ndvi_s2 = ndvi_temporal * hour_s2,
  ndvi_s3 = ndvi_temporal * hour_s3,
  ndvi_c1 = ndvi_temporal * hour_c1,
  ndvi_c2 = ndvi_temporal * hour_c2,
  ndvi_c3 = ndvi_temporal * hour_c3,

  ndvi_sq = ndvi_temporal ^ 2,
  ndvi_sq_s1 = (ndvi_temporal ^ 2) * hour_s1,
  ndvi_sq_s2 = (ndvi_temporal ^ 2) * hour_s2,
  ndvi_sq_s3 = (ndvi_temporal ^ 2) * hour_s3,
  ndvi_sq_c1 = (ndvi_temporal ^ 2) * hour_c1,
  ndvi_sq_c2 = (ndvi_temporal ^ 2) * hour_c2,
```

```
ndvi_sq_c3 = (ndvi_temporal ^ 2) * hour_c3,

canopy = canopy_01,
canopy_s1 = canopy_01 * hour_s1,
canopy_s2 = canopy_01 * hour_s2,
canopy_s3 = canopy_01 * hour_s3,
canopy_c1 = canopy_01 * hour_c1,
canopy_c2 = canopy_01 * hour_c2,
canopy_c3 = canopy_01 * hour_c3,

canopy_sq = canopy_01 ^ 2,
canopy_sq_s1 = (canopy_01 ^ 2) * hour_s1,
canopy_sq_s2 = (canopy_01 ^ 2) * hour_s2,
canopy_sq_s3 = (canopy_01 ^ 2) * hour_s3,
canopy_sq_c1 = (canopy_01 ^ 2) * hour_c1,
canopy_sq_c2 = (canopy_01 ^ 2) * hour_c2,
canopy_sq_c3 = (canopy_01 ^ 2) * hour_c3,

slope = slope,
slope_s1 = slope * hour_s1,
slope_s2 = slope * hour_s2,
slope_s3 = slope * hour_s3,
slope_c1 = slope * hour_c1,
slope_c2 = slope * hour_c2,
slope_c3 = slope * hour_c3,

herby = veg_herby,
herby_s1 = veg_herby * hour_s1,
herby_s2 = veg_herby * hour_s2,
herby_s3 = veg_herby * hour_s3,
herby_c1 = veg_herby * hour_c1,
herby_c2 = veg_herby * hour_c2,
herby_c3 = veg_herby * hour_c3,

spatial_memory = kde_memory_density,
spatial_memory_s1 = kde_memory_density * hour_s1,
spatial_memory_s2 = kde_memory_density * hour_s2,
spatial_memory_s3 = kde_memory_density * hour_s3,
spatial_memory_c1 = kde_memory_density * hour_c1,
spatial_memory_c2 = kde_memory_density * hour_c2,
spatial_memory_c3 = kde_memory_density * hour_c3,

spatial_memory_sq = kde_memory_density ^ 2,
spatial_memory_sq_s1 = (kde_memory_density ^ 2) * hour_s1,
spatial_memory_sq_s2 = (kde_memory_density ^ 2) * hour_s2,
spatial_memory_sq_s3 = (kde_memory_density ^ 2) * hour_s3,
spatial_memory_sq_c1 = (kde_memory_density ^ 2) * hour_c1,
spatial_memory_sq_c2 = (kde_memory_density ^ 2) * hour_c2,
spatial_memory_sq_c3 = (kde_memory_density ^ 2) * hour_c3,

step_l = sl,
step_l_s1 = sl * hour_s1,
step_l_s2 = sl * hour_s2,
```

```
  step_l_s3 = sl * hour_s3,
  step_l_c1 = sl * hour_c1,
  step_l_c2 = sl * hour_c2,
  step_l_c3 = sl * hour_c3,

  log_step_l = log_sl,
  log_step_l_s1 = log_sl * hour_s1,
  log_step_l_s2 = log_sl * hour_s2,
  log_step_l_s3 = log_sl * hour_s3,
  log_step_l_c1 = log_sl * hour_c1,
  log_step_l_c2 = log_sl * hour_c2,
  log_step_l_c3 = log_sl * hour_c3,

  cos_turn_a = cos_ta,
  cos_turn_a_s1 = cos_ta * hour_s1,
  cos_turn_a_s2 = cos_ta * hour_s2,
  cos_turn_a_s3 = cos_ta * hour_s3,
  cos_turn_a_c1 = cos_ta * hour_c1,
  cos_turn_a_c2 = cos_ta * hour_c2,
  cos_turn_a_c3 = cos_ta * hour_c3)

buffalo_data_matrix_scaled <- scale(buffalo_data_matrix_unscaled)

mean_vals <- attr(buffalo_data_matrix_scaled, "scaled:center")
sd_vals <- attr(buffalo_data_matrix_scaled, "scaled:scale")
scaling_attributes <- data.frame(variable = names(buffalo_data_matrix_unscaled),
                                 mean = mean_vals, sd = sd_vals)

buffalo_data_scaled_3p <- data.frame(id = buffalo_data$id,
                                     step_id = buffalo_data$step_id,
                                     y = buffalo_data$y,
                                     buffalo_data_matrix_scaled)
```

Formula with two pairs of harmonics

```
formula_twostep <- y ~

  ndvi +
  ndvi_s1 +
  ndvi_s2 +
  ndvi_s3 +
  ndvi_c1 +
  ndvi_c2 +
  ndvi_c3 +

  ndvi_sq +
  ndvi_sq_s1 +
  ndvi_sq_s2 +
  ndvi_sq_s3 +
  ndvi_sq_c1 +
  ndvi_sq_c2 +
  ndvi_sq_c3 +

  canopy +
```

```
canopy_s1 +
canopy_s2 +
canopy_s3 +
canopy_c1 +
canopy_c2 +
canopy_c3 +

canopy_sq +
canopy_sq_s1 +
canopy_sq_s2 +
canopy_sq_s3 +
canopy_sq_c1 +
canopy_sq_c2 +
canopy_sq_c3 +

slope +
slope_s1 +
slope_s2 +
slope_s3 +
slope_c1 +
slope_c2 +
slope_c3 +

herby +
herby_s1 +
herby_s2 +
herby_s3 +
herby_c1 +
herby_c2 +
herby_c3 +

spatial_memory +
spatial_memory_s1 +
spatial_memory_s2 +
spatial_memory_s3 +
spatial_memory_c1 +
spatial_memory_c2 +
spatial_memory_c3 +

spatial_memory_sq +
spatial_memory_sq_s1 +
spatial_memory_sq_s2 +
spatial_memory_sq_s3 +
spatial_memory_sq_c1 +
spatial_memory_sq_c2 +
spatial_memory_sq_c3 +

step_l +
step_l_s1 +
step_l_s2 +
step_l_s3 +
step_l_c1 +
step_l_c2 +
```

```
  step_l_c3 +

  log_step_l +
  log_step_l_s1 +
  log_step_l_s2 +
  log_step_l_s3 +
  log_step_l_c1 +
  log_step_l_c2 +
  log_step_l_c3 +

  cos_turn_a +
  cos_turn_a_s1 +
  cos_turn_a_s2 +
  cos_turn_a_s3 +
  cos_turn_a_c1 +
  cos_turn_a_c2 +
  cos_turn_a_c3 +

  strata(step_id) +
  cluster(id)
```

Fitting the model

As we have already fitted the model, we will load it here, but if the model_fit file doesn't exist, it will run the model fitting code. Be careful here that if you change the model formula, you will need to delete or rename the model_fit file to re-run the model fitting code, otherwise it will just load the previous model.

```
if(file.exists("outputs/model_twostep_3p_harms_dry.rds")) {

  model_twostep_3p_harms <- readRDS("outputs/model_twostep_3p_harms_dry.rds")

} else {

  tic()
  model_twostep_3p_harms <- Ts.estim(formula = formula_twostep,
            data = buffalo_data_scaled_3p,
            all.m.1 = TRUE,
            D = "UN(1)",
            itermax = 10000)
  toc()

  # save model object
  saveRDS(model_twostep_3p_harms, file = "outputs/model_twostep_3p_harms_dry.rds")

  beep(sound = 2)

}
```

Reading in saved model object

```
# model_twostep_3p_harms

# model_twostep_3p_harms$beta
# model_twostep_3p_harms$se
# model_twostep_3p_harms$vcov
```

```
# diag(model_twostep_3p_harms$D) # between cluster variance
# model_twostep_3p_harms$r.effect # individual estimates

# creating dataframe of coefficients
coefs_clr <- data.frame(coefs = names(model_twostep_3p_harms$beta),
                        value = model_twostep_3p_harms$beta)
coefs_clr$scale_sd <- scaling_attributes$sd
coefs_clr <- coefs_clr %>% mutate(value_nat = value / scale_sd)
head(coefs_clr)
```

```
##           coefs      value  scale_sd  value_nat
## ndvi       ndvi  1.1967830 0.1429054  8.3746549
## ndvi_s1 ndvi_s1  0.4191553 0.2371431  1.7675208
## ndvi_s2 ndvi_s2 -0.1087782 0.2372765 -0.4584446
## ndvi_s3 ndvi_s3  0.2679367 0.2396650  1.1179635
## ndvi_c1 ndvi_c1 -1.0141903 0.2426054 -4.1804104
## ndvi_c2 ndvi_c2  1.0862009 0.2425239  4.4787384
```

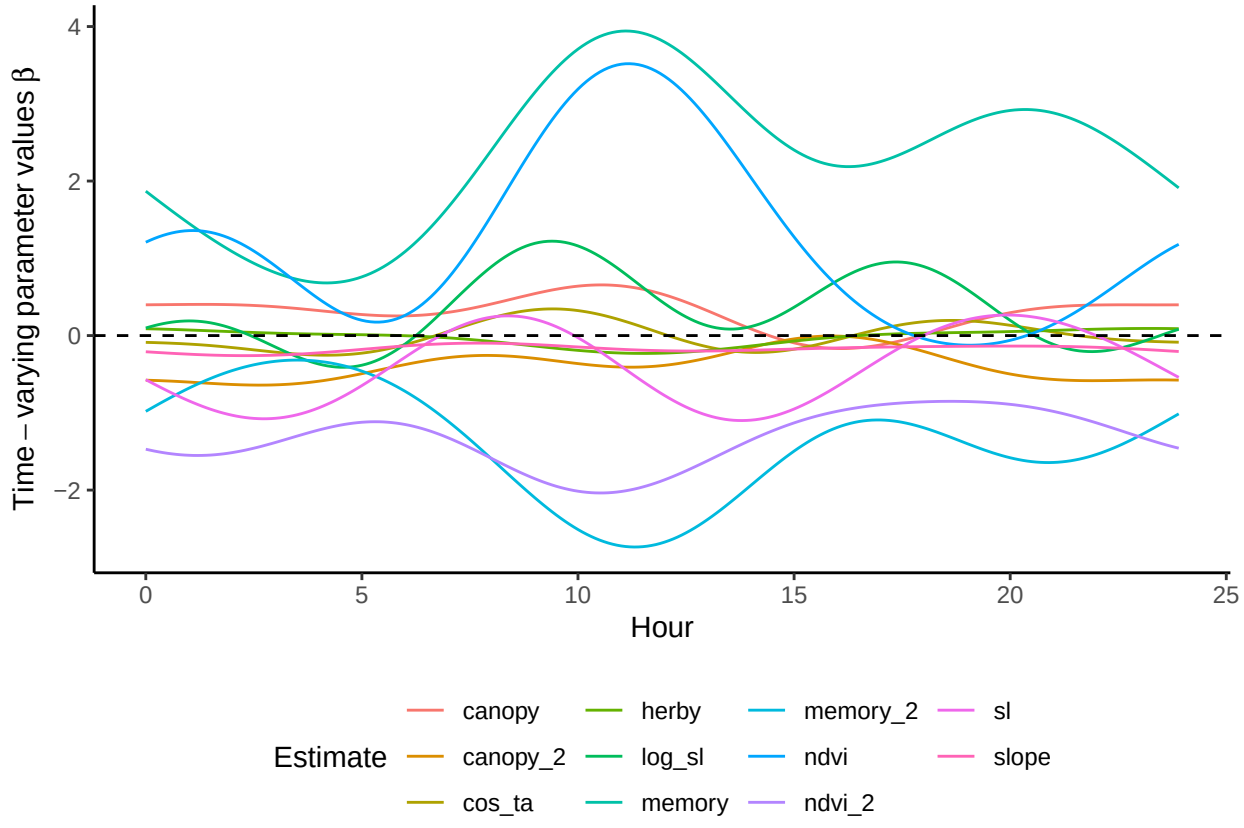
Reconstructing coefficients with three pairs of harmonics, with quadratic terms

```
# hour <- seq(0,23,1)
hour <- seq(0,23.9,0.1)

hour_harmonics_df <- data.frame("linear_term" = rep(1, length(hour)),
                                "hour_s1" = sin(2*pi*hour/24),
                                "hour_s2" = sin(4*pi*hour/24),
                                "hour_s3" = sin(6*pi*hour/24),
                                "hour_c1" = cos(2*pi*hour/24),
                                "hour_c2" = cos(4*pi*hour/24),
                                "hour_c3" = cos(6*pi*hour/24))

harmonics_scaled_df_3p <- data.frame(
  "hour" = hour,
  "ndvi" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "ndvi_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "slope" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("slope", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "herby" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("herby", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory", coefs) & !grepl("sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory_2" = as.numeric(
```

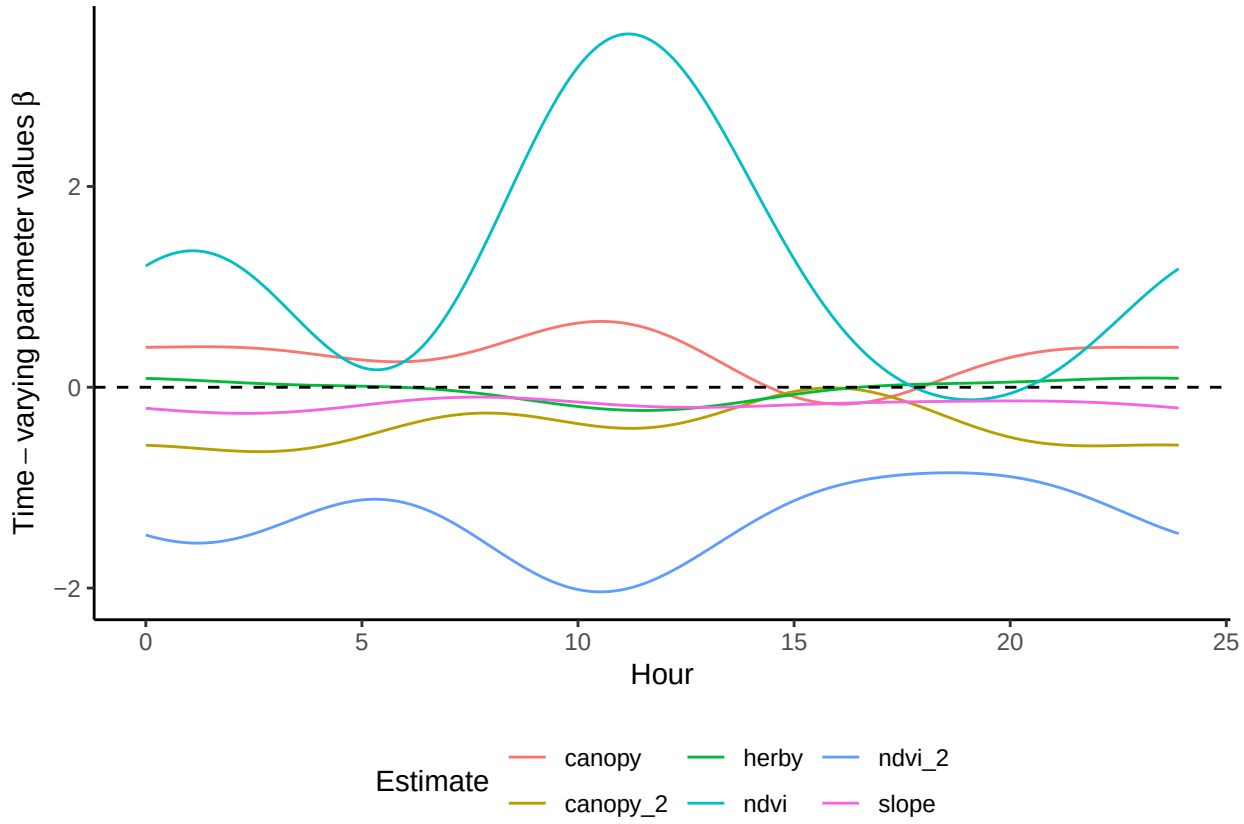
```
    coefs_clr %>% dplyr::filter(grepl("memory_sq", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("step_l", coefs) & !grepl("log", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "log_sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("log_step_l", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "cos_ta" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("cos", coefs)) %>%
      pull(value) %>% t() %*% t(as.matrix(hour_harmonics_df))))

harmonics_scaled_long_3p <- pivot_longer(harmonics_scaled_df_3p, cols = !1,
                                         names_to = "coef")
```

Temporally dynamic scaled external selection parameters

```
ggplot() +
    geom_path(data = harmonics_scaled_long_3p,
              aes(x = hour, y = value, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
    scale_x_continuous("Hour") +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



```
ggplot() +
    geom_path(data = harmonics_scaled_long_3p %>%
              filter(!coef %in% c("sl", "log_sl", "cos_ta", "memory", "memory_2")),
              aes(x = hour, y = value, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(Time-varying~parameter~values~beta)) +
    scale_x_continuous("Hour") +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



For the natural scale parameters

```
harmonics_nat_df_3p <- data.frame(
  "hour" = hour,
  "ndvi" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "ndvi_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("ndvi_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "canopy_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("canopy_sq", coefs)) %>%
```

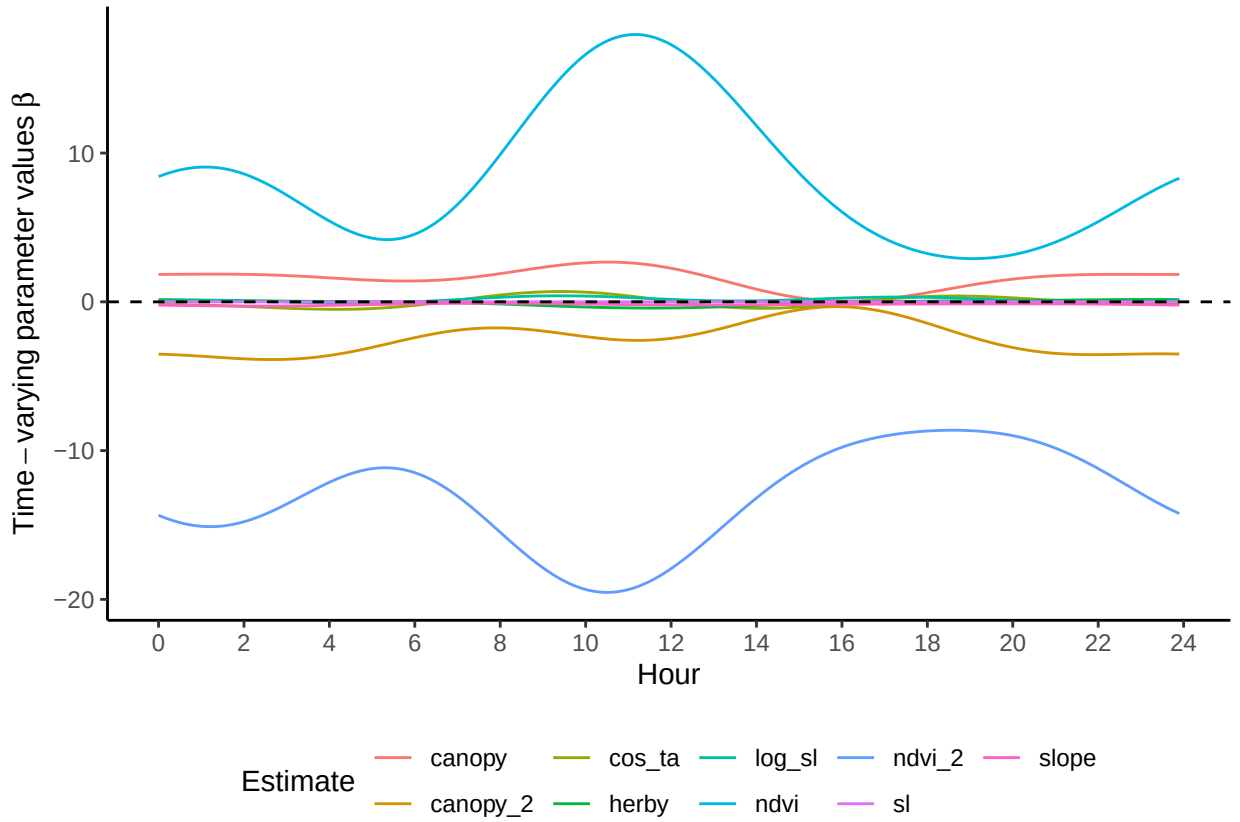
```
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "slope" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("slope", coefs) & !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "herby" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("herby", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory", coefs)& !grepl("sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "memory_2" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("memory_sq", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("step_l", coefs) & !grepl("log", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "log_sl" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("log_step_l", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))),
  "cos_ta" = as.numeric(
    coefs_clr %>% dplyr::filter(grepl("cos", coefs)) %>%
      pull(value_nat) %>% t() %*% t(as.matrix(hour_harmonics_df))))
```

Reconstructing the Gamma and von Mises distributions from the tentative distributions (from Fieberg et al 2021: Appendix C)

```
tentative_shape <- 0.438167
tentative_scale <- 534.3507
tentative_kappa <- 0.1848126

hour_coefs_nat_df_3p <- harmonics_nat_df_3p %>%
  mutate(shape = tentative_shape + log_sl,
         scale = 1/((1/tentative_scale) - sl),
         kappa = tentative_kappa + cos_ta)

# save the coefficients to use in the simulations
# write_csv(hour_coefs_nat_df_3p,
#           paste0("outputs/TwoStep_3pDaily_coefs_dry_",Sys.Date(), ".csv"))

# turning into a long data frame
hour_coefs_nat_long_3p <- pivot_longer(hour_coefs_nat_df_3p, cols = !1,
                                       names_to = "coef")
```

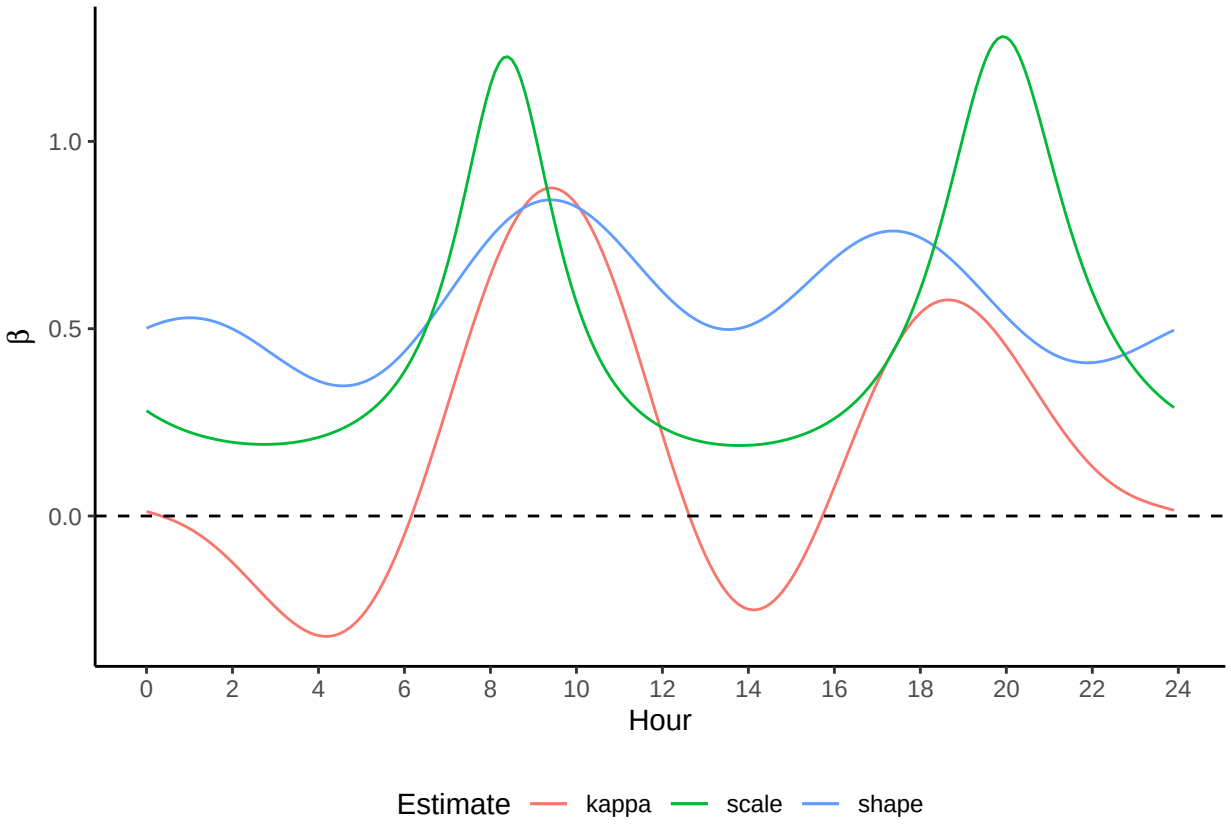
Temporally dynamic natural-scale external selection parameters

```
ggplot() +
  geom_path(data = hour_coefs_nat_long_3p %>%
              filter(!coef %in% c("shape", "scale", "kappa", "memory", "memory_2")),
            aes(x = hour, y = value, colour = coef)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  scale_y_continuous(expression(Time~varying~parameter~values~beta)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_color_discrete("Estimate") +
  theme_classic() +
  theme(legend.position = "bottom")
```



Temporally dynamic natural-scale movement parameters

```
ggplot() +
    geom_path(data = hour_coefs_nat_long_3p %>%
                filter(coef %in% c("shape", "kappa")),
                aes(x = hour, y = value, colour = coef)) +
  geom_path(data = hour_coefs_nat_long_3p %>%
                filter(coef == "scale"),
                aes(x = hour, y = value/1000, colour = coef)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    scale_y_continuous(expression(beta)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
    scale_color_discrete("Estimate") +
    theme_classic() +
    theme(legend.position = "bottom")
```



Sampling from movement kernel

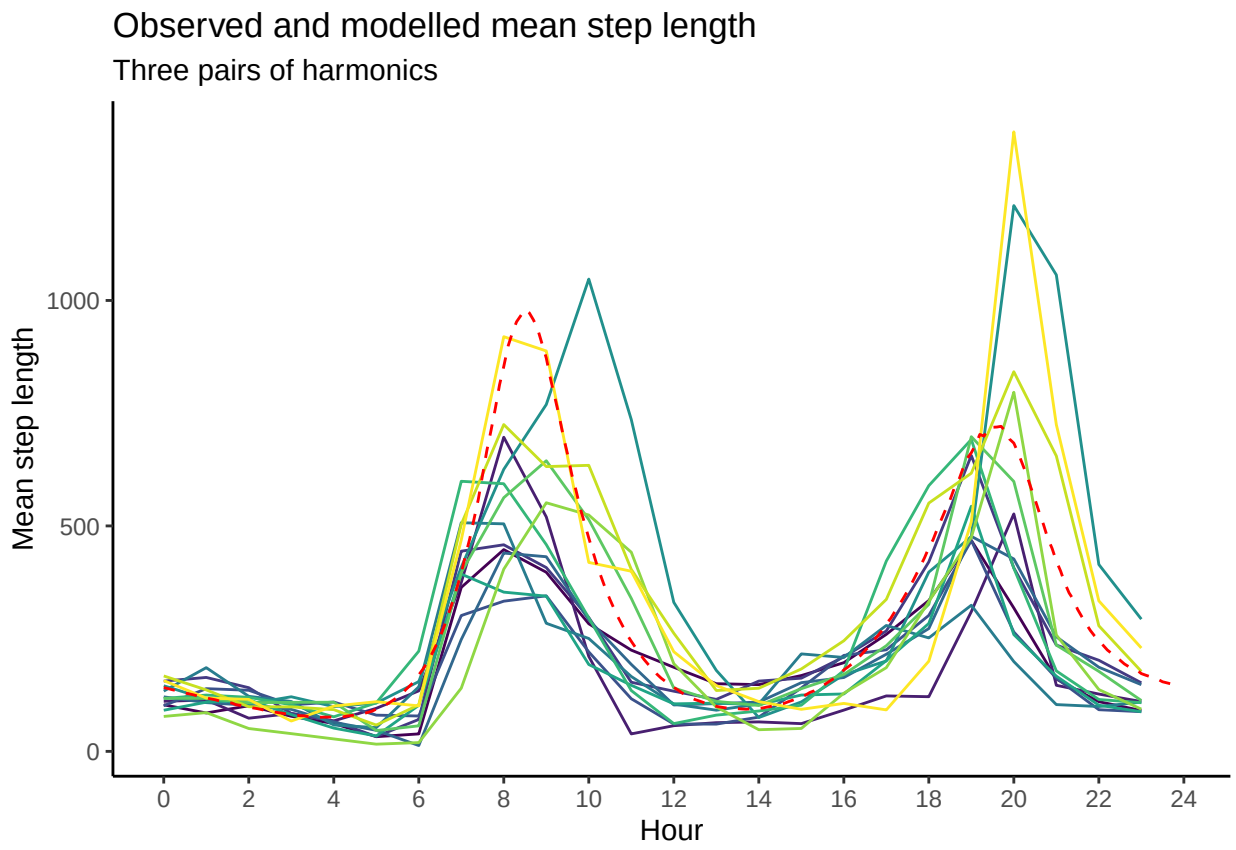
Here we sample from the movement kernel to generate a distribution of step lengths for each hour of the day, to assess how well it matches the observed step lengths.

```
movement_summary <- buffalo_data %>% filter(y == 1) %>%  group_by(id, hour) %>%
  summarise(mean_sl = mean(sl), median_sl = median(sl))
```

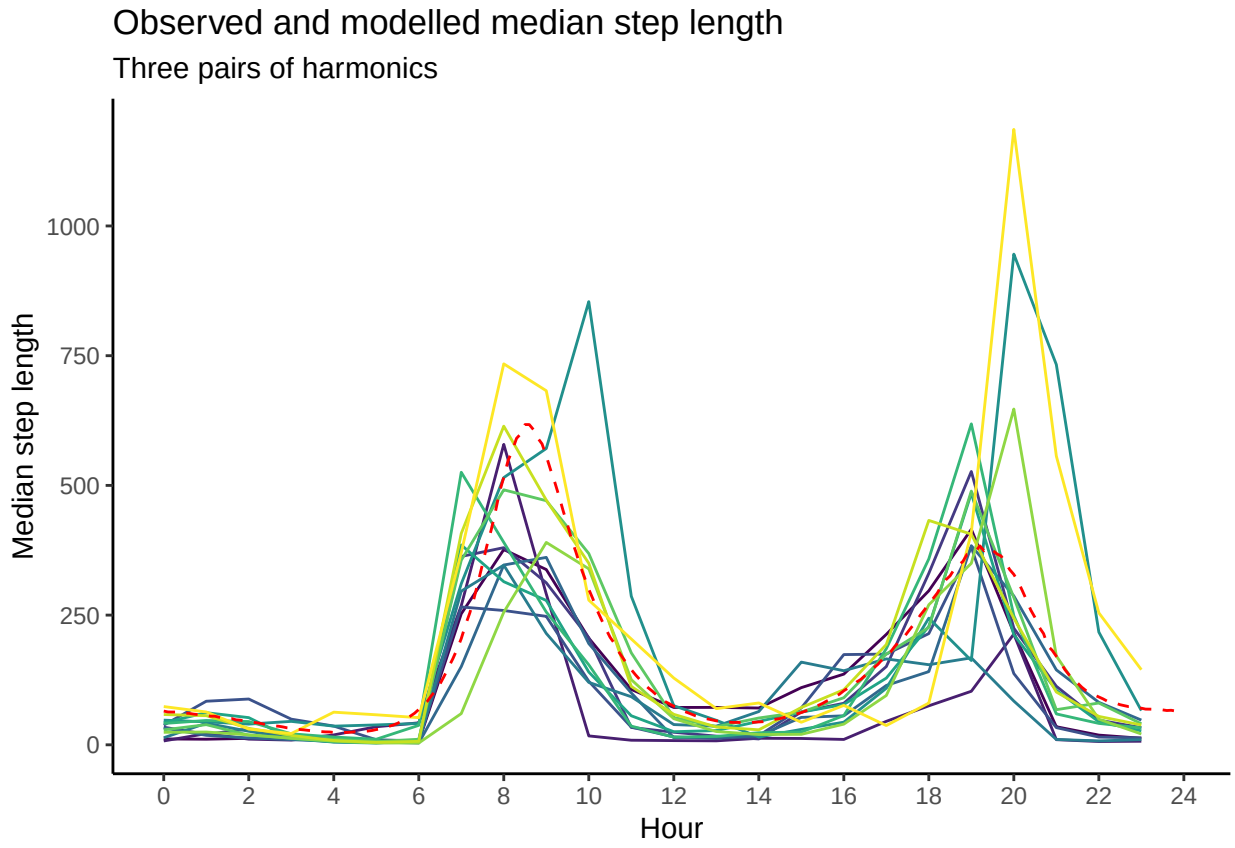
```
## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.
```

```
hour_no <- 1
n <- 1e5

gamma_dist_list <- vector(mode = "list", length = nrow(hour_coefs_nat_df_3p))
gamma_mean <- c()
gamma_median <- c()
gamma_ratio <- c()

for(hour_no in 1:nrow(hour_coefs_nat_df_3p)) {

gamma_dist_list[[hour_no]] <- rgamma(n,
                                     shape = hour_coefs_nat_df_3p$shape[hour_no],
                                     scale = hour_coefs_nat_df_3p$scale[hour_no])

gamma_mean[hour_no] <- mean(gamma_dist_list[[hour_no]])
gamma_median[hour_no] <- median(gamma_dist_list[[hour_no]])
gamma_ratio[hour_no] <- gamma_mean[hour_no] / gamma_median[hour_no]
```

```
}

gamma_df_3p <- data.frame(model = "3p",
                          hour = hour_coefs_nat_df_3p$hour,
                          mean = gamma_mean,
                          median = gamma_median,
                          ratio = gamma_ratio)

mean_sl_3p <- ggplot() +
  geom_path(data = movement_summary,
            aes(x = hour, y = mean_sl, colour = factor(id))) +
  geom_path(data = gamma_df_3p,
            aes(x = hour, y = mean),
            colour = "red", linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Mean step length") +
  scale_colour_viridis_d("Buffalo") +
  ggtitle("Observed and modelled mean step length",
          subtitle = "Three pairs of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

mean_sl_3p
```



```
median_sl_3p <- ggplot() +
  geom_path(data = movement_summary,
            aes(x = hour, y = median_sl, colour = factor(id))) +
  geom_path(data = gamma_df_3p,
            aes(x = hour, y = median),
            colour = "red", linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Median step length") +
  scale_colour_viridis_d("Buffalo") +
  ggtitle("Observed and modelled median step length",
          subtitle = "Three pairs of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

median_sl_3p
```




```
# across the hours
buffalo_data_all %>% filter(y == 1) %>%  group_by(id) %>%
  summarise(mean_sl = mean(sl),
            median_sl = median(sl),
            ratio = mean_sl/median_sl)
```

```
## # A tibble: 13 x 4
##       id mean_sl median_sl ratio
##    <dbl>   <dbl>     <dbl> <dbl>
##  1  2005    204.      98.0  2.08
```

```
##  2  2014    142.      15.9  8.94
##  3  2018    249.     103.   2.41
##  4  2021    183.      94.8  1.93
##  5  2022    216.      78.4  2.75
##  6  2024    229.      77.9  2.94
##  7  2039    357.     124.   2.87
##  8  2154    198.      95.6  2.07
##  9  2158    219.      84.8  2.58
## 10  2223    249.      80.2  3.10
## 11  2327    200.      51.3  3.91
## 12  2387    327.     111.   2.95
## 13  2393    322.     127.   2.53
```

```
buffalo_data_all %>% filter(y == 1) %>%
  summarise(mean_sl = mean(sl),
            median_sl = median(sl),
            ratio = mean_sl/median_sl)
```

```
## # A tibble: 1 x 3
##   mean_sl median_sl ratio
##     <dbl>     <dbl> <dbl>
## 1    238.      86.9  2.74
```

```
gamma_df_3p |> summarise(mean_mean = mean(mean),
                         median_mean = mean(median),
                         ratio_mean = mean_mean/median_mean)
```

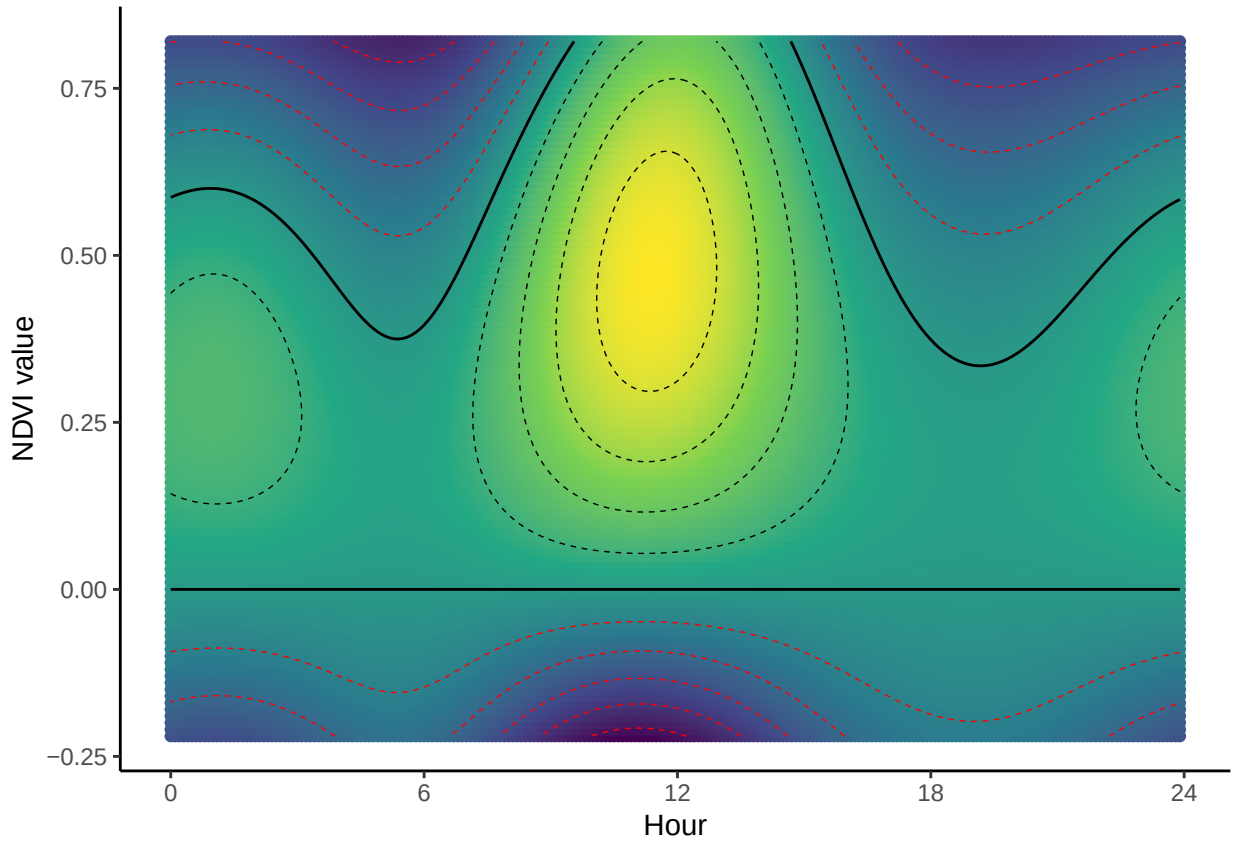
```
##   mean_mean median_mean ratio_mean
## 1  298.7919     160.192   1.865211
```

Creating selection surfaces

Contour plot for NDVI

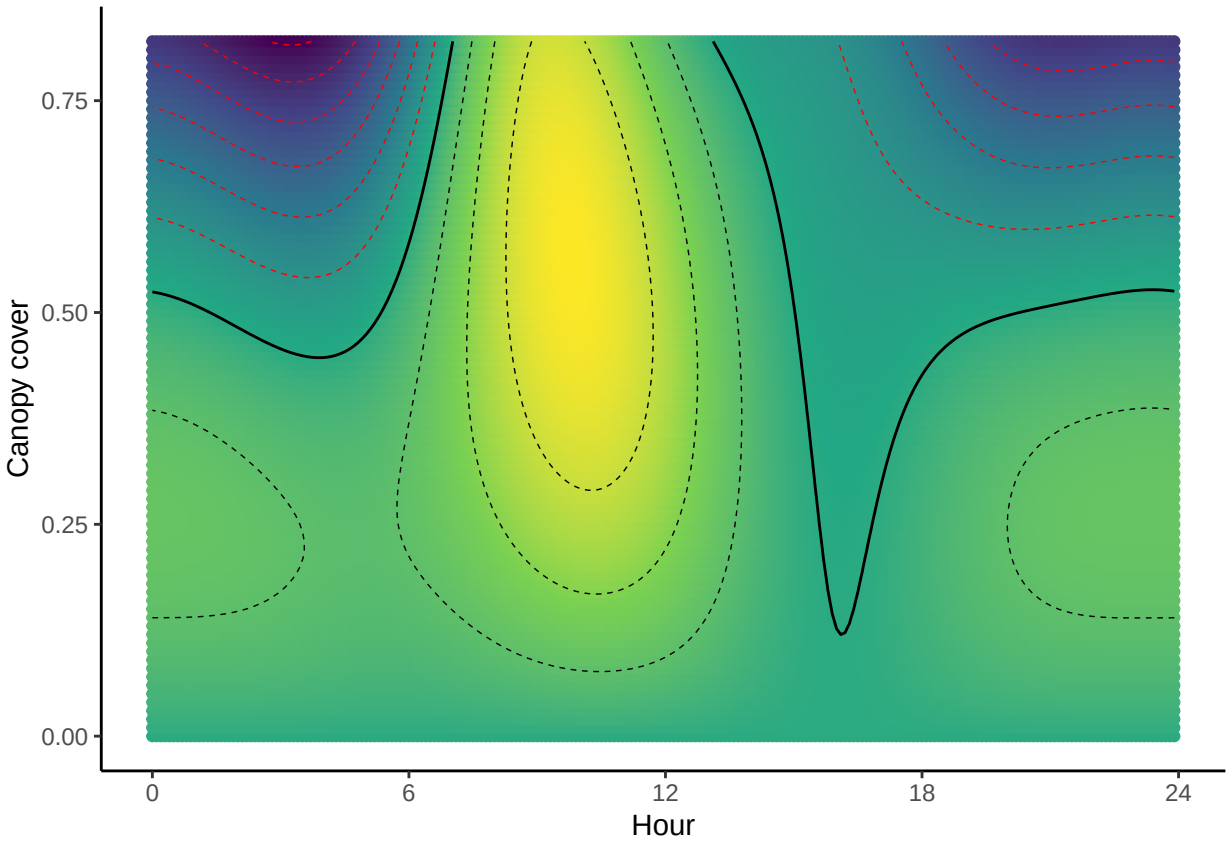
```
ndvi_min <- min(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_max <- max(buffalo_data$ndvi_temporal, na.rm = TRUE)
ndvi_seq <- seq(ndvi_min, ndvi_max, by = 0.01)

# Create empty data frame
ndvi_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_3p),
                                       nrow = length(ndvi_seq)))
for(i in 1:nrow(hour_coefs_nat_df_3p)) {
  # Assign the vector as a column to the dataframe
  ndvi_fresponse_df[,i] <- (hour_coefs_nat_df_3p$ndvi[i] * ndvi_seq) +
    (hour_coefs_nat_df_3p$ndvi_2[i] * (ndvi_seq ^ 2))
}

ndvi_fresponse_df <- data.frame(ndvi_seq, ndvi_fresponse_df)
colnames(ndvi_fresponse_df) <- c("ndvi", hour)
ndvi_fresponse_long <- pivot_longer(ndvi_fresponse_df, cols = !1,
                                    names_to = "hour")

ndvi_contour_max <- max(ndvi_fresponse_long$value) # 0.7890195
ndvi_contour_min <- min(ndvi_fresponse_long$value) # -0.7945691
ndvi_contour_increment <- (ndvi_contour_max-ndvi_contour_min)/10
```

```
ndvi_quad_3p <- ggplot(data = ndvi_fresponse_long,
                       aes(x = as.numeric(hour), y = ndvi)) +
  geom_point(aes(colour = value)) + # colour = "white"
  geom_contour(aes(z = value),
               breaks = seq(ndvi_contour_increment,
                            ndvi_contour_max,
                            ndvi_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-ndvi_contour_increment,
                            ndvi_contour_min,
                            -ndvi_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("NDVI value", breaks = seq(-1, 1, 0.25)) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Normalised Difference Vegetation Index (NDVI)") +
  theme_classic() +
  theme(legend.position = "none")

ndvi_quad_3p
```



Canopy cover

```r
canopy_min <- min(buffalo_data$canopy_01, na.rm = TRUE)
canopy_max <- max(buffalo_data$canopy_01, na.rm = TRUE)
canopy_seq <- seq(canopy_min, canopy_max, by = 0.01)

# Create empty data frame
canopy_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_3p),
                                         nrow = length(canopy_seq)))
for(i in 1:nrow(hour_coefs_nat_df_3p)) {
  # Assign the vector as a column to the dataframe
  canopy_fresponse_df[,i] <- (hour_coefs_nat_df_3p$canopy[i] * canopy_seq) +
    (hour_coefs_nat_df_3p$canopy_2[i] * (canopy_seq ^ 2))
}

canopy_fresponse_df <- data.frame(canopy_seq, canopy_fresponse_df)
colnames(canopy_fresponse_df) <- c("canopy", hour)
canopy_fresponse_long <- pivot_longer(canopy_fresponse_df, cols = !1,
                                      names_to = "hour")

canopy_contour_min <- min(canopy_fresponse_long$value) # 0
canopy_contour_max <- max(canopy_fresponse_long$value) # 2.181749
canopy_contour_increment <- (canopy_contour_max-canopy_contour_min)/10

canopy_quad_3p <- ggplot(data = canopy_fresponse_long,
                         aes(x = as.numeric(hour), y = canopy)) +
  geom_point(aes(colour = value)) +
  geom_contour(aes(z = value),
               breaks = seq(canopy_contour_increment,
                            canopy_contour_max,
                            canopy_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-canopy_contour_increment,
                            canopy_contour_min,
                            -canopy_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("Canopy cover", breaks = seq(0, 1, 0.25)) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Canopy Cover",
  #         subtitle = "Three pairs of harmonics") +
  theme_classic() +
  theme(legend.position = "none")

canopy_quad_3p
```

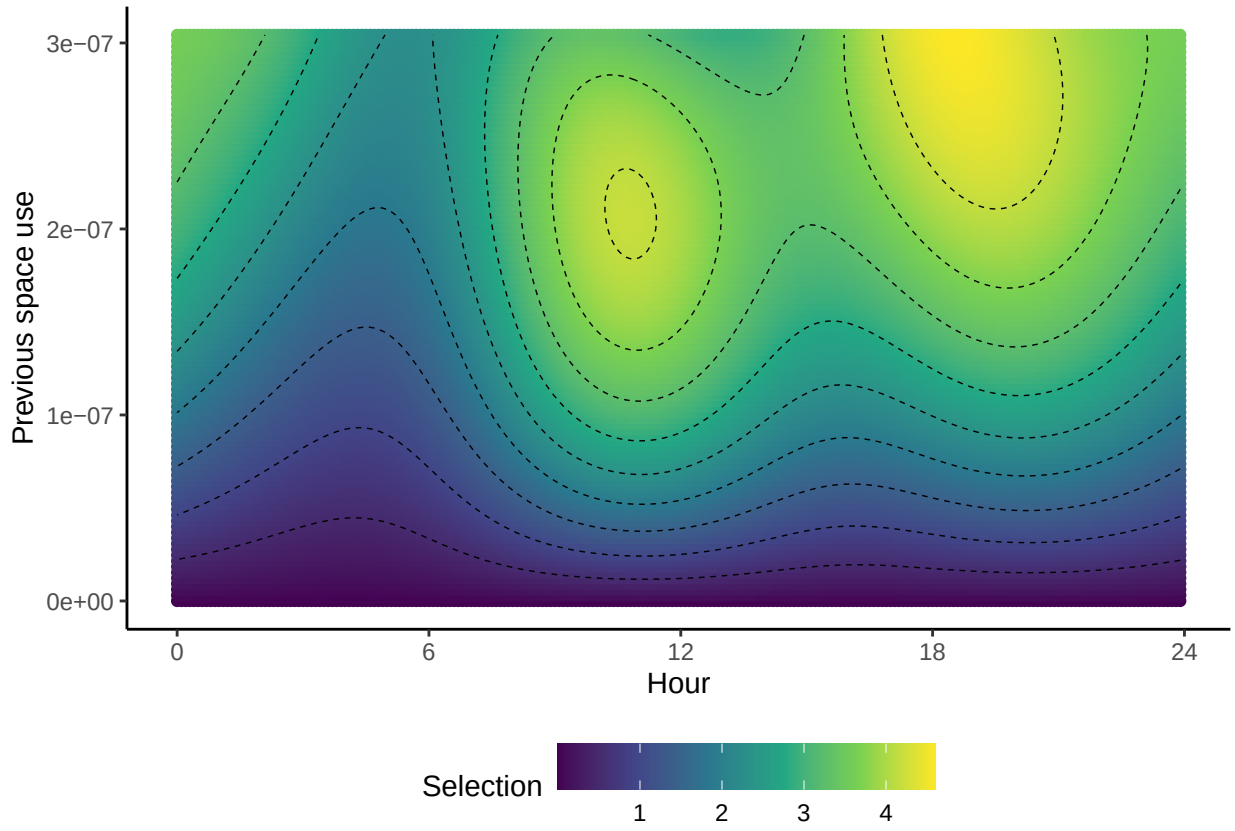


Spatial memory process

```
memory_min <- min(buffalo_data$kde_memory_density, na.rm = TRUE)
# memory_max <- max(buffalo_data$kde_memory_density, na.rm = TRUE)
memory_max <- quantile(buffalo_data |> filter(y == 1) |>
                         pull(kde_memory_density), probs = 0.95, na.rm = TRUE)
memory_seq <- seq(memory_min, memory_max, length.out = 100)

# Create empty data frame
memory_fresponse_df <- data.frame(matrix(ncol = nrow(hour_coefs_nat_df_3p),
                                         nrow = length(memory_seq)))
for(i in 1:nrow(hour_coefs_nat_df_3p)) {
  # Assign the vector as a column to the dataframe
  memory_fresponse_df[,i] <- (hour_coefs_nat_df_3p$memory[i] * memory_seq) +
    (hour_coefs_nat_df_3p$memory_2[i] * (memory_seq ^ 2))
}

memory_fresponse_df <- data.frame(memory_seq, memory_fresponse_df)
colnames(memory_fresponse_df) <- c("memory", hour)
memory_fresponse_long <- pivot_longer(memory_fresponse_df, cols = !1,
                                      names_to = "hour")

memory_contour_min <- min(memory_fresponse_long$value) # 0
memory_contour_max <- max(memory_fresponse_long$value) # 2.181749
memory_contour_increment <- (memory_contour_max-memory_contour_min)/10

memory_quad_3p <- ggplot(data = memory_fresponse_long,
```

```
                        aes(x = as.numeric(hour), y = memory)) +
  geom_point(aes(colour = value)) +
  geom_contour(aes(z = value),
               breaks = seq(memory_contour_increment,
                            memory_contour_max,
                            memory_contour_increment),
               colour = "black", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value),
               breaks = seq(-memory_contour_increment,
                            min(-memory_contour_increment, memory_contour_min),
                            -memory_contour_increment),
               colour = "red", linewidth = 0.25, linetype = "dashed") +
  geom_contour(aes(z = value), breaks = 0, colour = "black", linewidth = 0.5) +
  scale_x_continuous("Hour", breaks = seq(0,24,6)) +
  scale_y_continuous("Previous space use", labels = scientific) +
  scale_colour_viridis_c("Selection") +
  # ggtitle("Relationship to previous space use") +
  theme_classic() +
  theme(legend.position = "bottom",
        legend.key.width = unit(1, "cm"))

memory_quad_3p
```

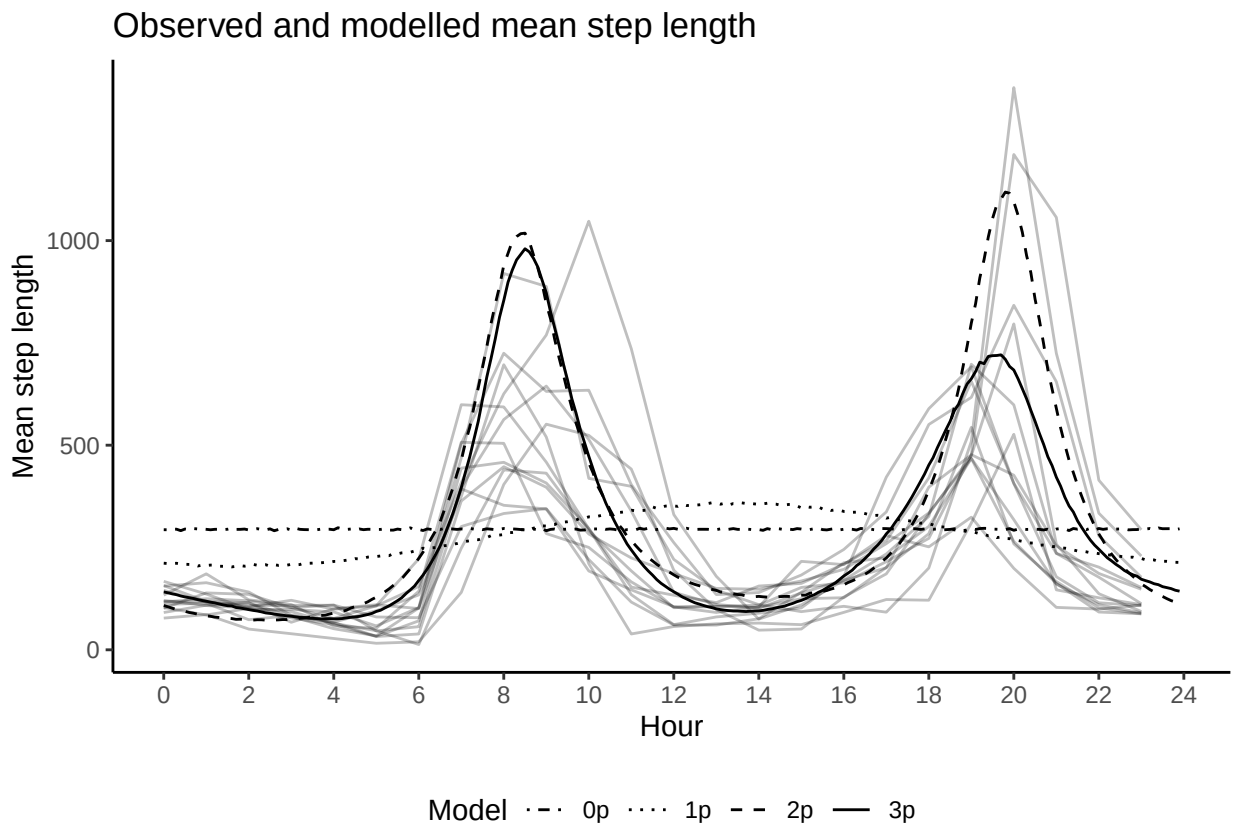
```
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```



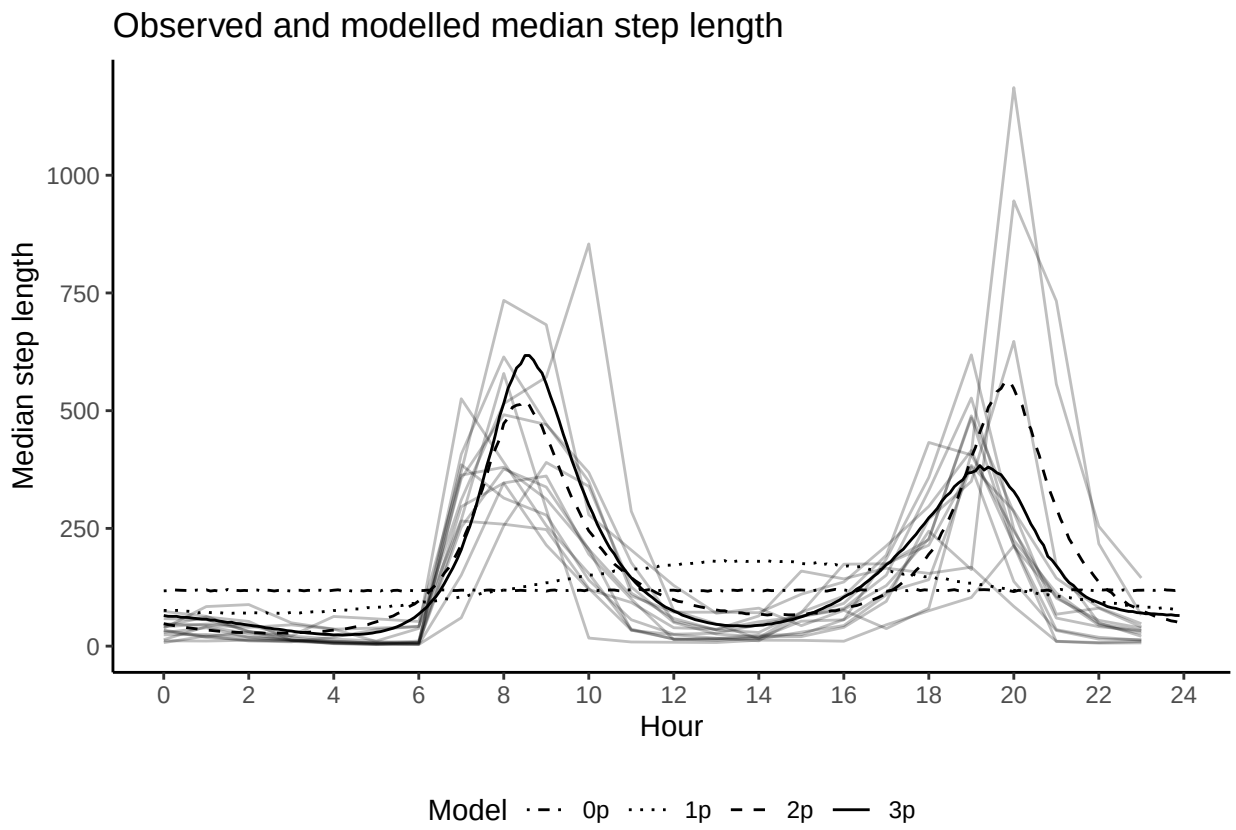
Combining the plots

Movement parameters

```
# combing gamma_dfs to plot
gamma_df <- rbind(gamma_df_0p, gamma_df_1p, gamma_df_2p, gamma_df_3p)
gamma_df <- gamma_df |> mutate(model_f = as.numeric(factor(model)))

mean_sl <- ggplot() +
  geom_path(data = movement_summary, aes(x = hour, y = mean_sl, group = factor(id)), alpha = 0.25) +
  geom_path(data = gamma_df, aes(x = hour, y = mean, linetype = model)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Mean step length") +
  # scale_colour_viridis_d("Harmonics", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
  ggtitle("Observed and modelled mean step length") +
  theme_classic() +
  theme(legend.position = "bottom")

mean_sl
```



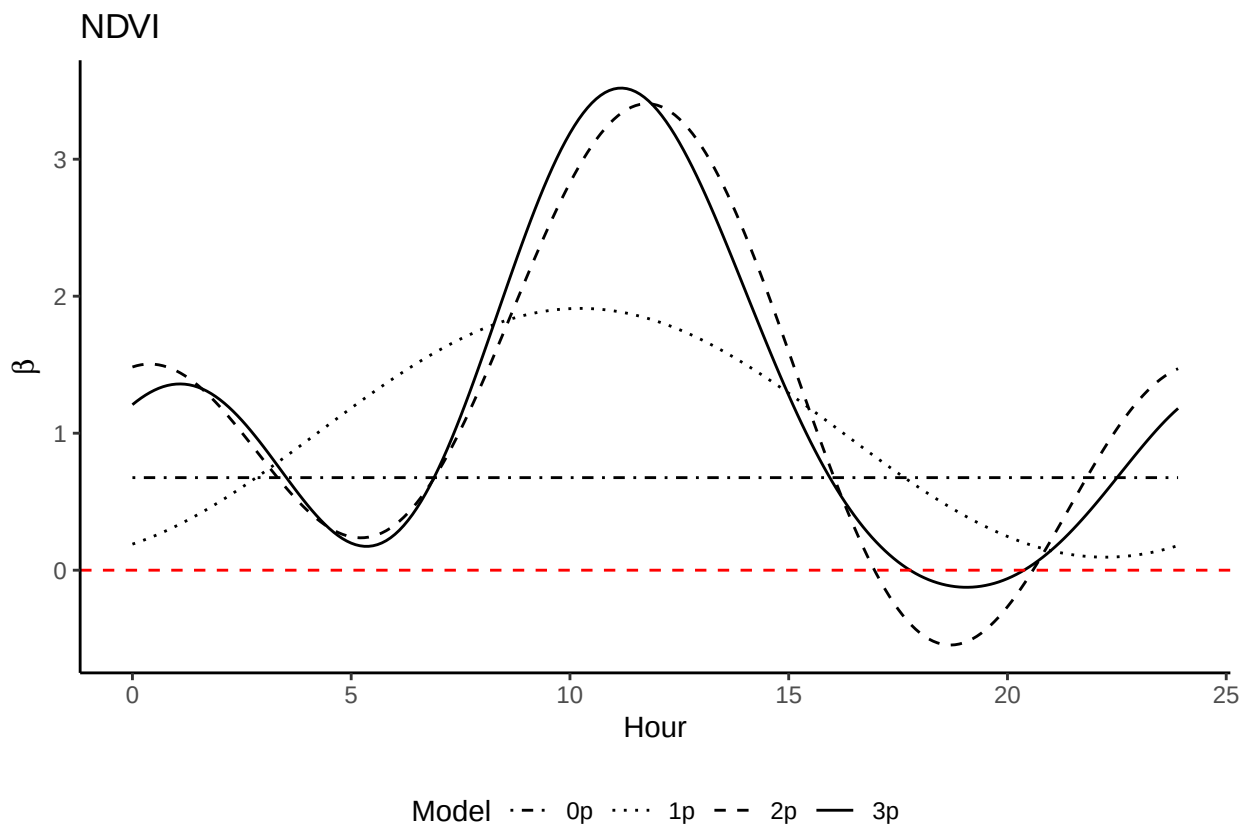

```
median_sl <- ggplot() +
  geom_path(data = movement_summary, aes(x = hour, y = median_sl, group = factor(id)), alpha = 0.25) +
  geom_path(data = gamma_df, aes(x = hour, y = median, linetype = model)) +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  scale_y_continuous("Median step length") +
  # scale_colour_viridis_d("Harmonics", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
  ggtitle("Observed and modelled median step length") +
  theme_classic() +
  theme(legend.position = "bottom")

median_sl
```



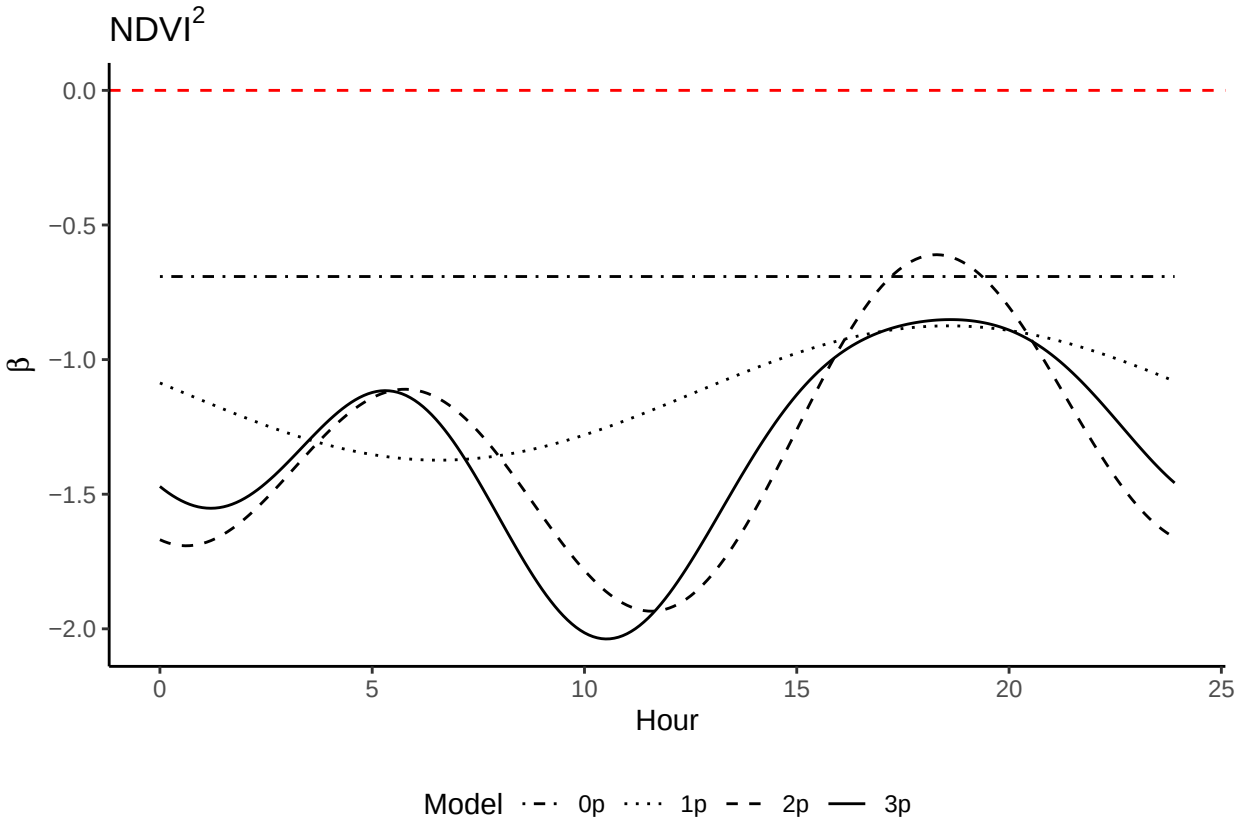
External selection

```
harmonics_scaled_long_0p <- harmonics_scaled_long_0p |> mutate(model = "0p")
harmonics_scaled_long_1p <- harmonics_scaled_long_1p |> mutate(model = "1p")
harmonics_scaled_long_2p <- harmonics_scaled_long_2p |> mutate(model = "2p")
harmonics_scaled_long_3p <- harmonics_scaled_long_3p |> mutate(model = "3p")

harmonics_scaled_long_Mp <- rbind(harmonics_scaled_long_0p, harmonics_scaled_long_1p, harmonics_scaled_l

coef_titles <- unique(harmonics_scaled_long_Mp$coef)


ndvi_harms <- ggplot() +
      geom_path(data = harmonics_scaled_long_Mp %>%
                filter(coef == "ndvi"),
                aes(x = hour, y = value, linetype = model)) +
      geom_hline(yintercept = 0, linetype = "dashed", colour = "red") +
      scale_y_continuous(expression(beta)) +
      scale_x_continuous("Hour") +
      # scale_color_viridis_d("Estimate", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
      ggtitle("NDVI") +
      theme_classic() +
      theme(legend.position = "bottom")
```

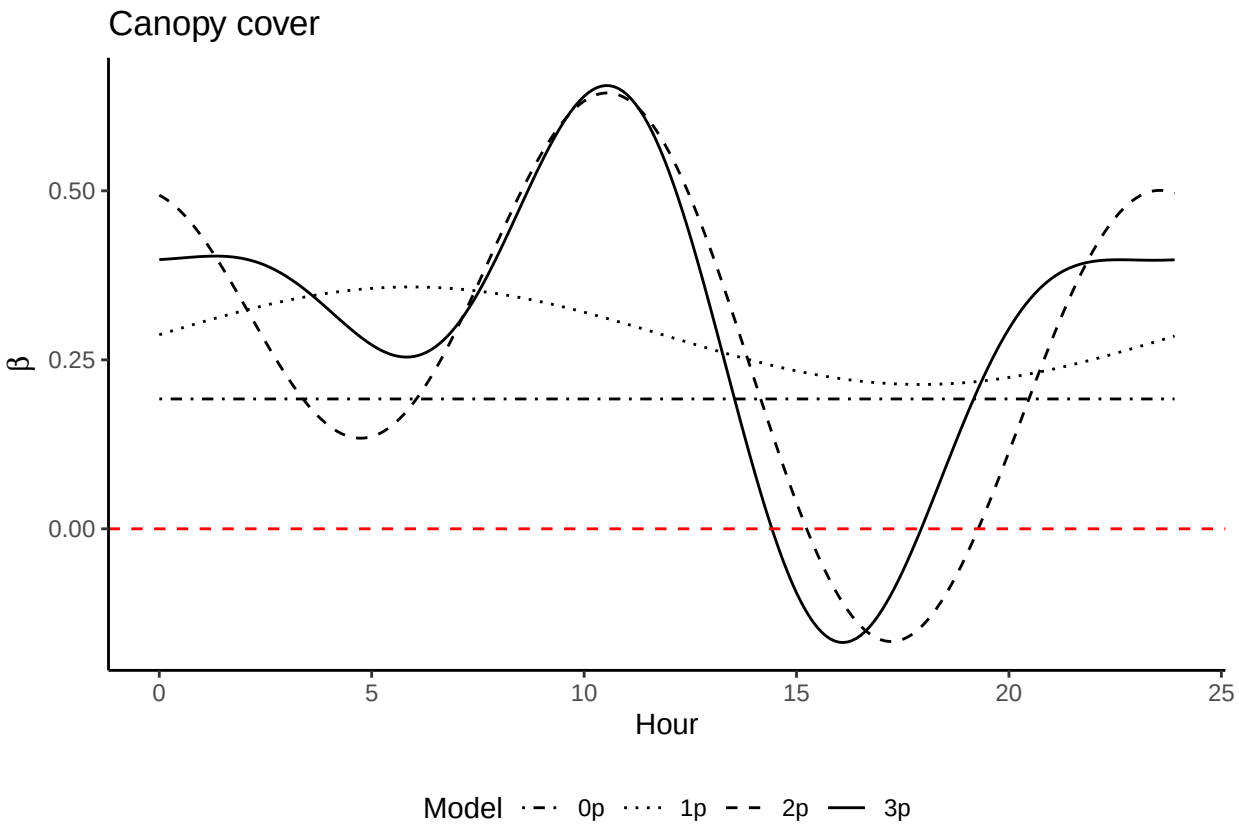
```
ndvi_harms
```



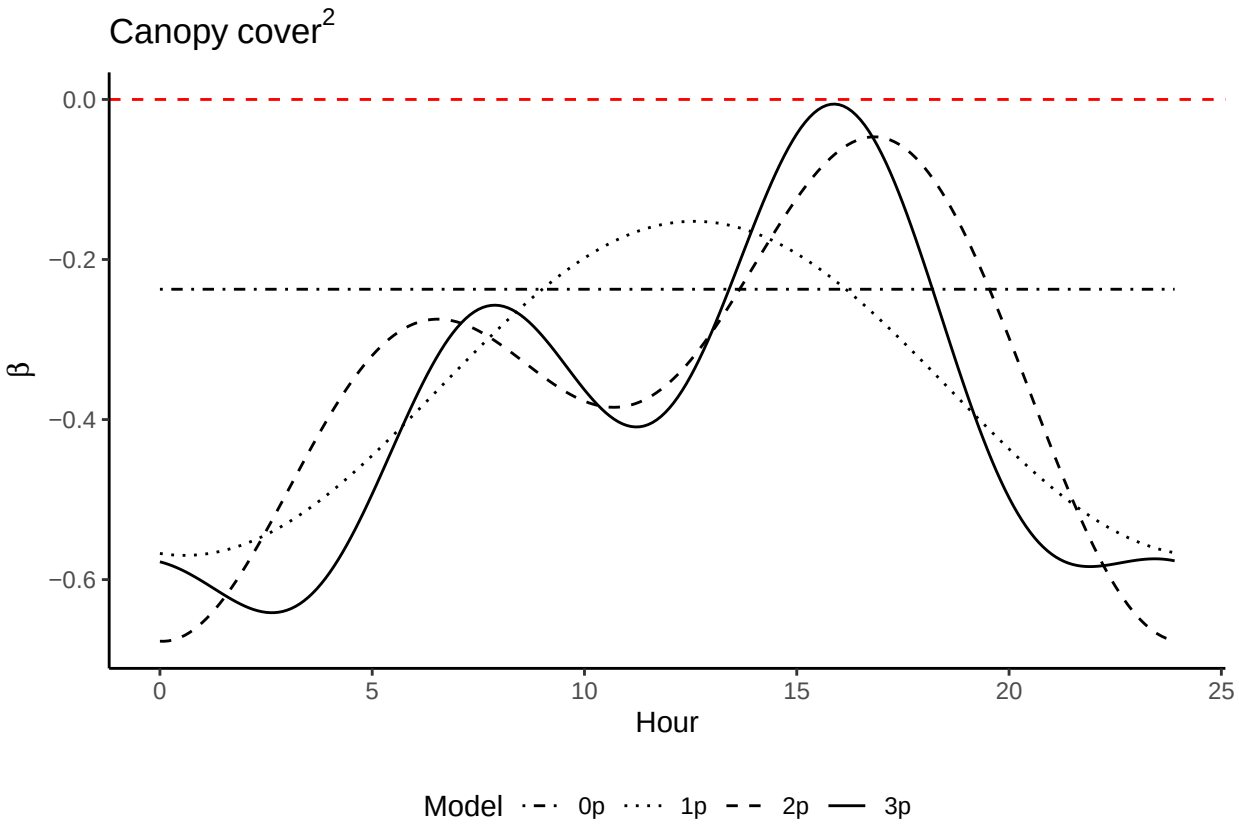

```
ndvi_2_harms <- ggplot() +
      geom_path(data = harmonics_scaled_long_Mp %>%
                filter(coef == "ndvi_2"),
                aes(x = hour, y = value, linetype = model)) +
      geom_hline(yintercept = 0, linetype = "dashed", colour = "red") +
      scale_y_continuous(expression(beta)) +
      scale_x_continuous("Hour") +
      # scale_color_viridis_d("Estimate", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
      ggtitle(expression(NDVI^2)) +
      theme_classic() +
      theme(legend.position = "bottom")

ndvi_2_harms
```



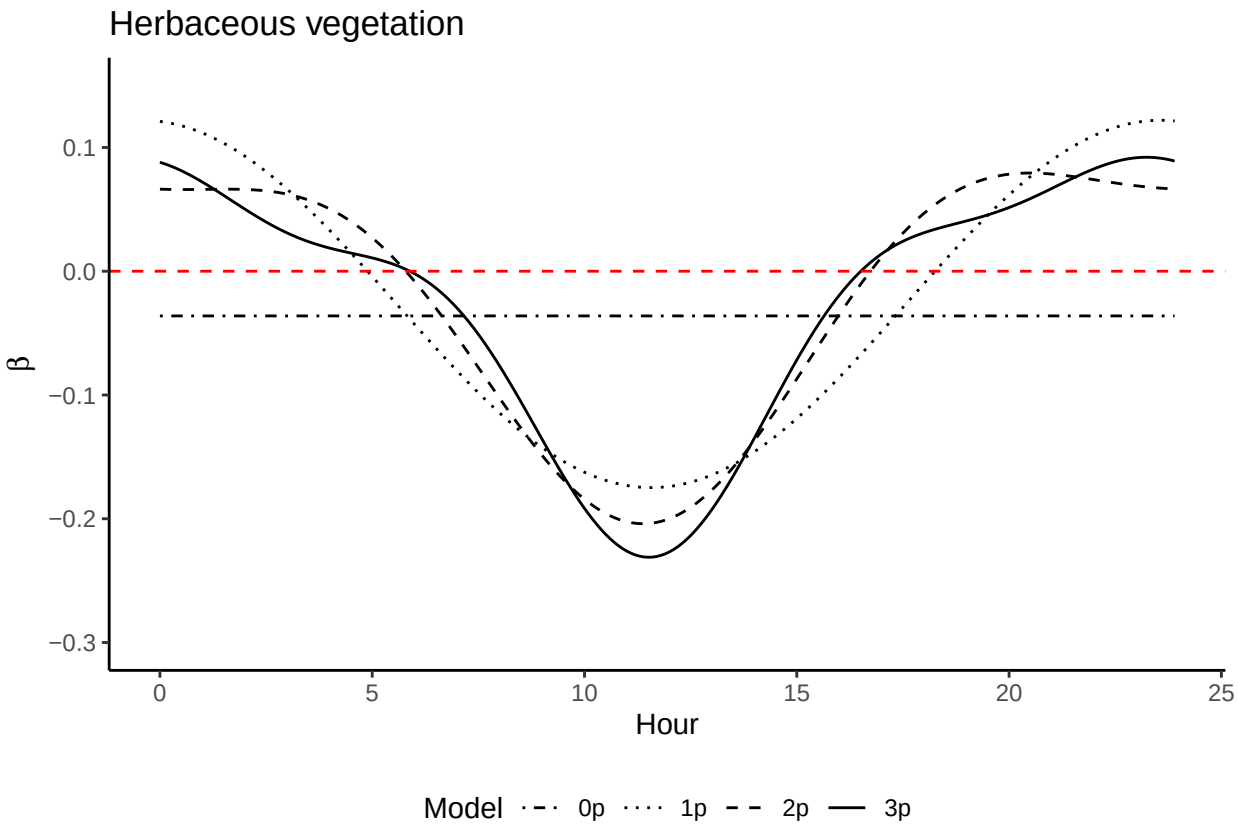

```
canopy_harms <- ggplot() +
      geom_path(data = harmonics_scaled_long_Mp %>%
                filter(coef == "canopy"),
                aes(x = hour, y = value, linetype = model)) +
      geom_hline(yintercept = 0, linetype = "dashed", colour = "red") +
      scale_y_continuous(expression(beta)) +
      scale_x_continuous("Hour") +
      # scale_color_viridis_d("Estimate", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
      ggtitle("Canopy cover") +
      theme_classic() +
      theme(legend.position = "bottom")

canopy_harms
```



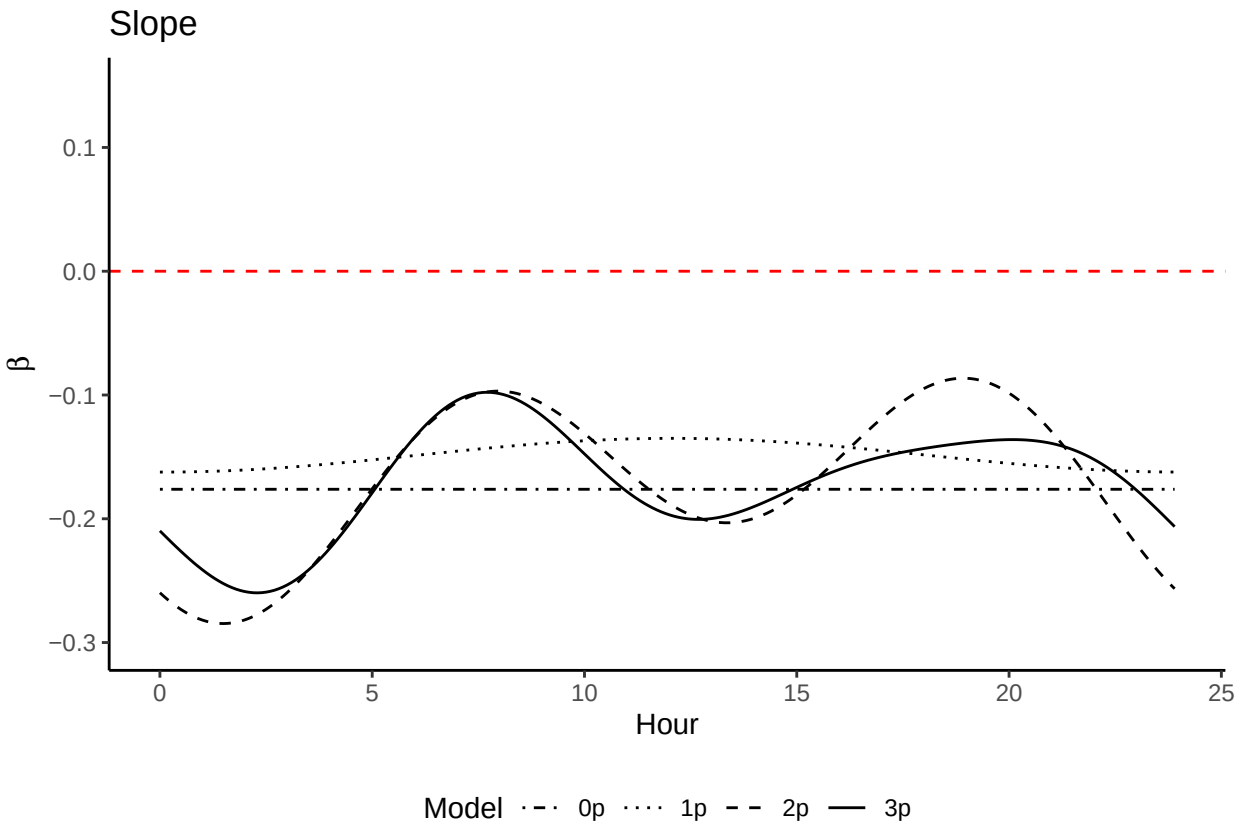

```
canopy_2_harms <- ggplot() +
      geom_path(data = harmonics_scaled_long_Mp %>%
                filter(coef == "canopy_2"),
                aes(x = hour, y = value, linetype = model)) +
      geom_hline(yintercept = 0, linetype = "dashed", colour = "red") +
      scale_y_continuous(expression(beta)) +
      scale_x_continuous("Hour") +
      # scale_color_viridis_d("Estimate", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
      ggtitle(expression(Canopy~cover^2)) +
      theme_classic() +
      theme(legend.position = "bottom")

canopy_2_harms
```



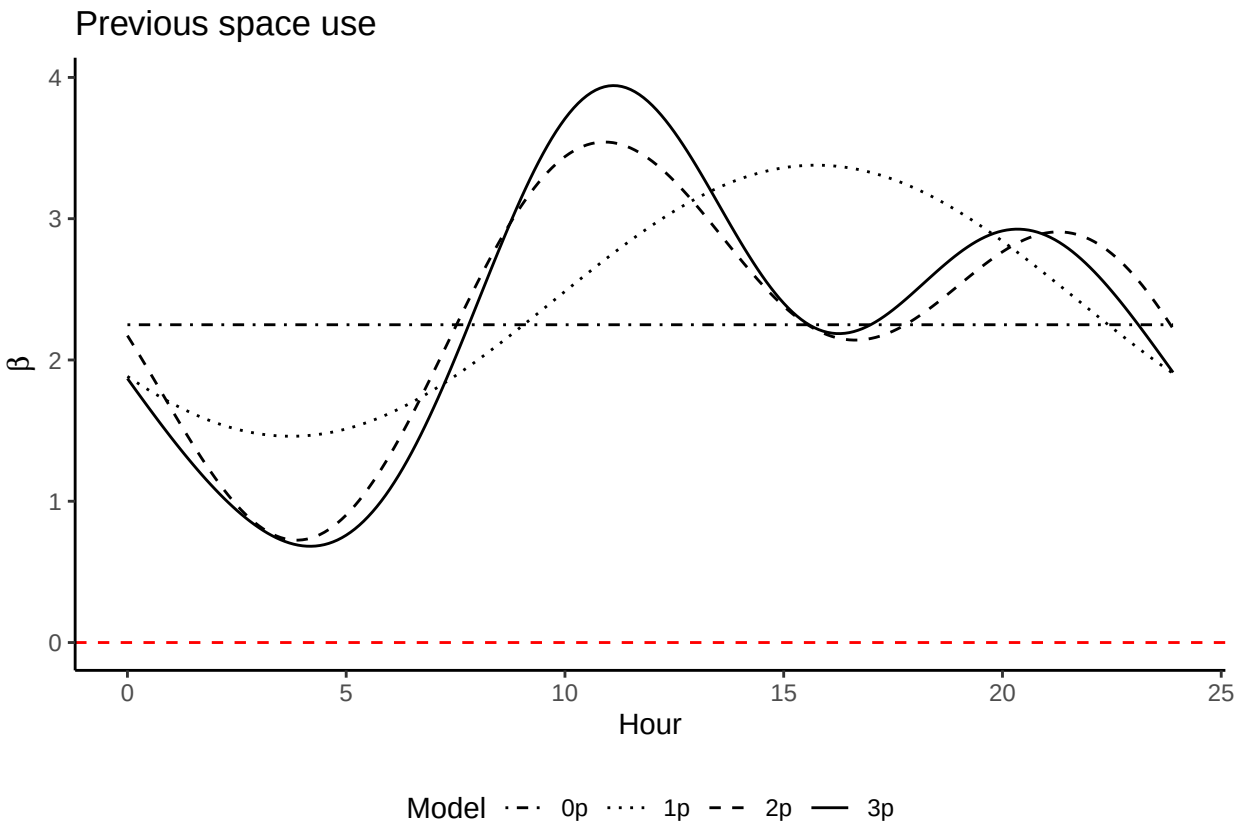

```
herby_harms <- ggplot() +
      geom_path(data = harmonics_scaled_long_Mp %>%
                filter(coef == "herby"),
                aes(x = hour, y = value, linetype = model)) +
      geom_hline(yintercept = 0, linetype = "dashed", colour = "red") +
      scale_y_continuous(expression(beta), limits = c(-0.3,0.15)) +
      scale_x_continuous("Hour") +
      # scale_color_viridis_d("Estimate", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
      ggtitle("Herbaceous vegetation") +
      theme_classic() +
      theme(legend.position = "bottom")

herby_harms
```



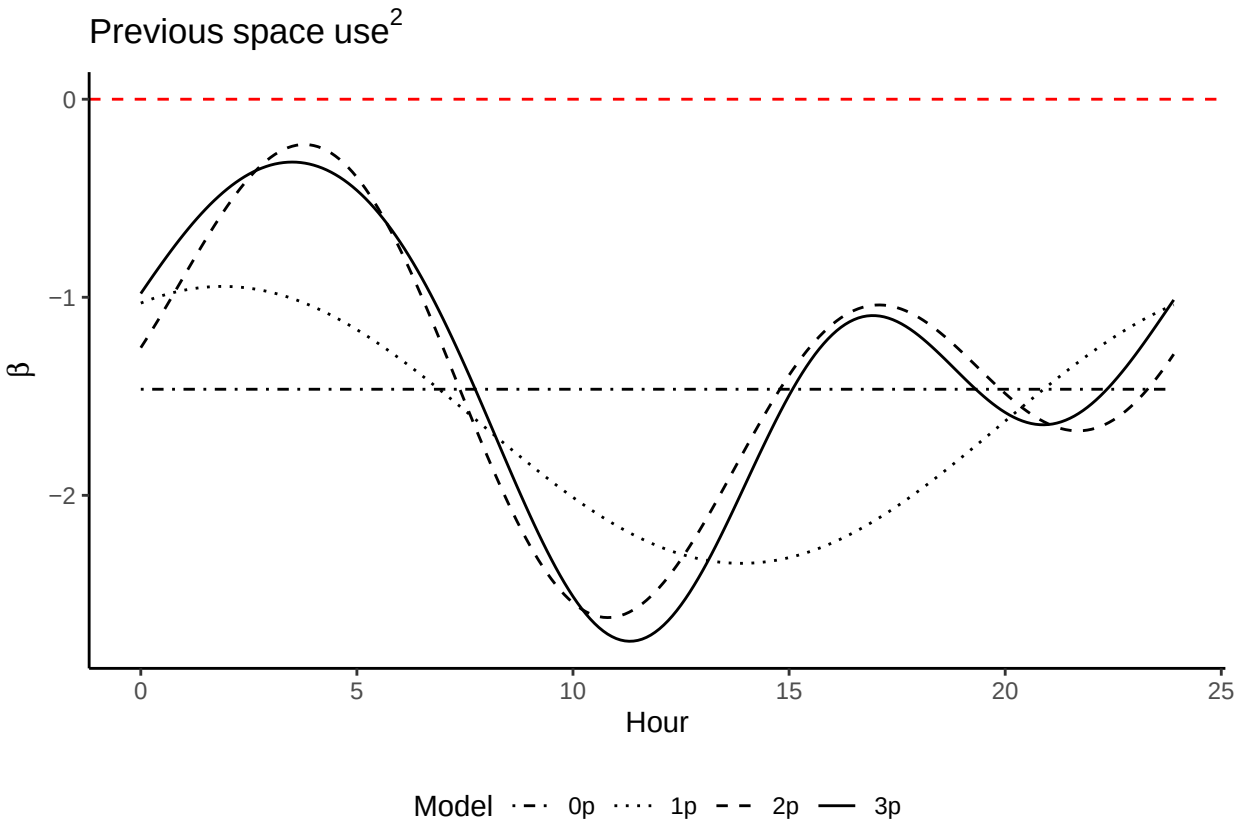

```
slope_harms <- ggplot() +
      geom_path(data = harmonics_scaled_long_Mp %>%
                filter(coef == "slope"),
                aes(x = hour, y = value, linetype = model)) +
      geom_hline(yintercept = 0, linetype = "dashed", colour = "red") +
      scale_y_continuous(expression(beta), limits = c(-0.3,0.15)) +
      scale_x_continuous("Hour") +
      # scale_color_viridis_d("Estimate", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
      ggtitle("Slope") +
      theme_classic() +
      theme(legend.position = "bottom")

slope_harms
```



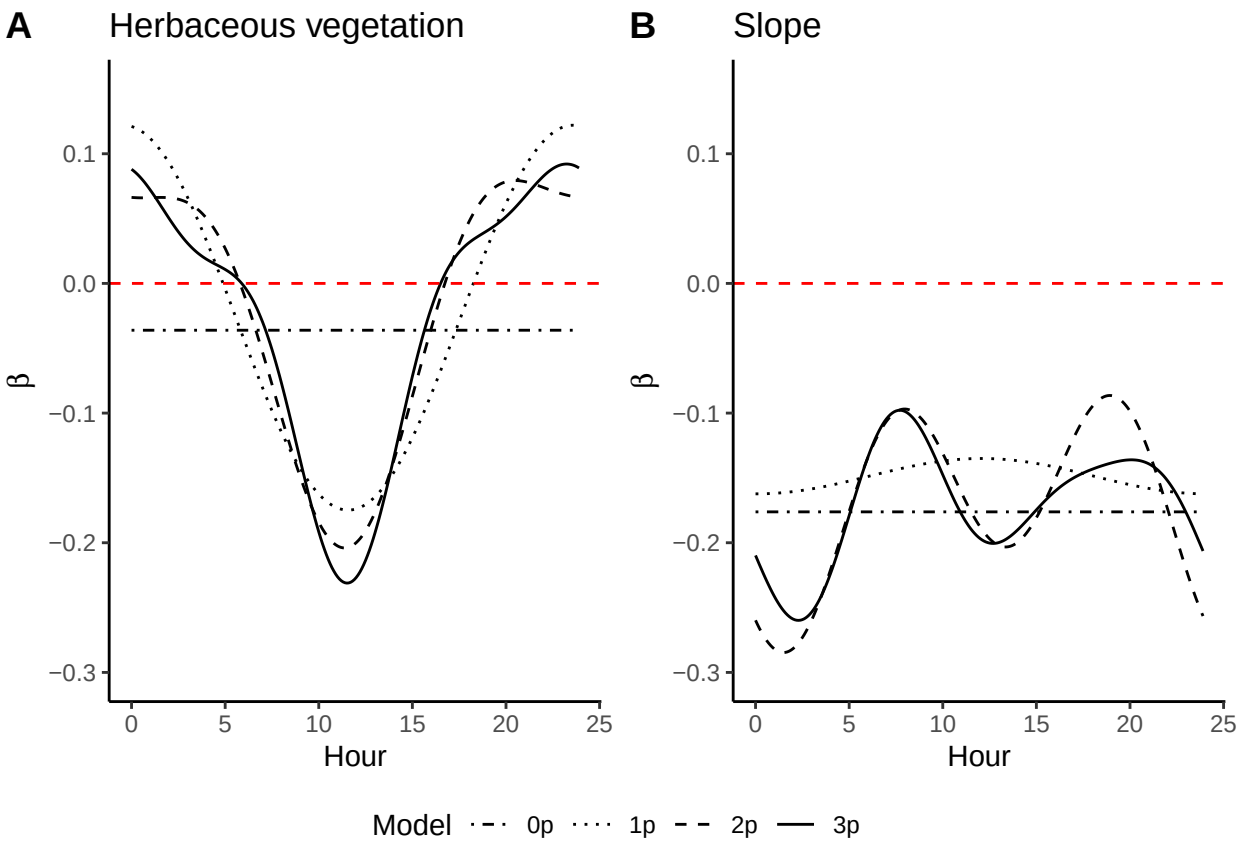

```
memory_harms <- ggplot() +
      geom_path(data = harmonics_scaled_long_Mp %>%
                filter(coef == "memory"),
                aes(x = hour, y = value, linetype = model)) +
      geom_hline(yintercept = 0, linetype = "dashed", colour = "red") +
      scale_y_continuous(expression(beta)) +
      scale_x_continuous("Hour") +
      # scale_color_viridis_d("Estimate", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
      ggtitle("Previous space use") +
      theme_classic() +
      theme(legend.position = "bottom")

memory_harms
```




```
memory_2_harms <- ggplot() +
      geom_path(data = harmonics_scaled_long_Mp %>%
                filter(coef == "memory_2"),
                aes(x = hour, y = value, linetype = model)) +
      geom_hline(yintercept = 0, linetype = "dashed", colour = "red") +
      scale_y_continuous(expression(beta)) +
      scale_x_continuous("Hour") +
      # scale_color_viridis_d("Estimate", direction = -1) +
  scale_linetype_manual("Model", breaks=c("0p","1p", "2p", "3p"), values=c(4,3,2,1)) +
      ggtitle(expression(Previous~space~use^2)) +
      theme_classic() +
      theme(legend.position = "bottom")

memory_2_harms
```

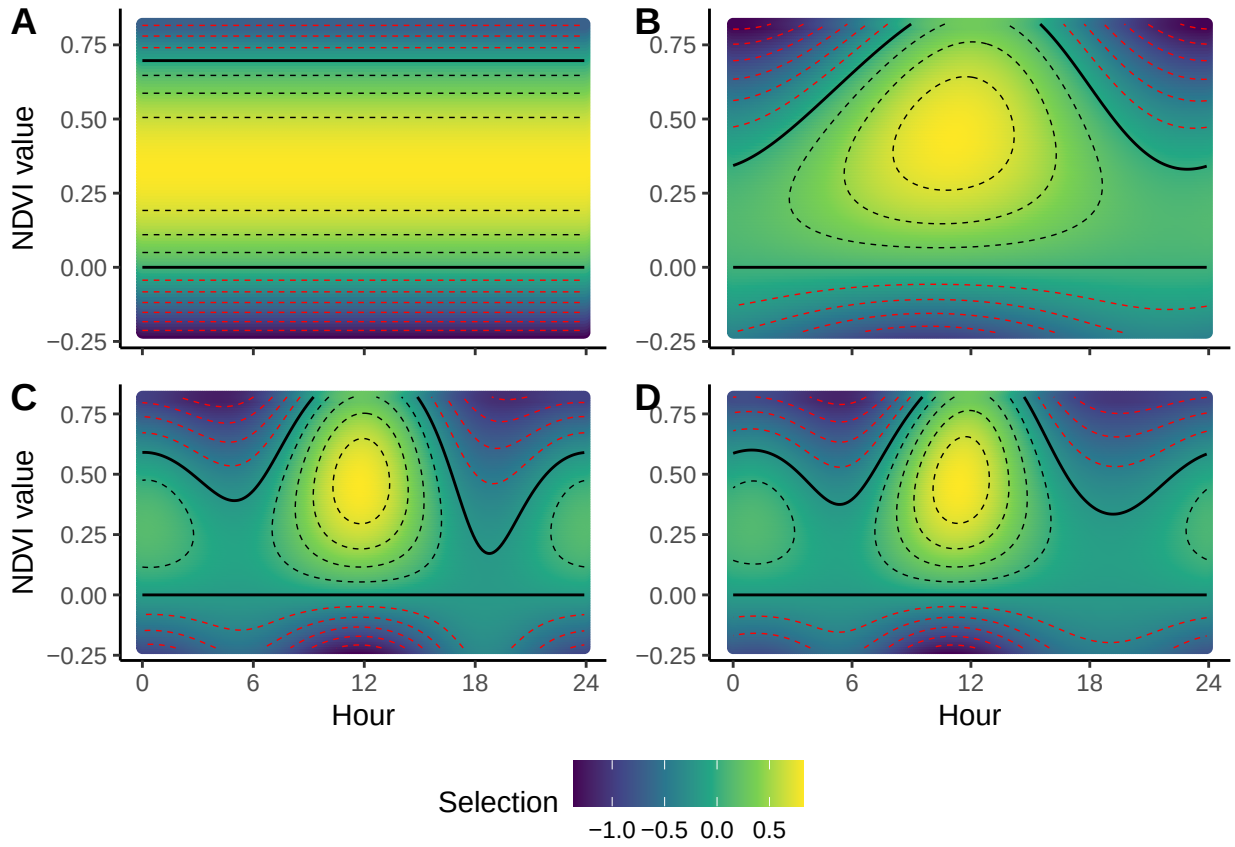



```
ggarrange(herby_harms,
          slope_harms,
          labels = c("A", "B"),
          ncol = 2, nrow = 1,
          align = "hv",
          legend = "bottom",
          common.legend = TRUE)
```

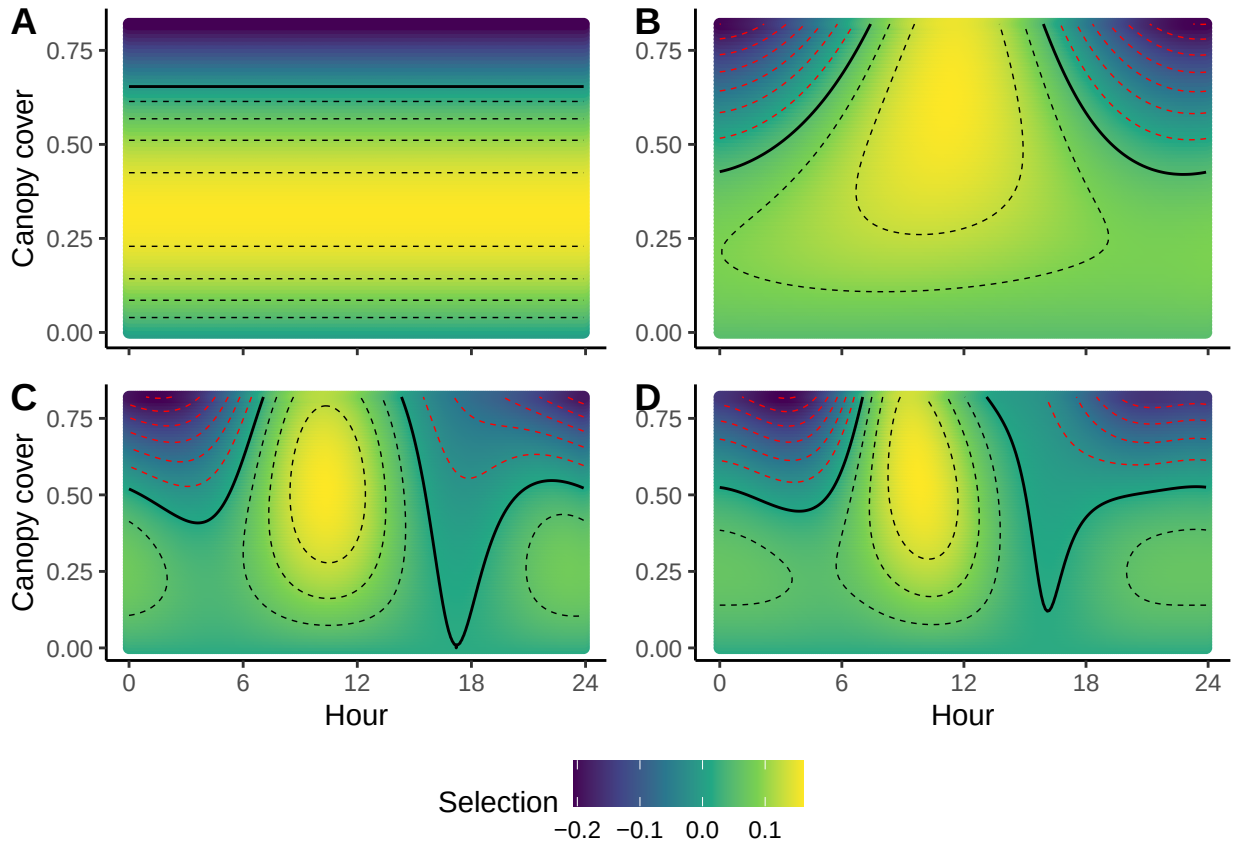


Combining selection surfaces

```
ggarrange(ndvi_quad_0p + theme(plot.title = element_blank(),
                               axis.title.x = element_blank(),
                               axis.text.x = element_blank()),

          ndvi_quad_1p + theme(plot.title = element_blank(),
                               axis.title.x = element_blank(),
                               axis.text.x = element_blank(),
                               axis.title.y = element_blank(),
                               ),

          ndvi_quad_2p,

          ndvi_quad_3p + theme(plot.title = element_blank(),
                               axis.title.y = element_blank(),
                               ),

          labels = c("A", "B", "C", "D"),
          ncol = 2, nrow = 2,
          legend = "bottom",
          common.legend = TRUE)
```

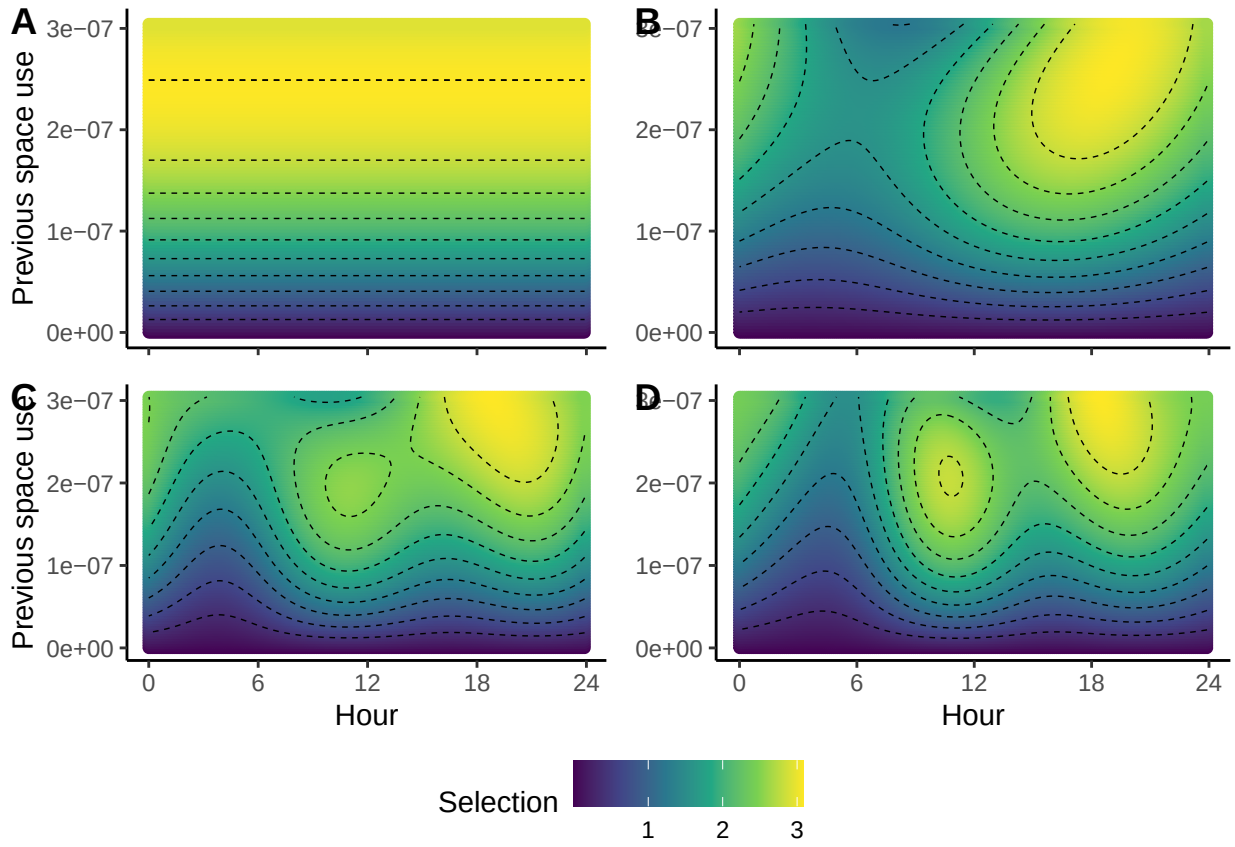



```
ggsave(
  paste0("outputs/plots/clr_fitting/NDVI_2x2_CLR_TS_daily_Mp_memory1000ALLoptim_GvM_10rs_",
         Sys.Date(), ".pdf"),
  width=150, height=120, units="mm", dpi = 300)


ggarrange(canopy_quad_0p + theme(plot.title = element_blank(),
                                 axis.title.x = element_blank(),
                                 axis.text.x = element_blank()),

          canopy_quad_1p + theme(plot.title = element_blank(),
                                 axis.title.x = element_blank(),
                                 axis.text.x = element_blank(),
                                 axis.title.y = element_blank(),
                                 ),

          canopy_quad_2p,

          canopy_quad_3p + theme(plot.title = element_blank(),
                                 axis.title.y = element_blank(),
                                 ),

          labels = c("A", "B", "C", "D"),
          ncol = 2, nrow = 2,
          legend = "bottom",
          common.legend = TRUE)
```




```
ggsave(
  paste0("outputs/plots/clr_fitting/canopy_2x2_CLR_TS_daily_Mp_memory1000ALLoptim_GvM_10rs_",
         Sys.Date(), ".pdf"),
  width=150, height=120, units="mm", dpi = 300)


ggarrange(memory_quad_0p + theme(plot.title = element_blank(),
                                 axis.title.x = element_blank(),
                                 axis.text.x = element_blank()),

          memory_quad_1p + theme(plot.title = element_blank(),
                                 axis.title.x = element_blank(),
                                 axis.text.x = element_blank(),
                                 axis.title.y = element_blank(),
                                 ),

          memory_quad_2p,

          memory_quad_3p + theme(plot.title = element_blank(),
                                 axis.title.y = element_blank(),
                                 ),

          labels = c("A", "B", "C", "D"),
          ncol = 2, nrow = 2,
          legend = "bottom",
          common.legend = TRUE)
```

```
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```




```
ggsave(
  paste0("outputs/plots/clr_fitting/memory_2x2_CLR_TS_daily_Mp_memory1000ALLoptim_GvM_10rs_",
         Sys.Date(), ".pdf"),
  width=150, height=120, units="mm", dpi = 300)
```

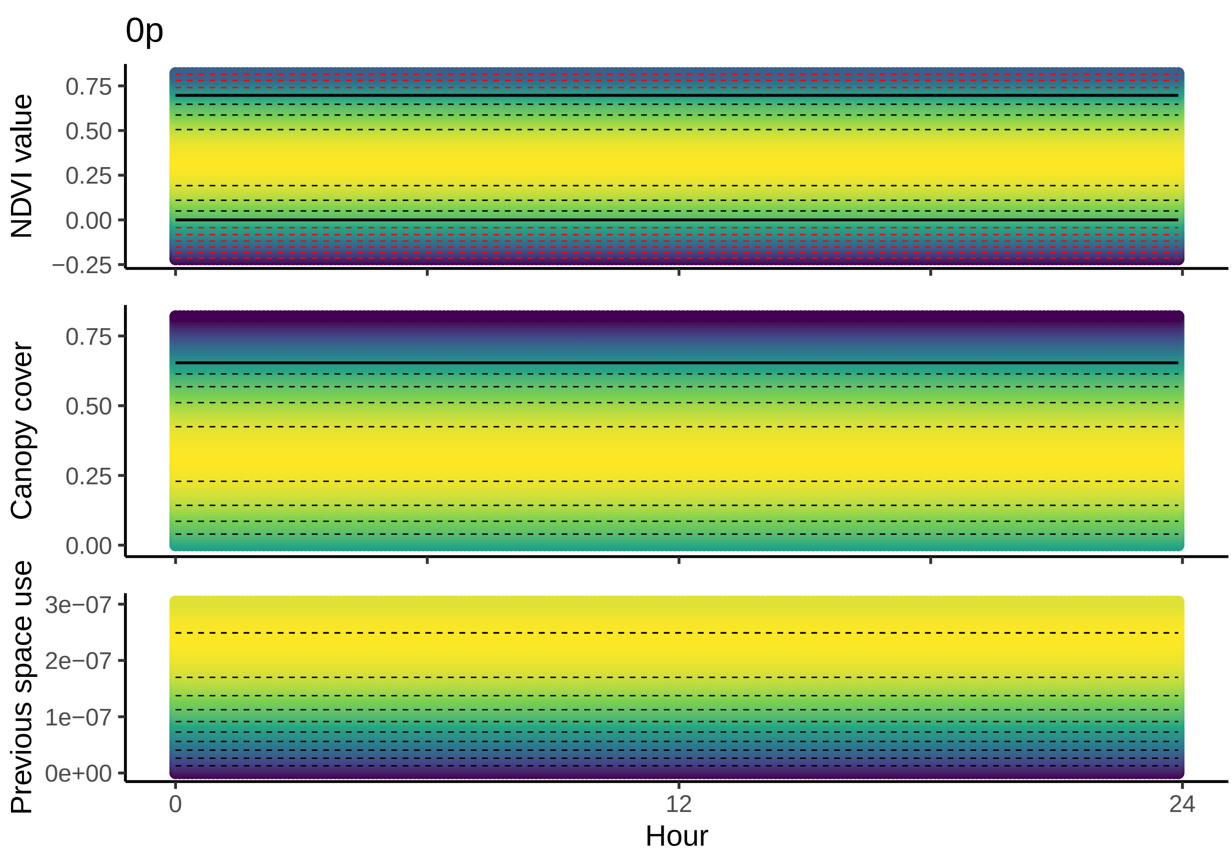
Adding all selection surfaces to the same plot

```
surface_plots_0p <- ggarrange(ndvi_quad_0p +
            ggtitle("0p") +
            theme(axis.title.x = element_blank(),
                  axis.text.x = element_blank()),

          canopy_quad_0p +
            theme(plot.title = element_blank(),
                  axis.title.x = element_blank(),
                  axis.text.x = element_blank()),

          memory_quad_0p +
            scale_x_continuous("Hour", breaks = c(0,12,24)) +
            theme(plot.title = element_blank()),

          ncol = 1, nrow = 3,
          align = "v",
          legend = "none",
          common.legend = TRUE)
```

```
## Scale for x is already present.
```

```
## Adding another scale for x, which will replace the existing scale.
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```

```
surface_plots_0p
```

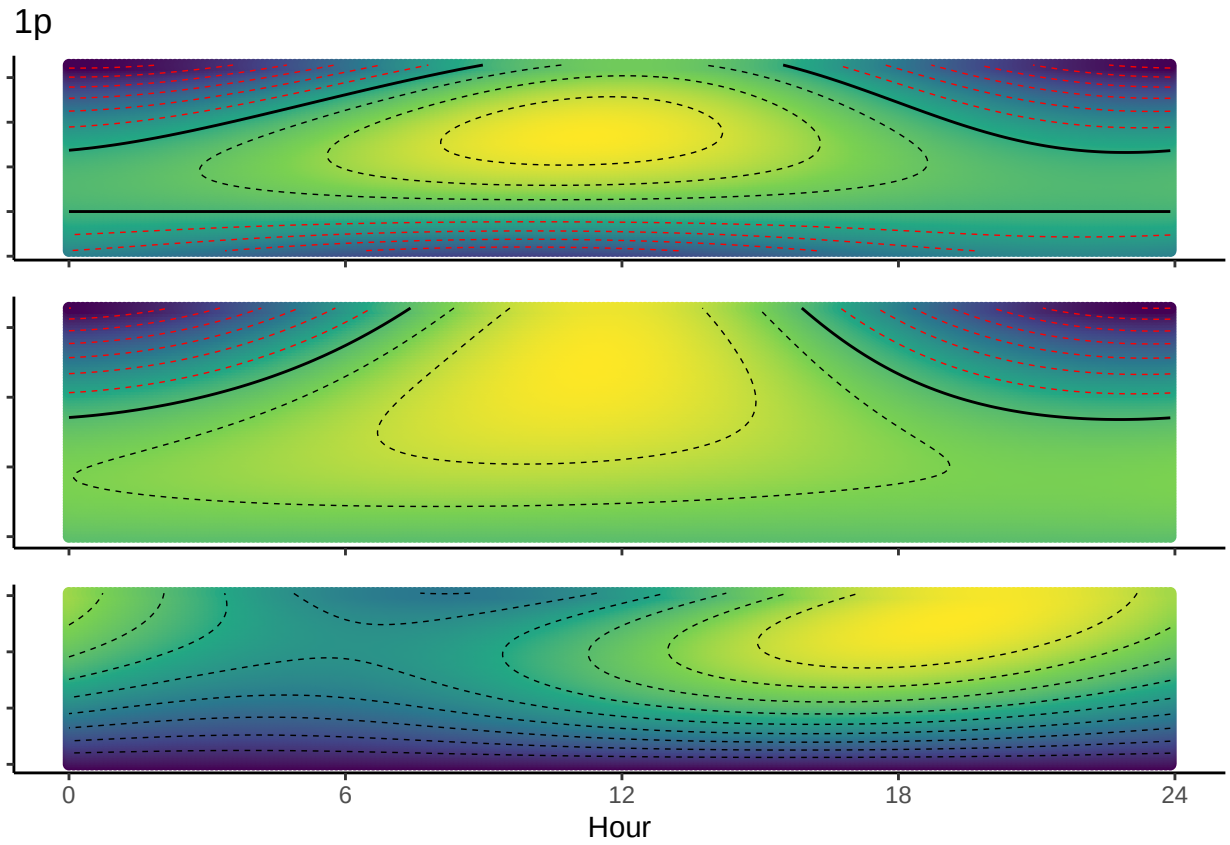



```
surface_plots_1p <- ggarrange(ndvi_quad_1p +
            ggtitle("1p") +
            theme(axis.title.x = element_blank(),
                  axis.text.x = element_blank(),
                  axis.title.y = element_blank(),
```

```
                  axis.text.y = element_blank()),

          canopy_quad_1p +
            theme(plot.title = element_blank(),
                  axis.title.x = element_blank(),
                  axis.text.x = element_blank(),
                  axis.title.y = element_blank(),
                  axis.text.y = element_blank()),

          memory_quad_1p +
            theme(plot.title = element_blank(),
                  axis.title.y = element_blank(),
                  axis.text.y = element_blank()),

          ncol = 1, nrow = 3,
          align = "v",
          legend = "none",
          common.legend = TRUE)
```

```
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: ‘stat_contour()‘: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```

```
surface_plots_1p
```



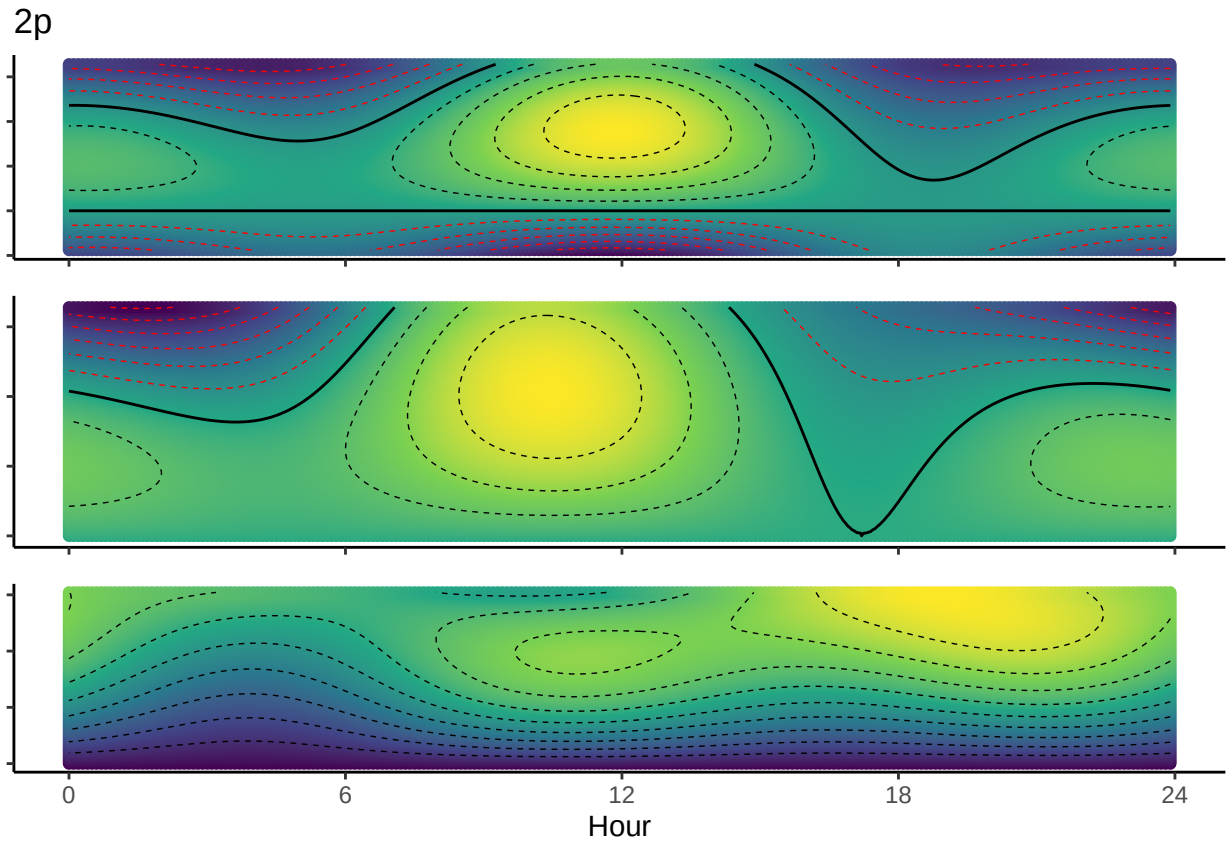

```
surface_plots_2p <- ggarrange(ndvi_quad_2p +
            ggtitle("2p") +
            theme(axis.title.x = element_blank(),
                  axis.text.x = element_blank(),
                  axis.title.y = element_blank(),
                  axis.text.y = element_blank()),

          canopy_quad_2p +
            theme(plot.title = element_blank(),
                  axis.title.x = element_blank(),
                  axis.text.x = element_blank(),
                  axis.title.y = element_blank(),
                  axis.text.y = element_blank()),

          memory_quad_2p +
            theme(plot.title = element_blank(),
                  axis.title.y = element_blank(),
                  axis.text.y = element_blank()),

          ncol = 1, nrow = 3,
          align = "v",
          legend = "none",
          common.legend = TRUE)
```

```
## Warning: `stat_contour()`: Zero contours were generated
```

```
## Warning in min(x): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```

```
surface_plots_2p
```

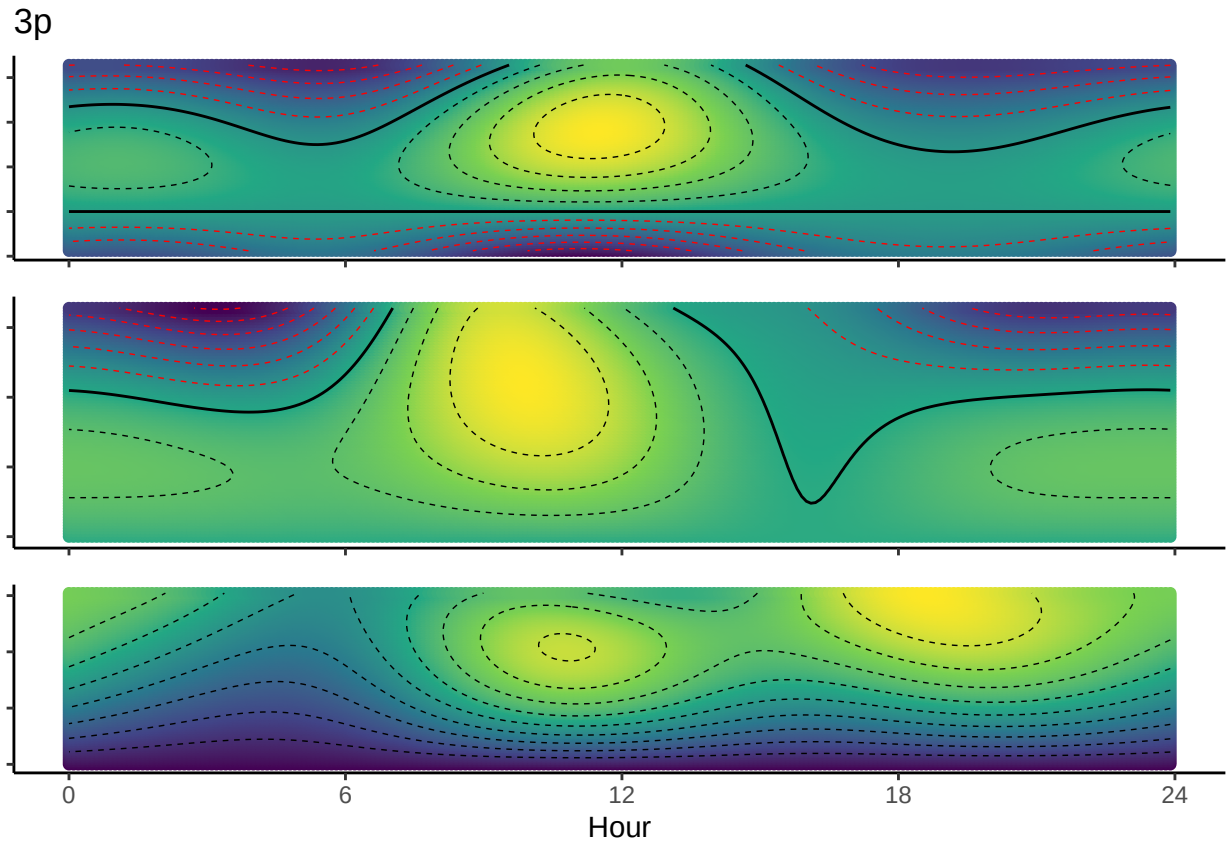



```
surface_plots_3p <- ggarrange(ndvi_quad_3p +
             ggtitle("3p") +
               theme(axis.title.x = element_blank(),
                     axis.text.x = element_blank(),
                     axis.title.y = element_blank(),
                     axis.text.y = element_blank()),

            canopy_quad_3p +
              theme(plot.title = element_blank(),
```

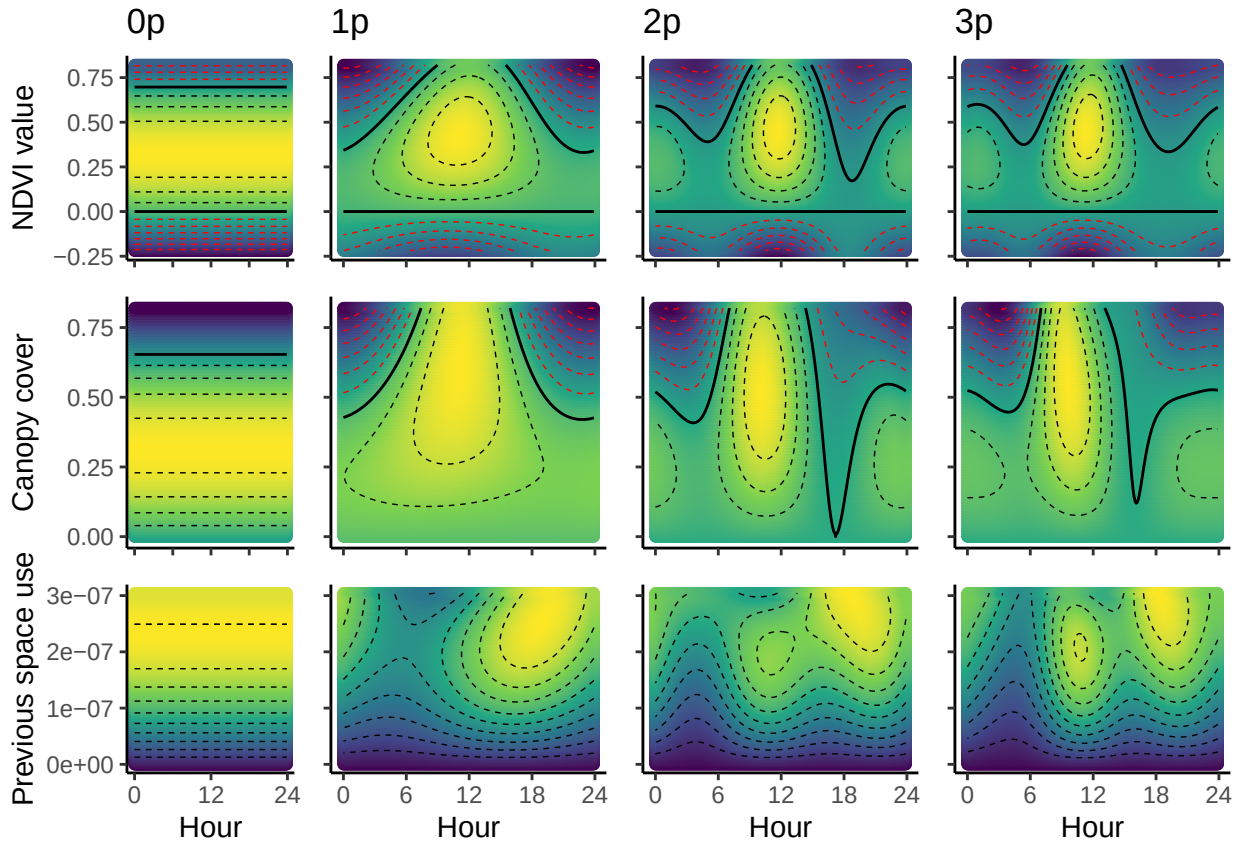
```r
                   axis.title.x = element_blank(),
                   axis.text.x = element_blank(),
                   axis.title.y = element_blank(),
                   axis.text.y = element_blank()),

           memory_quad_3p +
            theme(plot.title = element_blank(),
                  axis.title.y = element_blank(),
                  axis.text.y = element_blank()),

          ncol = 1, nrow = 3,
          align = "v",
          legend = "none",
          common.legend = TRUE)
```

```
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
## Warning: `stat_contour()`: Zero contours were generated
## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```

```r
surface_plots_3p
```




```
ggarrange(surface_plots_0p, surface_plots_1p, surface_plots_2p, surface_plots_3p,
          ncol = 4, nrow = 1,
          legend = "bottom",
          legend.grob = get_legend(ndvi_quad_2p)
          )
```




```
ggsave(
  paste0("outputs/plots/clr_fitting/all_quad_4x1_CLR_TS_daily_Mp_memory1000ALLoptim_GvM_10rs_",
         Sys.Date(), ".pdf"),
  width=150, height=110, units="mm", dpi = 300)
```

References

Session info

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19045)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_New Zealand.utf8  LC_CTYPE=English_New Zealand.utf8    LC_MONETARY=English_Ne
## [4] LC_NUMERIC=C                         LC_TIME=English_New Zealand.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] formatR_1.14        scales_1.2.1        glmmTMB_1.1.8       clogitL1_1.5        Rcpp_1.0.10
```

```
##  [6] ecospat_3.5         TwoStepCLogit_1.2.5 survival_3.5-5      viridis_0.6.2       viridisLite_0.4
## [11] matrixStats_1.0.0   patchwork_1.1.2     ggpubr_0.6.0        adehabitatHR_0.4.21 adehabitatLT_0.3
## [16] CircStats_0.2-6     boot_1.3-28.1       MASS_7.3-59         adehabitatMA_0.3.16 ade4_1.7-22
## [21] sp_1.6-0            ks_1.14.0           beepr_1.3           tictoc_1.2          terra_1.7-23
## [26] amt_0.2.1.0         lubridate_1.9.2     forcats_1.0.0       stringr_1.5.0       dplyr_1.1.2
## [31] purrr_1.0.1         readr_2.1.4         tidyr_1.3.0         tibble_3.2.1        ggplot2_3.4.2
## [36] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##   [1] backports_1.4.1        Hmisc_5.0-1            systemfonts_1.0.4      plyr_1.8.8             TM
##   [6] splines_4.2.1          TH.data_1.1-2          digest_0.6.31          foreach_1.5.2          htm
##  [11] earth_5.3.2            fansi_1.0.4            magrittr_2.0.3         checkmate_2.1.0        clu
##  [16] tzdb_0.3.0             vroom_1.6.1            sandwich_3.0-2         timechange_0.2.0       col
##  [21] textshaping_0.3.6      rbibutils_2.2.13       xfun_0.39              crayon_1.5.2           jso
##  [26] lme4_1.1-32            zoo_1.8-12             iterators_1.0.14       ape_5.7-1              glu
##  [31] PresenceAbsence_1.1.11 gtable_0.3.3           emmeans_1.8.5          car_3.1-2              abi
##  [36] mvtnorm_1.1-3          DBI_1.1.3              rstatix_0.7.2          isoband_0.2.7          ple
##  [41] xtable_1.8-4           htmlTable_2.4.1        units_0.8-1            foreign_0.8-84         bit
##  [46] proxy_0.4-27           mclust_6.0.0           Formula_1.2-5          htmlwidgets_1.6.2      RCo
##  [51] nabor_0.5.0            pkgconfig_2.0.3        reshape_0.8.9          farver_2.1.1           nnc
##  [56] sass_0.4.5             utf8_1.2.3             tidyselect_1.2.0       labeling_0.4.2         rla
##  [61] reshape2_1.4.4         munsell_0.5.0          TeachingDemos_2.12     tools_4.2.1            cac
##  [66] cli_3.6.1              generics_0.1.3         audio_0.1-10           broom_1.0.4            eva
##  [71] fastmap_1.1.1          yaml_2.3.7             ragg_1.2.5             maxnet_0.1.4           kni
##  [76] bit64_4.0.5            fitdistrplus_1.1-8     randomForest_4.7-1.1   nlme_3.1-162           pra
##  [81] biomod2_4.2-2          compiler_4.2.1         rstudioapi_0.14        e1071_1.7-13           ggs
##  [86] bslib_0.4.2            stringi_1.7.12         plotmo_3.6.2           highr_0.10             lat
##  [91] poibin_1.5             circular_0.4-95        Matrix_1.6-5           nloptr_2.0.3           cla
##  [96] permute_0.9-7          vegan_2.6-4            gbm_2.1.8.1            vctrs_0.6.2            pil
## [101] lifecycle_1.0.3        Rdpack_2.4             jquerylib_0.1.4        estimability_1.4.1     cow
## [106] data.table_1.14.8      raster_3.6-20          R6_2.5.1               KernSmooth_2.23-20     gri
## [111] codetools_0.2-19       gtools_3.9.4           withr_2.5.0            multcomp_1.4-23        mgc
## [116] parallel_4.2.1         hms_1.1.3              grid_4.2.1             rpart_4.1.19           coc
## [121] minqa_1.2.5            class_7.3-21           rmarkdown_2.21         carData_3.0-5          mda
## [126] sf_1.0-12              pROC_1.18.0            numDeriv_2016.8-1.1    base64enc_0.1-3        tin
```